# HPC survey paper: Cluster computing

**Sonali Mehta, Rushil Goomer, Alarsh Tiwari, Harsh Kataria, Souvik Mishra, Ambuje Gupta**
Department of Computer Science Engineering
Bennett University, Greater Noida, India
(sm4239, rg6277, at1693, hk9663, sm5939, ag5915)@bennett.edu.in

## Abstract

A large amount of data is coming into existence each day and the power to process this data needs cluster computers which are a set of computers that operate together, only to be viewed as a single high performance computer. As huge computational power is needed to get information from a large volume of data, it is highly recommended to process this data using parallel computing by distributing the data on different clusters. This paper presents a brief survey on clustering and its various emerging applications along with real time examples and their challenges in processing data parallely. The accessible literary works on applications under High Performance Computing are constrained and occasionally not talked about in subtleties. The data parallelization problem can be removed with the help of embedding devices such as Raspberry Pi. These devices are connected with each other with the help of Ethernet and have been given two parallelization strategies namely, OpenMP and MPI to scale them up to large grid sizes. Also the performance, power and energy efficiency obtained from Raspberry Pi clusters are compared with a well-known co-processor used in HPC.

## 1 Introduction

In the coming decade Future HPC computers will be limited to about 20 MW of power consumption. We expect to conclude this paper on a high note with hopes that the research on the same would be continued and that we will exercise maximum exploitation to the capabilities of today's hardware in the domain of High Performance Computing. A lot of progress in parallel computing was realized with the introduction of the Message Passing Interface(MPI). In today's world, we generate a lot of data on a daily basis and a single system isn't capable enough to process data of this scale. We need parallel computing to compute this data faster. Parallel computation has a lot of scope and can be used in image processing. For example we can count the number of wheat grains (Marković et al., 2018) by using image processing and to make it fast, we may parallelize this task using MPI. In the case of applications pertaining to agriculture applications, (Nasir et al., 2012), lesser computation time in image processing tasks may help plant recognition and other such algorithms implementation time to get much faster. There are constrained writings on parallel and distributed image processing for agriculture related applications which is not the case with some other applications such as those in medical fields. We have a large volume of data pertaining to plant species and we do need extensive computing for plant recolonization. Hence the overall process becomes more complex and takes a lot of time, and hence comes the need of parallellization techniques. Speaking of parallelization techniques, a major one among them is clustering. Clusters could be used for parallel computing to apply more processor force to complete the task in a more efficient manner. We may implement exemplary parallelization using this knowledge in the field of Artificial Intelligence to reduce the training time and processor load of ANNs (Sharif and Gürsoy, 2018). The implementation of such algorithms to perform the task of parallelization such that the training happens on different clusters can be conveniently performed on Java. As we may expect, we did conclude that the training time of ANNs by using different clusters is much lesser than a single machine's training time. Clusters have also found their major applications with Raspberry Pi(RPI). HighRes Flood forecasting (Singhal et al., 2013) is another domain which can benefit

highly from parallelization. Introduction of parallelism in the hydrological model simulations is done by mapping the independent watershed basins of spatial decomposition of the regions to multiple processes using MPI and employing shared memory parallelization within each process using OpenMP. A lot of research is also being conducted in the EEG domain and we need to pre-process raw EEG signals to get meaningful data. In a 32-channel EEG device, we get 256 values per second. So to process these raw EEG signals we apply techniques such as FFT, wavelet transformation etc. As there is a lot of data to process we use MPI to parallelize this data processing so that we can have our results in a comparatively faster time. There is a lot of scope for even more utilization of techniques such as clustering.

## 2 Definitions

**1. Raspberry PI -** A small low cost one board computer which also contains GPIO pins that allows you to control electronic components for IOT purposes.

**2. Xeon Phi -** is a Intel made series of x-86 many-core processors. It is made for use in supercomputers and high-end computational purposes. Its makes use of programming languages and APIs such as OpenMP.

**3. Aprioiri Algorithm -** It identifies the frequent unique items in the database and extends them to larger item sets in the database.

**4. Artificial Neural Networks(ANNs) -** ANNs are computational algorithms designed in such a manner that it tries to replicate the neurons present in our brain cells. It contains large number of interconnected nodes which are used to process and feed the information in the network.

**5. Clustering -** A computer cluster that distributes workload for data analysis across different cluster nodes (i.e computers) that work together to process the given task in parallel.

**6. Beowulf cluster -** A beowulf cluster is a computer cluster made out of commodity grade products to increase performance of a single task in a distributed environment mimicking and performing very much like a supercomputer.

