Heuristic Experiments of Threading and Equal Load Partitioning For Hierarchical Heterogeneous Cluster

Noor Elaiza Abdul Khalid[1], Rathiah Hashim[2], Noorhayati Mohamed Noor[1], Muhammad Helmi Rosli[1], Mazani Manaf[1]
[1]Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA Shah Alam, Selangor, Malaysia
[2]Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, Parit Raja, Malaysia

E-mail: elaiza@tmsk.uitm.edu.my, radhiah@uthm.edu.my, noorhayati@tmsk.uitm.edu.my, helmirosli@gmail.com, mazani@tmsk.uitm.edu.my

Abstract. Presently, the issue of processing large data on a timely manner poses as a challenge to many ICT researchers. Most commodity computers are interconnected in a network forming a cluster computing resource simulating a super computer. This paper explores heuristically the performance of homogeneous, heterogeneous and multi-core clusters. This work consists of five experiments: Equal task partitioning according to the number of nodes in homogeneous cluster, number of nodes in heterogeneous cluster, number of nodes in heterogeneous cluster with multithreading, number of cores in heterogeneous cluster and number of cores in heterogeneous cluster with multithreading. The task is Sobel edge detection method tested with an array of images. The images are processed in three different sizes; 1K x 1K, 2K x 2K and 3K x 3K. The performance evaluations are based on processing speed. The results yield promising impact of equal partitioning and threading in parallel processing hierarchical heterogeneous cluster.

Keywords: Parallel Processing, Task Partitioning, Hierarchical Heterogeneous Cluster, Multi-Core, Heuristic Testing

1. Introduction

Moore's law predicted that the volume of data generated will exceed computational power capability. Advance technology in machinery has made it possible to generate massive volume of data also known as the Big Data phenomenon. With this came the challenge of mining and analysing data to extract meaningful and accurate business or scientific information [1]. In spite of the existence of very efficient serial algorithms, the problems of harvesting information still remains due to the magnitude of processing and handling such datasets [2]. Such datasets usually requires supercomputers to ensure reasonable completion times [3].

Recent years have seen the rise of highly parallelized approaches on multi-core-servers or computer clusters to deal with this problem [4]. Low cost commodity PCs provides high computing resources which are often underutilized [4][5]. A group of these loosely coupled computers connected via high-speed network forms clusters which are scalable in nature [6]. Clusters provide a viable cost-effective platform for the execution of intensive parallel multithreaded applications [5] [7] [8] and

simulate a super computer [3].In addition to multi-core CPUs, current desktop systems are equipped with programmable accelerators, such as Graphics Processing Units (GPUs), which raise both the system's processing power and the level of intrinsic heterogeneity [9]. This converts the traditional clusters into hierarchical systems, where each computing node may have several multi-core processors [10].

The challenge of using these resources to process vast amount of data and information is intriguing as developers need to design their applications to run efficiently on distributed, hierarchical, heterogeneous environments [10]. Large-scale distributed processing platforms nodes consisting of multi-core processors motivates the integration of traditional APIs to articulate concurrency (threads) and scalable parallelism (messaging) [11]. Research in the area is still in its infancy as true potentials of these systems have not been fully explored [9]. Current systems solve this problem by dividing large data into chunks which can be distributed over computers in a cluster. However, the array of computing power and speed in heterogeneous distributed platform coupled with intrinsic characteristics of data-intensive problems instigate huge load imbalances which affect the efficiency of resources utilization [10].

Distributed computing systems have been deployed for the execution of data and computational intensive workloads[12]. In homogeneous clusters of PCs, static load balancing is accomplished by allocation equal tasks to each processor[7]. However, finding an efficient task partitioning techniques is becoming increasingly important for heterogeneous distributed systems where the availability and variability of nodes may change drastically over time[12]. The assignment of tasks to available nodes is referred to as a resource allocation or a mapping[13]. Factors affecting effective mapping policies are the sets of available nodes in the system, specification of these nodes, and their interconnectivity[7]. Efficient task partitioning and allocation techniques is critical in exploiting the utmost potential of highly heterogeneous distributed systems [9] [14]. These distributed system typically contain multiple processor cores, allowing multiple network packets to be processed concurrently [15]. Optimizing each nodes capability involves partitioning tasks into a number of separate concurrent tasks that matches the number of cores on the target architecture. Data can be divided into arbitrary-sized load fractions to suit the computation capability of each node[16].

