GRASS GIS ON HIGH PERFORMANCE COMPUTING WITH MPI, OPENMP AND NINF-G PROGRAMMING FRAMEWORK

S. Akhter [a,b,*], K. Aida [a,b] and Y. Chemin [c]

[a] National Institute of Informatics, Hitotsubashi, Tokyo Japan –(shamim, aida)@nii.ac.jp
[b] Tokyo Institute of Informatics, Nagatsuta-ku, Yokohama, Japan
[c] International Centre of Water for Food Security, NSW, Australia- yann.chemin@gmail.com

Commission VIII, WG VIII/6

KEY WORDS: GRASS GIS, High Performance Computing, MPI, OpenMP, Ninf-G

ABSTRACT:

GRASS GIS (Geographical Resources Analysis Support System) is a free, open source software and has been used for Remote Sensing (RS) and Geographic Information System (GIS) data analysis and visualization. Inside GRASS, different modules have been developed for processing satellite images. Currently, GRASS uses databases to handle large datasets and the performance and capabilities of GRASS for large datasets can be greatly improved by integrating GRASS modules with parallel and distributed computing. Multi computer based distributed systems (clusters and Grids) have a large processing capacity for a lower cost, naturally, choice turns towards developing High Performance Computing (HPC) applications. However, it is not an easy job to port GRASS modules directly to HPC environment. The developers of satellite image processing applications need to solve the problem of both data and task distribution, or how to distribute data and tasks among single or multiple clusters environment. The workload in HPC, the bandwidth, the processors speed, parameters of evaluation methods and data size are additional concerning factors. GRASS modules, i.e. i) "i.vi" is developed by Kamble and Chemin (2006) to process 13 vegetation indices, ii) "i.lmf" is developed by Akhter et al. (2008) to remove the atmospheric effects from RS images and iii) "r.gaswap" is developed by Akhter et al. (2006) to find out the crop parameters those are not directly visible from RS images, will be discussed as three case studies to developed GRASS module framework on HPC. Developing the methodology, which enables to run GRASS GIS environment for RS images processing on HPC systems, will be the main concerning issue of this paper. Additionally, different implementations for distributed GRASS models will be discussed on three different programming platforms (MPI, Ninf-G and OpenMP) and their performance will also be presented in this paper.

1. INTRODUCTION

Satellite imagery provides a large amount of useful information and plays a vital role for research developments in Astronomy, Remote Sensing, GIS, Agriculture, Disaster Management and many other fields of study. Over the past decade Geospatial Information Systems (GIS) have evolved from a highly specialized niche to a technology that affects nearly every aspect of our lives, from finding driving directions to managing natural disasters. GIS is often described as integration of data, hardware, and software designed for management, processing, analysis and visualization of georeferenced data. GRASS GIS (Geographical Resources Analysis Support System) is a free, open source software/tool and has been used for RS and GIS data analysis and visualization (Neteler and Mitasova, 2003). GRASS GIS is capable of handling raster, topological vector, image processing, and graphic data. It includes more than 350 modules for management, processing, analysis and visualization of georeferenced data. Currently, GRASS uses databases to handle large datasets and the performance and capabilities of GRASS for large datasets can be greatly improved by integrating GRASS modules with parallel and distributed computing. Additionally, processing those satellite images requires a large amount of computation time due to its complex and large processing criteria. This seems a barrier for real time decision making. To switch the job faster, high performance computing can be a suitable solution. Multi computer based distributed systems (clusters and Grids) have a large processing capacity for a lower cost, naturally, choice turns towards developing High Performance Computing (HPC) applications. However, it is not an easy job to port any application in HPC environment. The application performance is significantly affected by the data distribution and task distribution methods on the HPC. Data and task distribution are the procedures to apply parallel techniques on data and task domain to gain efficiency. Thus the developers of GRASS modules or satellite image processing applications need to solve the problem of both data and task distribution, or how to distribute data and tasks among single or multiple clusters environment. The workload in HPC, the bandwidth, the processors speed, parameters of evaluation methods and data size are additional concerning factors.