**7. HSI -** It is a colour space that represents colours similarly how the human eye see's them.

**8. Moore's law -** It says that with time, computing will increase by increasing the number of transistors and the cost will decrease. This law held true for a long time but is starting to diminish as we are developing and adding more and more parallel systems instead of adding transistors to a single system.

**9. EEG -** EEG stands for electroencephalogram, this gives us the brain waves signal that we can use to decrypt different events related to our brain.

**8. FFT -** FFT stands for Fast Fourier Transform, it is an essential data processing technique in relation with EEG signals, images and signals in general. It is widely used on sequence data to convert it to frequency domain and back.

## 3 Applications

### 3.1 Image Processing in Agriculture Application

We shall follow the below described steps to enhance the image processing in Agricultural Applications (Nasir et al., 2012).

We need to identify the image processing operations to analyse parallelism. We need to apply parallelism in these applications using any of the three ways:- Data parallel, Task Parallel and Pipeline Parallel. We need to identify the architecture for achieving parallelism, whether we should use parallel processing hardware or a distributed system.

**A. Data Parallel**

In this particular approach, we divide data into Computing Units(CUs). The two major issues that need to be taken care of are data locality and load balancing. Image data is divided among all available Processing Units(PUs), and hence we can reduce communication between PUs that is not required. Each part from the taken image data shall be distributed to each Processing Unit such that each of them will get approximately the same load balancing. The figure 1 shows the division of an image into 8 parts, that is image data is divided into 8 blocks for data parallelization. This particular example is for predicting leaf population chlorophyll content from cotton leaf plants. Data parallel approach finds its major use

when there are jobs like scanning all the pixels of the image to remove certain noises. We understand that the noises would be conventionally scanned sequentially(pixel by pixel). This process would be repeated till all the pixels have been scanned. Hence it is a huge workload and becomes larger when pixel size is large. Here's where Data Parallelization comes into picture. We distribute each block with similar load to each PU. Processor 1 will scan the noises in the first block, Processor 2 will do the same in the second block and so on. We will get optimal performance if the same load to each PU is distributed. The drawback of the approach being that this distribution of blocks as such is not beyond parallel computers equipped with multiple PUs.

**B. Task Parallel**

In this, imagine a task as an individual step in image processing and imagine all different steps in it as different tasks. Now each of these different tasks are given to different PUs. Some of these tasks may depend on results from other tasks and some may be independent of others. The main challenge that task parallel approach poses for us is to determine the most efficient way for decomposition of tasks and composition of results. The figure 2 shows a picture of the cotton plant leaves where the boxed area in red color is represented as RGB(3X3 pixels). The primary objective is to predict leaf population chlorophyll content from cotton plant images. In the stages of image processing, we need to convert the image in RGB mode to IHS(intensity-hue saturation) mode. The procedure to do so is represented in equation 1.

$$I = \frac{R+G+B}{3} \tag{1}$$

$$S = 1 - \frac{3}{R+G+B}[min(R,G,B)] \tag{2}$$

$$W = cos^{-1}\left\{\frac{1/2[(R-G)+(R-B)]}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right\} \tag{3}$$

$$\begin{array}{ll} \text{H=W} & \text{If B<=G} \\ \text{H=}2\pi\text{ - W} & \text{If B>=G} \end{array}$$

We may see that the values of H, I and S depend wholly on the values of B, G and R. This is where Task Parallelization comes into the picture. It can be very well noted that the task parallel approach can be applied to this process by distribution of tasks wherein equation 1 (for calculating I) is given to processor 1, equation 2 (for calculating S) is given to processor 2 and equation 3 (for calculating H) is given to processor 3. We may appreciate the time we saved by using task parallelization rather than doing the procedure sequentially by looking at table 1. It took only 3 steps in parallel execution compared to 9 steps by the conventional process.(3X faster). The bigger pixel calculation will give an even better effect on speed performance.

**C. Pipeline Parallel**

This method is generally used when image processing is done in several stages. If there is a requirement of multiple data or multiple stages to be processed, we go with the Pipeline Parallel approach. In a nutshell, we use this when multiple data is to be processed at once. We may represent the parallelism of image processing in the Pipeline Parallel model using a simple neural network diagram as shown in figure 3. Suppose a user wishes to identify some sort of plant species, then the features from the query image(s) (I1, I2 and I3) will be compared (in distance or similarity) with the features from image data present in the database (H1, H2 and H3) to obtain respective outputs A, B and C. Now let's imagine that all the possible connections from the input layer to H1 in hidden layer(I1–H1, I2–H1, I3–H1) are categorized as task-1, all the possible connections from the input layer to H2 in hidden layer(I1–H2, I2–H2, I3–H2) are categorized as task-2 and all the possible connections from the input layer to H3 in hidden layer(I1–H3, I2–H3, I3–H3) are categorized as task-3. Each of these data is to be distributed to each PU, so that they all will have equal load balancing. Now if we try to illustrate this concept via a table representation, then we have the table 2.