Parallelism can be classified according to task or data. In task parallelism, each processor is allocated different task which executes the same data. Whereas in data parallelism, each processor executes different data with the same task such as Single Program Multiple Data (SPMD) model. SPMD can be applied in shared and distributed memory environments with the benefits of global communication, scalability, synchronization and ease of use [17]. OpenMP is one of the popular programming modes to solve limited memory capacity for shared memory environments [18]. On the other hand, MPI platform [8] allows communication through message passing which is important in distributed memory environments. To date, no literature has been found that empirically explores basic distributed systems architecture impact which involves both hardware and software.

This paper discusses static heuristics tasks partitioning that map to resources in hierarchical heterogeneous system. Images are partitioned into chunks according to number of nodes and number of cores which are distributed over a cluster of computers. In addition to this, the use of threading in optimizing the usage of cores is also explored. The images are processed using Sobel edge detection algorithm which can be classified as task intensive and since the size of the image are significant it can also be classified as data Intensive.

2. Data intensive case study

The experiment is done based on image data involving an algorithm namely Sobel to detect edges of objects in the image. Sobel is a simple and popular edge detection algorithm that is used to verify the output consistency throughout the experiments of the heuristic parallel processing. The data consists of ten different types of images in three different sizes (1Kx1K, 2Kx2K and 3Kx3K). Images are selected as the experiment data as it fit the criteria of both data and computationally intensive. The lists of images are shown in Figure 1.

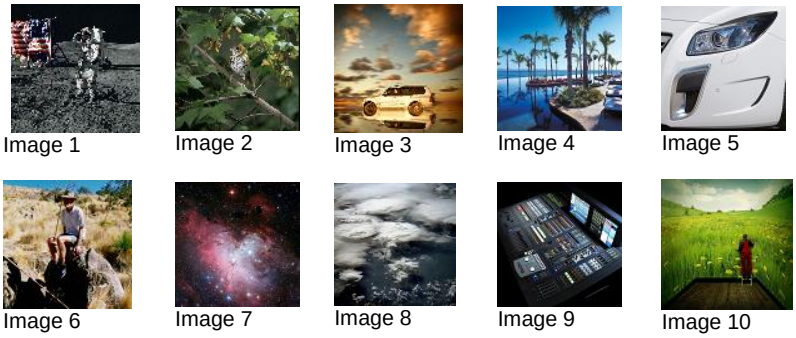

Figure 1. Test Image

3. Experimental Works

3.1 Parallel processing heuristic experiments

The experiments contain two major testing components; hardware and software. The hardware architecture relies on parallel architecture and specification of the node in the cluster. The software design involves scheduling and task partitioning. The experimental begins with determining the processors' specifications in the clusters then networked together. Next, task partitioning is programmed at the master node (Master-Worker [11]) to equally divide the image according to the hardware specifications. Then, the intermediate results at each node are communicated to the master node to produce a final output.

Two types of parallel architectures are homogeneous and heterogeneous. Homogeneous parallel architecture consists of identical processors; meanwhile heterogeneous parallel architecture is a mixture of no identical processors working in tandem as a system. In the experiment, the homogeneous cluster is built using four single core processors and in the heterogeneous cluster, a mix of two single core processors and two duo core processors.

The flow of the experiments consists of four phases; partitioning, allocating, image processing and image stitching. Initially, the image is partitioned equally according to the number of nodes. Then, the sub-images are allocated to each node before being processed by its processor. Finally, all sub-images are stitched together to produce a single output. Figure 2 depicts a conceptual flow of the parallel algorithm.