Developing GRASS module framework on HPC, three case studies, i.e. i) "i.vi" is developed by Kamble and Chemin (2006) to process 13 vegetation indices, ii) "i.lmf" is developed by Akhter et al. (2008b) to remove the atmospheric effects from RS images and iii) "r.gaswap" is developed by Akhter et al. (2006) to find out the crop parameters those are not directly visible from RS images, are discussed. The foremost case study (i.vi) discusses the interoperability framework designing issues between the GRASS tool and HPC. The second case study discuss about the satellite image data distribution issues and the

* Corresponding author. Shamim Akhter, National Institute of Informatics, Japan.

final case study discusses the processing level data and task distribution on both cluster and grid platforms. Thus, the three case studies enable to run GRASS GIS environment for RS images processing on HPC systems and discuss the performance improvement issues also.

2. METHODOLOGY DEVELOPMENT AND EXPERIMENTS

2.1 Case Study 1: GRASS and HPC Interoperability Framework

Developing the methodology, which enables to run GRASS GIS environment for satellite image processing on distributed computing systems, is the main concerning issue here. Additionally, three different implementation methodologies for distributed i.vi are discussed for three different programming platforms MPI (MPI, 2007), Nin-G (Ninf-G, 2007) (Tanaka et al, 2003) and OpenMP (Akhter, S. and Roberts, J., 2006). GRASS module i.vi is developed by (Kamble et al., 2006), and is used as a test example for this study. GRASS module i.vi, is used to process 13 different Vegetation Indices (VI) for the satellite images (Akhter et al., 2008b).

Grass module i.vi works with raster images (rows x columns). Different band raster images are required for different indices. The generic indices (NDVI, RVI etc) use red and nir（near infrared）band images. However, arvi uses red, nir and blue band images, GVI uses red, nir, blue, green, chan5 and chan7 of landsat images and GARI uses red, nir, blue, green band images. GRASS functions are used to extract row wise data from the specific band images and store them in buffers. Then, each column value is extracted sequentially from the buffers and sends them for generating the specific VI values. Thus, after completing the VI from row buffers, the row wise VI values are put back into output image and this process will continue for each row. Figure 1 presents the structure of serial running i.vi module (for simplicity, only two band images are presented).

Master-worker approach has been taken to distribute i.vi module in MPI and Ninf-G platforms. The master process runs in the GRASS environment, and decomposes the target images in rows and dispatches the computation of rows to multiple worker processes. Worker processes are free from GRASS, they just run the computation and send back the row wise result to the master process. The module i.vi is implemented using MPI on a PC cluster system (i.vi.mpi) and Ninf-G on the same PC cluster system (i.vi.grid) to hold the similarity between the experimental environments. However, i.vi.grid module has been structured as it is capable to run in distributed GRID system. In Figure 2, the implementing structure of distributed i.vi is presented (for simplicity only two band images have been presented). Here, S1, S2,.....,Sn are different worker processes. However, OpenMP, a shared memory and multi-threading approach, whereby the master "thread" executes a series of instructions consecutively and calls a specified number of slave "threads" to divide the tasks among them. The threads then run concurrently to execute a block of tasks in parallel. In our GRASS OpenMP implementation (i.vi.omp) multiple threads execute the computation of a row (all columns) concurrently. Each cpu core is responsible to process each thread. The i.vi.omp runs in similar style of serial module under GRASS environment.