We may implement such an approach in cases of classification processes. High computational rates are gained by higher degree(extent) of parallelism resulting from an arrangement of weights (or interconnections) and neurons (or processors) that allow processing of datasets of considerably large sizes in

Table 1: Serial vs parallel execution

| Serial Execution | | | Time | Parallel Execution | | | | |
|---|---|---|---|---|---|---|---|---|
| Pixel | PU1 | Step | | Pixel | PU1 | PU2 | PU3 | Step |
| 1 | Calculate I | 1 | t1 | 1 | Calculate I | Calculate I | Calculate I | 1 |
| | Calculate S | 2 | t2 | 2 | Calculate S | Calculate S | Calculate S | 2 |
| | Calculate H | 3 | t3 | 3 | Calculate H | Calculate H | Calculate H | 3 |
| 2 | Calculate I | 4 | t4 | | | | No. of steps | 3 |
| | Calculate S | 5 | t5 | | | | | |
| | Calculate H | 6 | t6 | | | | | |
| 3 | Calculate I | 7 | t7 | | | | | |
| | Calculate S | 8 | t8 | | | | | |
| | Calculate H | 9 | t9 | | | | | |
| | No. of steps | 9 | | | | | | |

Table 2: Pipeline processing

| Time | Task1 on PU1 | Task2 on PU2 | Task3 on PU3 |
|---|---|---|---|
| t1 | (I1-H1) | (I3-H2) | (I2-H3) |
| t2 | (I2-H1) | (I1-H2) | (I1-H3) |
| t3 | (I3-H1) | (I2-H2) | (I3-H3) |

Table 3: Approach of parallelism and their mechanisms

| SUITABLE MECHANISMS FOR DIFFERENT APPROACHES | |
|---|---|
| Approach of Parallelism | Suitable Mechanisms |
| Data Parallel | Parallel processing hardware |
| Task Parallel | Parallel processing hardware and/or distributed systems |
| Pipeline Parallel | Distributed system |

real time. For each of the discussed approach of parallelism for agricultural applications in the image processing domain, we summarize suitable mechanisms for the same in the Table 3. Hence, after discussing and analyzing such techniques for working on image processing in a parallel fashion, we may conclude that it is very possible to apply parallelism in applications pertaining to agriculture.

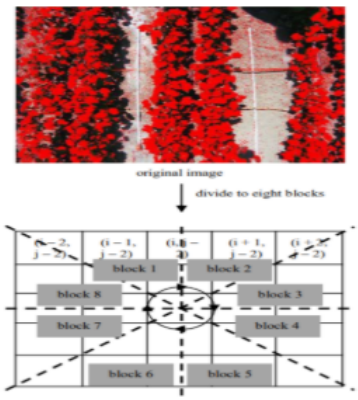


Figure 1: Omnidirectional scan filtering through 5x5 pixels grid, pixel (i,j) in the centre

### 3.2 HighResolution Operational Flood Forecasting

The process starts with setting up the static domain, a region-specific domain setup that is just a one-time setup which collects and incorporates static information that is available publicly on that particular region/area. Post this one-time setup a daily operations process is to be executed which involves process parameters such as collection of precipitation estimates, determining the surface runoff expected in accordance to the precipitation estimates that were collected and then distribution of the excess water through the flood routing engine. Finally, in periodic intervals, the verification model of the system automatically obtains actual rainfall and flood data from publicly available resources to further calibrate the model. The authors (Singhal et al., 2013) describe the basic algorithm of flood routing engine, which involves