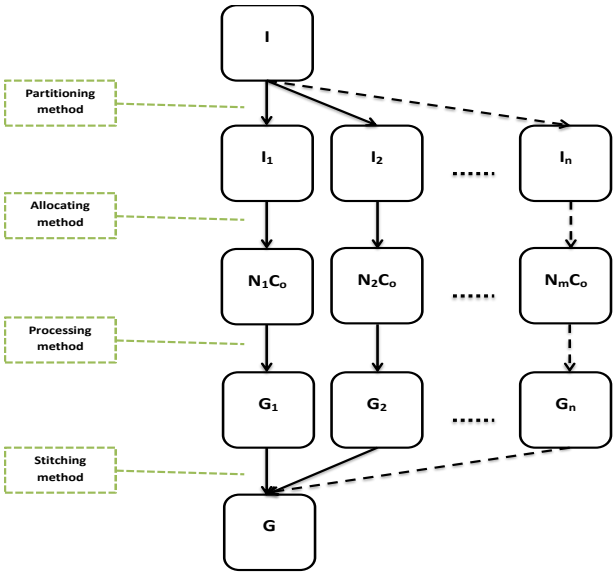

Figure 2. Conceptual Flow

Partitioning method is a method that involves how to separate the large data; 1kx1k, 2kx2k and 3kx3k image split into smaller images called sub-images (image A.1, A.2 image) in FIGURE 3-1. To split a large image cannot be split blindly. It should take 31 into account the Sobel case studies. The Sobel edge detection algorithm is done in 3x3 filters to apply the edge. So the division should be made with preferred images in one pixel per cut. Split the image according to the heuristic experience specification needs pixel overlapping due the nature of Sobel. Each partition is numbered by the location of the partition.

One image divided with total node on heterogeneous architecture. All this can be summarize in term mathematical equation below in EQUATION 3-1. Where n is a total number of node and L is a length of image. So the ($\boldsymbol{x}$) is an area every node can get.

Partitioning Equation

$$f(x) = \frac{L}{n}$$

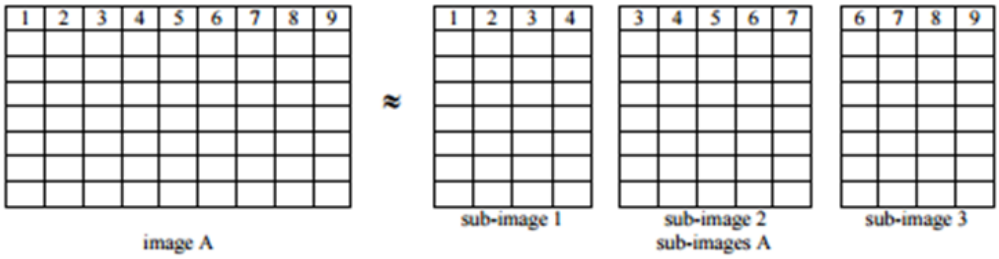

Figure 3. 3 Partitioning Images in Research

3.2. Heuristic experiment

In order to understand the hardware and software performances in parallel processing, series of experiments were outlined. The parallel experiments cover both the homogeneous and heterogeneous architectures. The heterogeneous architecture with equal partitioning according to nodes and cores are investigated together with/without threading. Table 1 lists all the experiment conducted in this research.

Table 1: Experiment listing

Experiment No	Parallel Architecture	Node Partitioning	Core Partitioning	Multithreading
1	Homogeneous	✓	×	×
2	Heterogeneous	✓	×	×
3	Heterogeneous	✓	×	✓
4	Heterogeneous	✓	✓	×
5	Heterogeneous	✓	✓	✓

First, the tasks are partitioned equally among identical nodes. Secondly, a heterogeneous cluster is tested by partitioning tasks equally among the nodes. Then, partitioning of tasks is performed according to the hardware specifications (number of cores in each node) of the heterogeneous cluster to explore the implication of cores to the parallel performance.

4. Results and discussion

The result for each node is based on speed of processing in seconds. The total time is taken from Original Image A to Full Result Image A (refer to Figure 2). In addition, the CPU is measured by percentage of usage.

4.1. Results for Homogeneous Architecture with Node Partitioning

The total processing time for homogeneous cluster on each node shows almost the same value. The ratio between image size and the processing time are approximately linear. Table 2 shows the details of the performance measurements.

Table 2. Speed of Homogeneous Architecture with Node Partitioning

No	1k x 1k					2k x 2k					3k x 3k				
	Node speed (s)				Total	Node speed (s)				Total	Node speed (s)				Total
	1	2	3	4	Time	1	2	3	4	Time	1	2	3	4	Time
1	6	7	7	7	9	25	25	25	24	28	55	55	55	55	60
2	6	6	6	7	9	25	25	25	25	29	54	55	54	54	60
3	7	7	7	6	8	24	24	24	25	28	54	55	54	54	60
4	6	6	7	6	9	25	25	25	24	28	55	54	54	54	60
5	6	6	6	7	9	25	25	25	25	28	54	55	54	54	59
6	7	7	7	6	9	25	25	24	24	27	54	55	54	54	60
7	6	6	7	6	9	24	25	24	24	29	55	54	55	54	60
8	6	6	7	7	9	25	25	24	25	28	54	55	54	55	60
9	7	6	7	6	9	24	24	26	24	28	54	54	54	54	59
10	6	6	6	6	9	25	24	17	33	28	55	55	54	54	60

It is also found that the type of image does not have any influence on the processing speed. Table 3 shows the usage of CPU on each node in percentage. It is found that all CPUs are fully utilized.