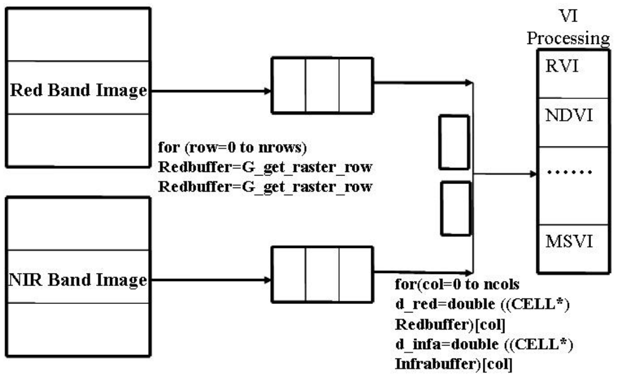
VI Processing
RVI
NDVI
……
MSVI
Red Band Image
NIR Band Image
for (row=0 to nrows)
Redbuffer=G_get_raster_row
Redbuffer=G_get_raster_row
for(col=0 to ncols
d_red=double ((CELL*) Redbuffer)[col]
d_infa=double ((CELL*) Infrabuffer)[col]

Figure 1: Serially Running i.vi Module Structure

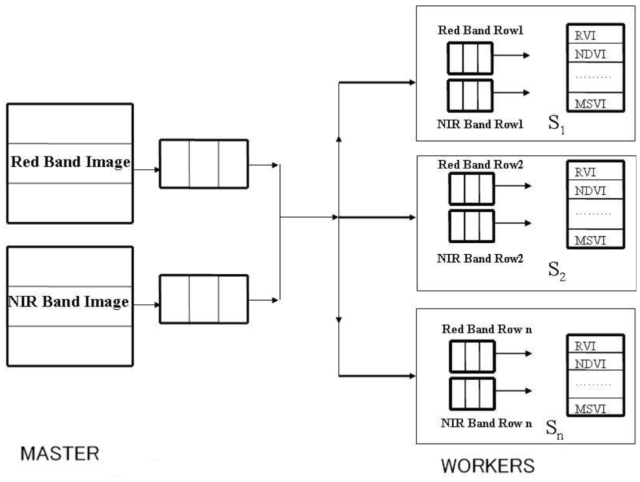

Figure 2: Distributed i.vi Module Structure

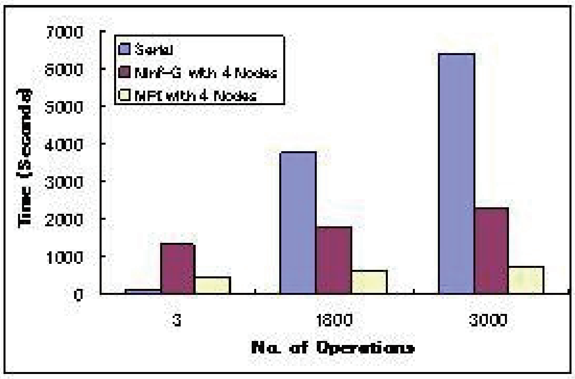

Figure 3: i.vi Module Performance Evaluation: Serial, MPI and Ninf-G Implementations