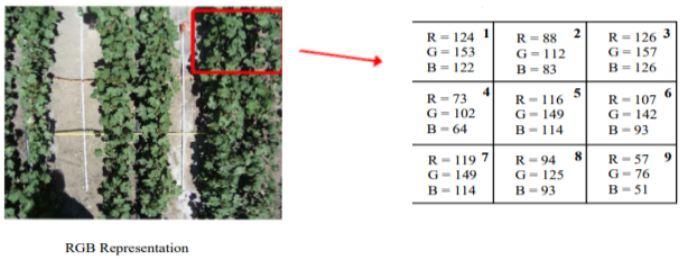


Figure 2: RGB representation

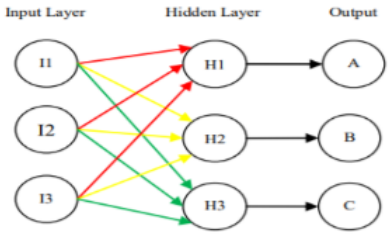


Figure 3: Neural network diagram

input of DEM and SR grids of same resolution and then using the principle of hydrostatic equilibrium, the overall height of a block, cells of which are completely filled, is computed using precipitation units mapped to ascending order of elevation blocks and if the block isn't completely filled then the height of water in that block is determined by that particular subset of cells which are filled and the rest remain at 0 height. Keeping the argument in mind that adjacent basins do not require computation of each other to be sufficient and can be considered parallel in nature, a master-slave approach is taken for distributed memory parallel processing of a basin. Further to increase parallelism the watershed basin is considered in a bounding rectangular grid which in turn is divided into rectangular tiles of 4x2 using a two step splitting process. To achieve the state of load balancing, a track of all the valid grid points are kept including any point that is in the original watershed basin and has a valid value of height. Now finally to handle race conditions, every single tile is then divided in quadrants of 2x2 size, treating them as a basic unit of work for each of the threads. Processing of the four quadrants follow an order making it so that with the execution of each quadrant synchronisation is there amongst the threads by a barrier to ensure work is isolated. The authors describe their experimental evaluation in which they had utilised two different hardware configurations both running Linux v6.4 over two real-world domains. The first domain being Brunei-MPI(they also tested hybrid OpenMP/MPI approach on this domain) and the other being Rio de Janeiro-OpenMP. The MPI implementation observed a speed-up cap for 8 or more processes due to the fact that there were 4 major basins that occupied the majority of the area. The Open MPI implementation saw a scale up of 13x on 16 threads on the Intel architecture however reduced speed-ups of the total computation time were observed suggesting that sequential processes are a larger part of the total time. The hybrid implementation saw a speed up of 10.4x in the total time instead of the previous 6.5x. However a speedup of 10.6x was observed when considering only the routing time. Fine-tuning of the respective machines would lead to somewhat better results. The parallelization techniques utilised produced good results and in some cases showed potential of very high speedups. A speedup of 13x with 16 threads using OpenMP in the Rio de Janeiro basin was observed.

### 3.3 ANN training

The paper (Sharif and Gürsoy, 2018) tries to explain the working of artificial neural networks and how we can split the training data on different clusters to achieve the goal of parallelization. Basically, there are 3 types of layers in ANNs - first is the input layer, then comes the hidden layer and final layer is

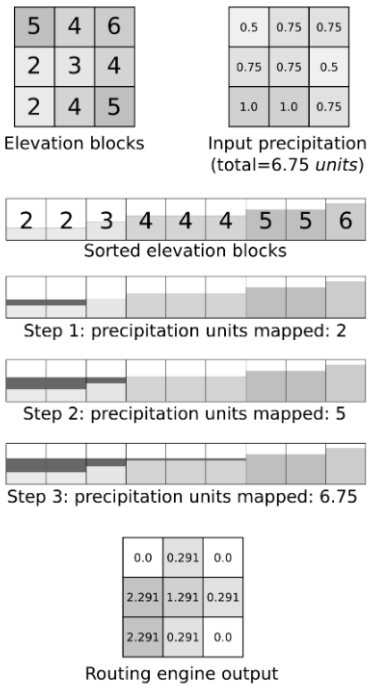


Figure 4: Example to explain flood routing algorithm (steady state)