Table 3. CPU usage of Homogeneous Architecture with Node Partitioning

size	Node CPU usage (%)			
	1	*2*	*3*	*4*
1k x 1k	100	100	100	100
2k x 2k	100	100	100	100
3k x 3k	100	100	100	100

This experiment shows that equal partitioning of data according to the number of nodes works well in the homogenous architecture.

4.2. Results for heterogeneous architecture with node partitioning

The results for heterogeneous cluster with equal node partitioning show almost similar values when compared to homogeneous cluster as depicted in Table 4. Therefore, it can be concluded that the node (in heterogeneous cluster) performs at equal speed regardless of two or single core. This indicates that task allocation totally relies on user's partitioning software rather than hardware manager.

Table 4. Speed of Heterogeneous Architecture with Node Partitioning

No	1k x 1k					2k x 2k					3k x 3k				
	Node speed (s)				Total	Node speed (s)				Total	Node speed (s)				Total
	1	*2*	*3*	*4*	Time	*1*	*2*	*3*	*4*	Time	*1*	*2*	*3*	*4*	Time
1	6	6	7	6	9	25	24	24	24	24	55	54	54	54	65
2	6	7	6	7	10	26	23	25	25	25	55	54	54	54	62
3	6	6	6	7	9	24	24	24	24	24	54	54	54	54	60
4	6	6	7	6	8	25	24	24	24	24	55	54	54	53	59
5	7	6	6	6	10	25	24	25	24	24	55	53	54	54	61
6	6	6	6	6	9	24	24	24	24	24	55	54	54	53	61
7	6	6	7	6	9	24	24	25	24	24	54	53	54	54	60
8	6	6	6	7	9	24	24	24	24	24	55	54	54	53	60
9	6	6	7	7	9	24	24	24	24	24	54	53	54	54	60
10	6	6	7	6	10	24	24	24	25	25	56	54	54	54	60

In terms of CPU usage, nodes that have two cores are found to be partially (about half) utilized. Whereas the CPU in single core nodes are fully utilized as shown in Table 5. This implies that in the case of two core nodes, the task is only allocated to either one of them which suggest that hardware manager does not support automatic task partitioning.

Table 5. CPU usage of Heterogeneous Architecture with Node Partitioning

No	1k x 1k				2k x 2k				3k x 3k			
	Node CPU usage (%)				Node CPU usage (%)				Node CPU usage (%)			
	1	*2*	*3*	*4*	*1*	*2*	*3*	*4*	*1*	*2*	*3*	*4*
1	55.22	52.17	100	100	64.39	54.29	100	100	67.65	52.31	100	100
2	54.35	51.52	100	100	61.72	52.34	100	100	63.89	52.82	100	100
3	55.15	52.21	100	100	53.68	52.94	100	100	51.56	52.24	100	100
4	52.27	52.14	100	100	52.34	66.67	100	100	65.07	52.31	100	100
5	64.39	52.99	100	100	53.03	52.14	100	100	68.84	52.17	100	100
6	51.47	52.21	100	100	53.03	54.29	100	100	62.69	52.94	100	100
7	53.73	52.21	100	100	53.03	52.21	100	100	60.61	52.21	100	100
8	53.03	52.14	100	100	67.36	52.24	100	100	66.42	52.21	100	100
9	55.15	52.78	100	100	53.68	51.47	100	100	61.97	52.86	100	100
10	52.90	51.43	100	100	52.17	51.56	100	100	57.86	52.94	100	100

In conclusion, the overall results of this experiment show that the hardware manager alone cannot optimize the use of all available CPUs in heterogeneous hierarchical architecture.