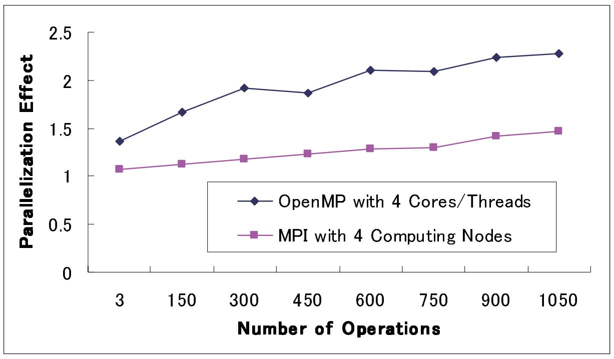

Figure 4: Parallel i.vi Performance with OpenMP and MPI

2.1.1 Experiments: The details MPI and Ninf-G implementations and performance comparisons were presented in (Akhter et al., 2008b) and the results (Figure 3) concluded that the i.vi MPI implementation performs better than Serial and Ninf-G implementations in a single cluster with 4 computing nodes. However, OpenMP performance was not been integrated. Running GRASS module with OpenMP requires to enable the –fopenmp flag and gomp libraries support. GRASS version 7 and gcc compiler > 4.0 are the additional compulsory requirements (GRASSWiki, 2010). Figure 4 presents the performance of i.vi with OpenMP (i.vi.omp) and MPI (i.vi.mpi) implementations. OpenMP version is implemented in a single node with 4 cpu cores platform where as the MPI version using 2 machines with 2 cpu cores each. According to Figure 4, the OpenMP implementation is performing better than the MPI implementation. Data distribution workload in MPI implementation is the main performance bottleneck. OpenMP implementation shares the same memory so data distribution workload is absent here.

2.2 Case Study 2: Satellite Image Data Distribution Issues

The Local Maximum Fitting (LMF) (Sawada et al., 2001) uses temporally splined procedure by combining the time series filtering and functional fitting for removing the clouds, hazes and other atmospheric effects from time series data of each pixel and ensure the data consistency (Sawada et al., 2002). OpenMP based LMF was initially implemented in (Sawada et al., 2005). A cluster based parallel LMF procedure was approached and implemented in (Akhter et al., 2007).

Data distribution methodology became necessary to address the Asia regional yearly temporal MODIS images (46 bands, 3932 rows and 11652 columns). A script is developed to read each row from all temporal images and stack them together to formulate a row-image. Each row-image is then passed into the parallel LMF model for processing. Although, working with the Row Distribution approach on the Asia regional MODIS images a LMF software limitation, increasing the column numbers more than 7000 creates a software segmentation fault, was traced. This happens because of the data storing constraint inside the LMF programming environment. As a result, both row-wise and column-wise data distribution mechanism is required and implemented.

In this methodology, all the temporal images in column direction are virtually (programmatically) partitioned into a desirable block. The block window size (BWS) needs to be chosen according to the image data type. We used a threshold value (7000) for window size selection so that the column data will be equally distributed. Thus, from each column portion, each row of all temporal images will be merged together to become a row image and then processed by LMF program. The image formulation scenarios of the Row and Row Column Distribution are presented in Figure 5. In the Row Column Distribution, two images with BWS=2 (Image1, Image2) have been formulated from each row of all temporal images.

2.2.1 Experiments: In Figure 6, both data distribution scenarios are implemented with Ninf-G programming platforms. It seems Row Column Distribution takes much time than the row distribution. This is a usual scenario in distributed computing aspects. More partitioning in the image, creates more data distributions (sending and receiving), much more timing.