the output layer. All these layers contain many different nodes to perform different functions, each node has its own weights. Also, these layers are connected to each other. Parallelisation can be achieved in many ways. The paper discussed the exemplar parallelism in which training data is divided into smaller chunks of data and distributed to different clusters. Errors are calculated by matching the original data with the output of ANN, then these errors in different clusters are combined together before sending the next chunk of data to different clusters, then the updated weights are sent to each cluster to carry on the further training. In this way the training is parallelized by combining the computational power of different clusters as shown in figure 5. Exemplar parallelism is the best method for parallelization as it tries to handle training time parallelism, weight-updation parallelism, and node parallelism. In the paper, the method of exemplar parallelism is used to prepare neural network topology. In this method, the initial weights of the nodes which are assigned randomly are sent to slave nodes by the master node, these slave nodes act as different clusters and have four different cores to perform the operation. In Java, the master node is used for message coordination among slave nodes as shown in figure 6. This master-slave node helps to distribute the task to achieve the goal of parallelism by making different clusters and distributing the task to them. These clusters have small chunks of data and different nodes from the layers to calculate the errors. During training, each cluster completes its job and calculates the error, which is then combined together and sent back to update the weights in each cluster, this task of re-distribution of weights to each cluster is done with the help of master node.Similarly, all nodes start doing their next iteration with these new weight values. Communication between the clusters is done with the help of message passing by the creation of shared objects. These message objects or shared objects in the network contain all the important information and the ANN model on which the training is being carried out. Using similar computers in a cluster with each having 4 cores and 8GB RAM the experiment to perform exemplar parallelism on ANNs using different clusters was successful.

### 3.4 Cluster of Raspberry pi for image processing

The paper (Marković et al., 2018) talks about practical examples of image processing such as analysis of X-ray images in the medical field. Analysis of satellite/drone images of a crop which can help in the agricultural domain. The author has focused on object detection and counting of wheat grains with the

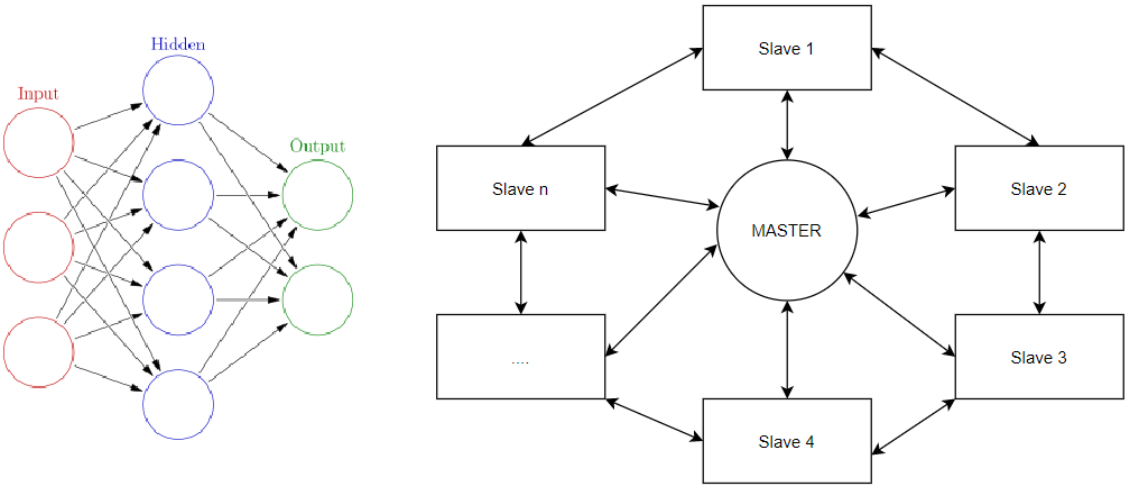


Figure 5: Neural network and cluster

help of object detection and has talked about the same in it's paper. In wheat production farmers wants maximum number of wheat grain and counting wheat grains manually is a hectic task. By parallelising this task the count of wheat grains were out in a some few seconds. Multiple raspberry pi were used to parallelize the task. Raspberry pi is a one board computer operating on the basis on linux. The architecture of the same is shown in figure 7. TCP/IP methods are usually used to form cluster with the help of MPI and PVM. One of the system here is assigned as head node and others are referred as computational node. They are connected through ethernet. Main advantage of this architecture is it's modularity i.e you can add more number of computational nodes without changing the architecture. Here python MPI library is used to parallelize the task. With the help of SSH commands nodes are setup and each of them is assigned a unique name (Host name) and are enabled so that they can work with the SSH protocol. In the machine file all the IP address of each nodes are stored. The machine file is stored on individual node and to communicate between nodes MPICH3 is used. This system can easily be used by other users with the help of internet. Using 3 images the comparison is shown. It took 2.4 seconds to process an image on a single system. Different programming languages can be used to compare the time and different platforms can also be used to see the time difference.