4.3. Results for heterogeneous architecture with node partitioning and multithreading

Since this experiment is similar to experiment 2 except the use of multithreading, the results show the node speed significantly increased by almost doubled for small image size. However, as the size of images increase the difference in speed becomes more obvious. Table 6 illustrates the results.

Table 6. Speed of Heterogeneous Architecture with Node Partitioning with multithreading

No	1k x 1k					2k x 2k					3k x 3k				
	Node speed (s)				Total	Node speed (s)				Total	Node speed (s)				Total
	1	*2*	*3*	*4*	Time	*1*	*2*	*3*	*4*	Time	*1*	*2*	*3*	*4*	Time
1	3	3	4	4	11	9	15	15	22	24	21	21	33	33	42
2	3	3	4	4	11	10	15	15	18	25	22	21	32	32	38
3	2	2	4	4	6	9	15	15	17	24	21	21	32	32	38
4	2	3	3	3	6	9	14	14	18	24	22	21	32	32	37
5	3	3	4	4	7	9	14	14	18	24	21	21	32	33	37
6	3	3	4	5	7	9	15	15	18	24	21	21	32	32	37

7	2	3	4	4	6	10	15	14	18	24	22	21	32	33	37
8	2	2	4	4	6	9	15	15	18	24	22	21	32	33	38
9	3	3	3	4	6	9	15	15	18	24	21	21	32	32	37
10	3	2	4	4	6	10	15	15	18	25	22	21	33	32	38

In addition, multithreading helps increase the speed for two core nodes with regards to single core while the CPU of all nodes shows full utilization as depicted in Table 7.

Table 7: CPU usage of Heterogeneous Architecture with Node Partitioning with multithreading

size	Node CPU usage (%)			
	1	**2**	**3**	**4**
1k x 1k	100	100	100	100
2k x 2k	100	100	100	100
3k x 3k	100	100	100	100

This experiment shows that using multithreading increases the speed of processing for all the nodes. Multithreading also manage to utilize the duo core CPUs more efficiently as seen in node 1 and 2 in table 5.

4.4. Results for heterogeneous architecture with node and core partitioning

Table 8 shows the results of experiment 4. The results show that node 3 and 4 which are single core, perform almost twice better than node 1 and 2 (dual core). This corresponds to the allocated sub-image size.

Table 8. Speed of Heterogeneous Architecture with Node and Core Partitioning

No	1k x 1k					2k x 2k					3k x 3k				
	Node speed (s)				Total Time	Node speed (s)				Total Time	Node speed (s)				Total Time
	1	***2***	***3***	***4***		***1***	***2***	***3***	***4***		***1***	***2***	***3***	***4***	
1	8	8	4	5	12	33	32	16	16	39	74	71	37	37	81
2	8	8	4	5	12	33	32	16	17	37	73	71	36	36	80
3	8	8	4	4	12	33	32	17	16	37	73	72	36	36	79
4	8	8	4	5	12	33	33	16	17	38	73	72	36	37	82
5	8	8	5	4	12	32	32	17	17	36	73	71	36	37	80
6	8	8	4	5	12	32	32	17	16	38	74	72	36	36	81
7	9	8	5	4	14	32	32	16	17	38	73	73	36	36	79
8	9	8	4	5	13	32	31	17	16	37	74	72	37	36	80
9	9	8	5	4	12	32	32	16	17	38	73	72	36	36	80
10	8	8	5	4	13	34	33	17	16	40	73	73	36	36	80

In terms of CPU usage, nodes that have two cores are found to be partially (about half) utilized. Whereas the CPU in single core nodes are fully utilized as shown in Table 9. This also indicates that in the case of two core nodes, the task is only allocated to either one of them which suggest that hardware manager does not support automatic task partitioning.

Table 9. CPU usage of Heterogeneous Architecture with Node and Core Partitioning

No	1k x 1k				2k x 2k				3k x 3k			
	Node CPU usage (%)				Node CPU usage (%)				Node CPU usage (%)			
	1	***2***	***3***	***4***	***1***	***2***	***3***	***4***	***1***	***2***	***3***	***4***
1	52.21	52.17	100	100	52.94	74.24	100	100	65.63	51.52	100	100
2	51.52	52.78	100	100	52.70	52.34	100	100	63.08	52.82	100	100
3	52.14	51.47	100	100	71.01	51.52	100	100	70.42	52.17	100	100
4	51.52	53.03	100	100	53.03	87.91	100	100	64.93	52.86	100	100
5	51.52	50.77	100	100	53.03	51.54	100	100	64.93	53.79	100	100
6	51.47	51.47	100	100	65.15	53.03	100	100	59.85	53.79	100	100