### 3.5 Beowulf Cluster for EEG data processing

Distributed and parallel computing exists in various applications with different types of parallelisms. This paper (Yao et al., 2009) mentions prior literature that is based on the similar ideology as the authors' which has been used to conclude the feasibility of the usage of such distributed parallel clusters for EEG data processing. The system description consists of a beowulf cluster formed with several pc nodes equipped with 2GB memory, intel DUO CPU at 2.8GHz interconnected by a Gigabit ethernet switch as shown in figure 8 . It also has 2NICs for internet and private network connections. Cluster is equipped with LINUX and MPI. This cluster runs the distributed and parallel compute engine for providing EEG produced data processing requirements like, (1) Computation task management or resource management carried out by the master node, (2) Task synchronisation to ensure global state consistency, (3) Runtime libraries. Neuronal population sync helps us find normal or abnormal brain functioning. This method is traditionally done in two steps by first finding the correlation between all pairs of signals and then a correlation matrix analysis. However the parallel sync measurement approach is both spatial and temporal. The implementation displays optimization opportunities. In this paper two major methods were performed (1) original method's parallel implementation (2) routine for randomization of EEG data for parallel surrogating which can be performed and coordinated on multiple nodes.

The sequential method took 9864 sec whereas the clusters in different modes took 5736sec and 2301