7	67.81	54.17	100	100	52.99	52.21	100	100	66.91	53.13	100	100
8	64.29	52.86	100	100	54.48	52.34	100	100	58.59	53.03	100	100
9	69.40	53.91	100	100	52.94	52.99	100	100	64.71	55.22	100	100
10	60.14	52.14	100	100	55.30	52.27	100	100	53.08	73.38	100	100

This indicates that when a duo core node is given twice the size of single core, the sub-images are not automatically partition according to the available cores.

4.5. Results for heterogeneous architecture with node and core partitioning inclusive multithreading
Table 10 shows the results of experiment 5. The results show that the single core performs only slightly better than dual core node. This implies that multithreading the processing speed for both single and duo core nodes.

Table 10. Speed of Heterogeneous Architecture with Node and Core Partitioning with multithreading

No	1k x 1k					2k x 2k					3k x 3k				
	Node speed (s)				Total	Node speed (s)				Total	Node speed (s)				Total
	1	*2*	*3*	*4*	Time	*1*	*2*	*3*	*4*	Time	*1*	*2*	*3*	*4*	Time
1	4	3	3	2	7	13	12	10	10	17	28	28	22	22	34
2	3	3	3	2	7	13	12	10	10	17	28	28	22	22	35
3	3	4	3	3	7	13	12	9	10	18	28	28	22	22	35
4	3	3	2	2	8	12	13	10	10	17	28	27	22	22	34
5	4	3	3	3	8	13	13	11	10	18	29	27	22	22	34
6	3	3	2	3	7	13	13	11	10	18	28	28	22	22	34
7	3	3	3	3	8	13	12	11	10	19	28	28	22	23	34
8	3	3	3	2	7	12	12	10	11	17	28	27	22	22	35
9	4	3	3	2	8	12	12	10	11	17	29	28	22	22	34
10	3	3	2	3	8	13	13	10	10	18	28	28	22	22	35

Multithreading also helps in CPU utilization as shown in Table 11.

Table 11. CPU usage Heterogeneous Architecture with Node and Core Partitioning with multithreading

size	Node CPU usage (%)			
	1	*2*	*3*	*4*
1k x 1k	100	100	100	100
2k x 2k	100	100	100	100
3k x 3k	100	100	100	100

This experiment shows that multithreading is able to optimize the processing speed by utilizing the all the cores and CPU in the cluster.

5. Discussion and Conclusion

Experiment 1 shows that in homogenous architecture, a simple equal partition will equally speedup the processing with equal partition allocation. The result of Experiment 2 which is almost similar to experiment 1 indicates that a heterogeneous architecture will perform the same as homogenous architecture when it is allocated with equal partition. However, adding multithreading in Experiment 3 shows an improvement in processing time compared to Experiment 2. Hence, utilizing multithreading helps to optimize the heterogeneous architecture. Moreover, CPU usage indicates the cores are fully utilized which is reflected by the smaller processing time of the duo cores node.

In Experiment 4 where partition allocation corresponds to number of core, the speed corresponds to the size of partition. This indicates the additional core does not have any impact on the

number of partition. This implies that the hardware manager cannot automatically optimize the load increase. Experiment 5 proves that the multithreading helps the hardware manager to optimize the core utilization to its potential.

In conclusion, partitioning according to number of node and core is viable. The full potential of the cores can be seen through multithreading.

6. Acknowledgments

Authors would like to thank Universiti Tun Hussein Onn Malaysia for the support in publishing this article and also to Gates IT Solution for the incentive to publish.

7. References

[1] Qiu, J., & Zhang, B. Mammoth Data in the Cloud: Clustering Social Images.2012
[2] Thomas Seidl, Brigitte Boden, Sergej Fries, CC-MR – Finding Connected Components in Huge Graphs with MapReduce, Machine Learning and Knowledge Discovery in Databases, Lecture Notes in Computer ScienceVolume 7523, pp 458-473, 2012.
[3] Ali, Md Firoj, and Rafiqul Zaman Khan. "The Study On Load Balancing Strategies In Distributed Computing System." International Journal of Computer Science & Engineering Survey (IJCSES) Vol.3, No.2, 2012.
[4] W.Kang, H.H.Huang, A.Grimshaw, "Achieving high job executing using underutilized resources in a computational economy", Future Generation Computer System, Special Section: Recent Developments in High Performance Computing and Security, Elsevier, vol. 29. Issue. 3, pp:763-775, 2013.
[5] T. Sterling, T. Cwik, D. Becker, J. Salmon, M. Warren, B. Nitzberg, An assessment of beowulf-class computing for NASA requirements: initial findings from the first NASA workshop on beowulf-class clustered computing, in: Proceedings of the IEEE Aerospace Conference, 1998. http://loki-www.lanl.gov/papers/ieee_aero98/p312.ps.
[6] Zhiquan Sui, Shrideep Pallickara,2011 ,A Survey of Load Balancing Techniques for Data Intensive Computing, Handbook of Data Intensive Computing, pg 157-168, 10.1007/978-1-4614-1415-5_6
[7] Christopher A. Bohn, Gary B. Lamont, Load balancing for heterogeneous clusters of PCs, Future Generation Computer Systems, Volume 18, Issue 3, January 2002, Pages 389-400, ISSN 0167-739X, 10.1016/S0167-739X(01)00058-9. (http://www.sciencedirect.com/science/article/pii/S0167739X01000589)
[8] J.L. Hennessy, D.A. Patterson, Computer Architecture: A Quantitative Approach, 2nd Edition, Morgan Kaufmann, San Francisco, 1996, p. 17.
[9] Ilic, A.; Sousa, L.; , "On Realistic Divisible Load Scheduling in Highly Heterogeneous Distributed Systems," Parallel, Distributed and Network-Based Processing (PDP), 2012 20th Euromicro International Conference on , vol., no., pp.426-433, 15-17 Feb. 2012 doi: 10.1109/PDP.2012.56
[10] George Teodoro, Timothy D. R. Hartley, Ümit V. Çatalyürek, Renato Ferreira: Optimizing dataflow applications on heterogeneous environments. Cluster Computing 15(2): 125-144 (2012)
[11] Berka, T., Kollias, G., Hagenauer, H., Vajtersie, M., & Grama, (2012). Concurrent Programming Constructs For Parallel MPI Applications.
[12] Vladimir Shestak, Edwin K.P. Chong, Anthony A. Maciejewski, Howard Jay Siegel, Probabilistic resource allocation in heterogeneous distributed systems with random failures, Journal of Parallel and Distributed Computing, Volume 72, Issue 10, October 2012, Pages 1186-1194, ISSN 0743-7315, 10.1016/j.jpdc.2012.03.003. (http://www.sciencedirect.com/science/article/pii/S0743731512000688)
[13] Lizhe Wang, Dan Chen, Ze Deng, Fang Huang, Large scale distributed visualization on computational Grids: A review, Computers & Electrical Engineering, Volume 37, Issue 4,

July 2011, Pages 403-416, ISSN 0045-7906, 10.1016/j.compeleceng.2011.05.010. (http://www.sciencedirect.com/science/article/pii/S0045790611000796)

[14] Qin-Ma Kang, Hong He, Hui-Min Song, Rong Deng, Task allocation for maximizing reliability of distributed computing systems using honeybee mating optimization, Journal of Systems and Software, Volume 83, Issue 11, November 2010, Pages 2165-2174, ISSN 0164-1212, 10.1016/j.jss.2010.06.024. (http://www.sciencedirect.com/science/article/pii/S0164121210001718)

[15] R. Ennals, R. Sharp, and A. Mycroft. Task Partitioning for Multi-Core Network Processors. In Proceedings of the IEEE International Conference on Computer Communications (ICCC), Mauritius, April 2005.

[16] Rosas Mendoza, C. (2012). Performance Improvement Methodology based on Divisible Load Theory for Data Intensive Applications. Thesis , Universidad Autonomy de Barcelona. Department of Computer Architecture and Operating Systems,http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6332177&isnumber=6331993

[17] Kamil, A., & Yelick, K. (2012). Hierarchical Additions to the SPMD Programming Model.

[18] Kwon, O., Jubair, F., Min, S.-J., Bae, H., Eigenmann, R., & Midkiff, S. (2011). Automatic Scaling of OpenMP Beyond Shared Memory.