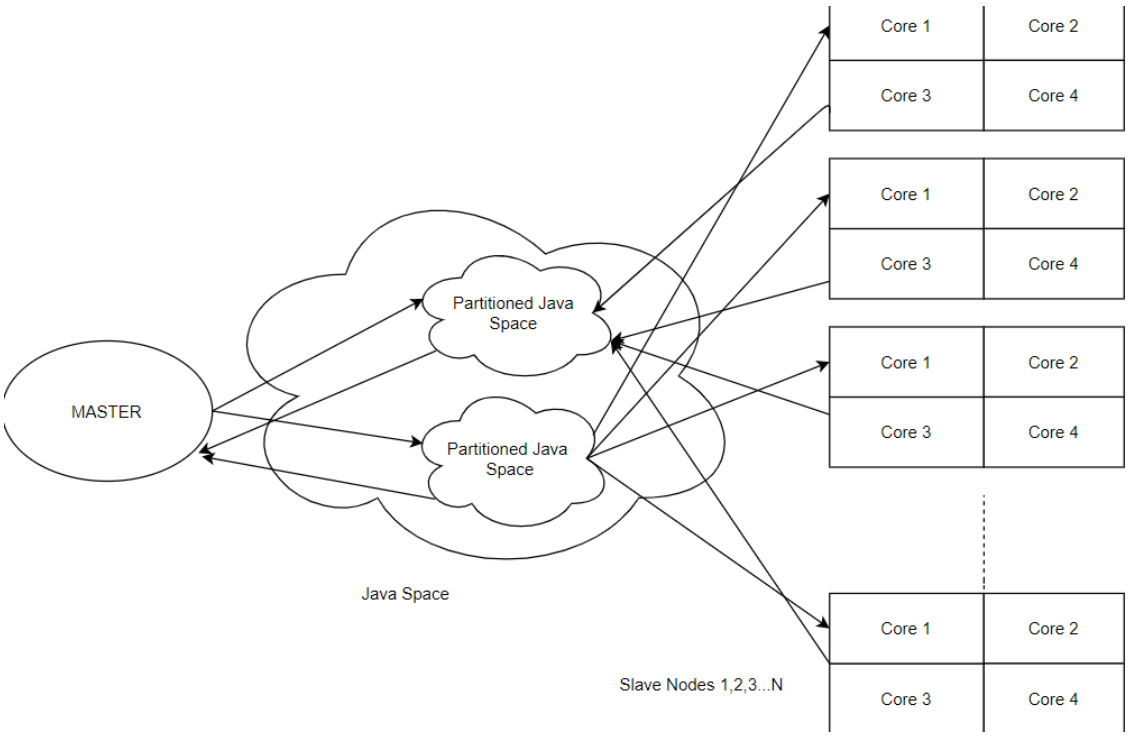


Figure 6: Processing units illustration

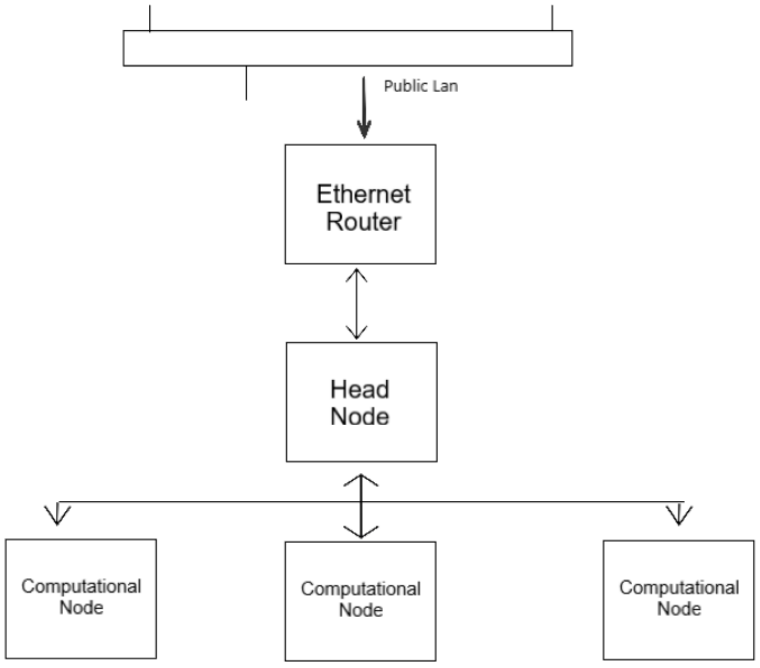


Figure 7: The architecture of cluster

sec for 2 node cluster and 5 node cluster respectively which was significantly better than the original.

### 3.6 Data Mining Algorithms on Raspberry Pi Cluster

They (Saffran et al., 2016) used the multi-threading in both Apriori and K-means to carry out the experiments on the Intel Xeon Phi. MIC System Management and Configuration (MICSMC) tool by Intel was used to monitor power consumption. They varied the number of threads proportionally to the number of

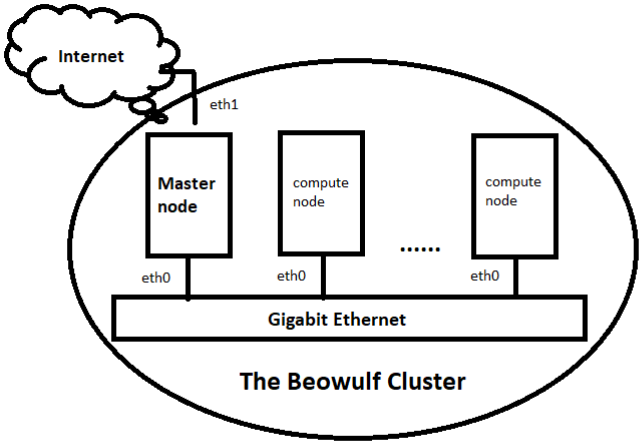


Figure 8: The Beowulf Cluster and its components

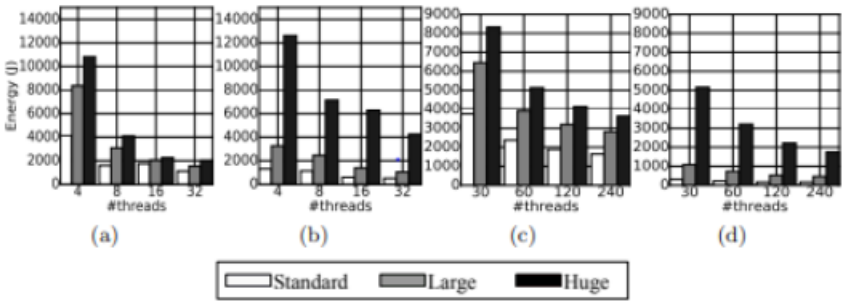


Figure 9: Execution time: (a)Raspberry Pi cluster - Apriori, (b) Raspberry Pi cluster - K-Means, (c) Intel Xeon Phi - Apriori, (d)Intel Xeon Phi - K-Means

cores available in the Intel Xeon Phi,( 30, 60, 120 and 240 threads)

For the Rpi cluster, they adopted a hybrid programming model: utilized the OpenMPI library to implement the distributed version of the applications and OpenMP inside each node to exploit four cores available in it. The total consumption in kilowatts-hour of the whole system was measured using a watt-meter connected between the power source and the cluster. Four threads per cluster were used and varied the number of nodes in 4, 8, 16 and 32.(fig 3) For the Apriori algorithm: 70 for the standard workload, 60 for large and 50 for huge ,minimum support values were used. For K-Means, increased the number of data points to be clustered and used the following number of data points: 214 for the standard workload, 215 for large and 216 for the huge one. Finally, 10 runs for each configuration was performed and computed the average execution time and power. The maximum standard deviation observed in Raspberry Pi cluster executions was 11.04 percent , while the Intel Xeon Phi presented at most 7.07 percent.

## 4 Implementation and Result

We implemented 2 survey paper and one application of our own. Applications implemented are - 1. Wheat count using image processing 2. Applying FFT on raw eeg signals. 3. Finding prime numbers in an array

### 4.1 Results

For the implementation, a cluster of 2 Raspberry Pi's was taken and its performance was evaluated on the above mentioned applications. The setup consisted of a Raspberry Pi 3b with a quad core 1.2GHz CPU and 1GB RAM and another Raspberry Pi 3b+ model with Cortex A53 @1.4GHz and 1GB RAM, both had Gigabit Ethernet ports which were used to connect to a spare router converted into a switch. All of these made up our cluster. This was inspired by the Beowulf clusters as all the material was commodity grade and easy COTS available. The software consisted of a custom python code which was used to find prime numbers, count the number of wheat grains from different images and the other used to apply FFT to EEG signal, both were written in python3 and used the Numpy, MPI4PY, Pandas libraries. Here the MPI4PY was the most important as it facilitated most of the intra-cluster communication and processes. MPI was the backbone of these codes. The processing completion times were calculated and analysed to find out that the 2 node cluster would perform about 50% better and even 70% in some cases depending on the heating of the boards and headless conditions. In table 4, 5, 6 we can see the comparison of time between sequential and parallel code of different implementations that are mentioned above.

Table 4: Wheat count using image processing

| | Number of images | Edge detection(s) (2rpi) | Watershed algorithm(s) (2rpi) |
|---|---|---|---|
| Sequential | 4 | 0.0367 | 0.38 |
| Parallel | 4 | 0.0151 | 0.28 |

Table 5: Conversion of raw EEG signals to FFT

| | Length of array with raw EEG value | FFT conversion(s) |
|---|---|---|
| Sequential | 70 | 0.0294 |
| Parallel | 70 | 0.0134 |

Table 6: Finding prime numbers in an array

| | Length of array | Prime number detection (s) |
|---|---|---|
| Sequential | 100000 | 992.25 |
| Parallel | 100000 | 515.15 |

## 5 Conclusion

Parallel processing has evolved significantly over the last 20 years. The evolution of parallel processing in many fields is on a fast track. Parallel processing could be implemented on many different applications in the field of computer science to improve a system's performance. The purpose of high performance computing clusters is to use parallel computing to apply more processor force to complete the task in a more efficient manner. Parallelization techniques produced good results and in some cases showed potential of very high speedups. Since we do have access to advanced parallel processing hardware, we should fully exploit the capabilities of these tools in the image processing domain as well. Hence we need to have a deeper knowledge about the workings of parallelism. In order to increase the performance while training the data in the field of Artificial Intelligence we can use parallel computing. As expected, the results showed that training time of artificial neural networks by using different clusters is much lesser than a single machine's training time. A large cluster of RPi can be formed to parallelize different tasks. Also, different programming languages can be used to compare the time and different platforms can also be used to check the time difference, hence judge the performance. In addition to it, several edge detection algorithms can also be used to validate the result. This paper proved its viability. It was

successful in exploiting the spatial and temporal parallelism of the sequential system where it performed exceptionally by reducing the execution time considerably and proved to provide efficiency when degree of parallelism is high only when parallelism is thoughtfully taken care of.

## References


Dušan Marković, Dejan Vujičić, Dragana Mitrović, and Siniša Ranić. 2018. Image processing on raspberry pi cluster. *International Journal of Electrical Engineering and Computing*, 2(2):83–90.

A Fakhri A Nasir, M Nordin A Rahman, and A Rasid Mamat. 2012. A study of image processing in agriculture application under high performance computing environment. *International Journal of Computer Science and Telecommunications*, 3(8):16–24.

João Saffran, Gabriel Garcia, Matheus A Souza, Pedro H Penna, Márcio Castro, Luís FW Góes, and Henrique C Freitas. 2016. A low-cost energy-efficient raspberry pi cluster for data mining algorithms. In *European Conference on Parallel Processing*, pages 788–799. Springer.

Md. Haidar Sharif and Osman Gürsoy. 2018. Parallel computing for artificial neural network training using java native socket programming. *Periodicals of Engineering and Natural Sciences*, 6, 02.

Swati Singhal, Lucas Villa Real, Thomas George, Sandhya Aneja, and Yogish Sabharwal. 2013. A hybrid parallelization approach for high resolution operational flood forecasting. In *20th Annual International Conference on High Performance Computing*, pages 405–414. IEEE.

Yufeng Yao, Jinyi Chang, and Kaijian Xia. 2009. A case of parallel eeg data processing upon a beowulf cluster. In *2009 15th International Conference on Parallel and Distributed Systems*, pages 799–803. IEEE.