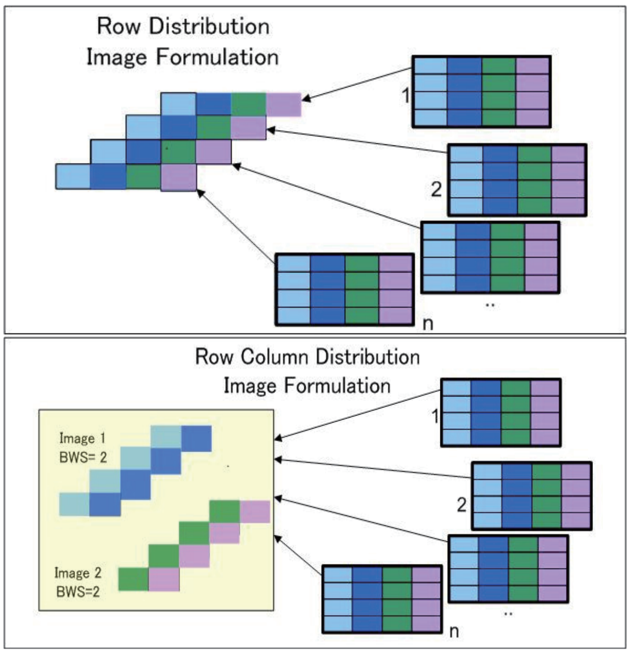

Figure 5: Image Formulation with Data Distribution Strategies

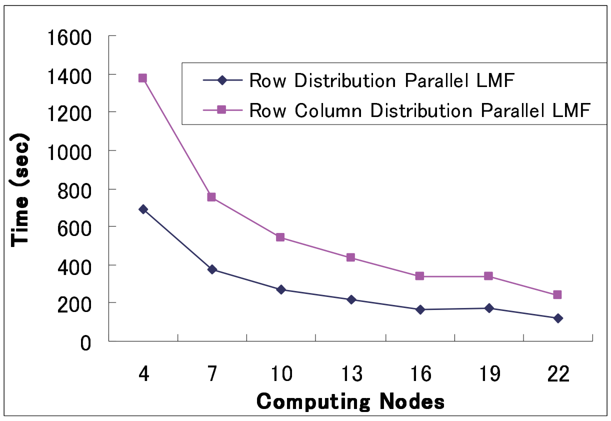

Figure 6: Run time comparison of Row Distribution and Row Column Distribution is a Single Cluster Implementation

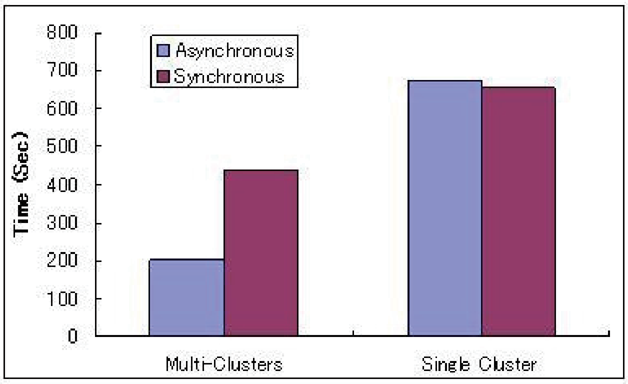

Figure 7: Time Comparison of Asynchronous and Synchronous Call

However, increasing computing nodes trends to reduce the runtime to the same level influenced to implement in Grid or multi-cluster based system with much more computing resources. According to the grid implementation of Row Column Distribution achieves 27.1% speedup with three clusters implementation from single cluster implementation. Additionally, Row Column Distribution on three clusters implementation gain 12% speedup over Row Distribution in single cluster implementation. Thus, grid based implementation can reduce the additional overhead by the Row Column Distribution.

Additionally, Ninf-G itself contains two different implementations (API Reference, 2008), Synchronous and Asynchronous, for data distribution. In Synchronous calling, client waits until the completion of the computation on the server side. Asynchronous call returns immediately after all the arguments have been sent. Figure 7 presents a comparison of Ninf-G calling systems for LMF processing. Both calling systems show nearly same performance in single cluster environment. However, their performance is different in multi-clusters environment. Asynchronous call seems faster than Synchronous call. This is happening because Asynchronous calling strategy provides a load balancing job distribution as First Come First Serve (FCFS). Thus, the fastest cluster will process more jobs than slower clusters. On the other side, Synchronous calling waits till one distribution of the submitted jobs to the clusters will not be finished. It means that due to the slowest performing cluster the overall performance will be delayed.

2.3 Case Study 3: Processing Level Data and Task Distribution Issues

SWAP-GA (Akhter et al., 2008a) is a combined model of the SWAP (Soil Water Atmosphere and Plant) crop model and the Remote Sensing (RS) data assimilation technique, which is optimized by Genetic Algorithm (GA). SWAP-GA is used to find out the crop parameters those are not directly visible from RS images. "r.gaswap", a GRASS module developed by Akhter et al. (2006), enables SWAP-GA to run inside GRASS environment.

The full SWAP-GA executable module is made with RSImageAccess, GASWAP, and Evaluation sub-modules. The RSImageAccess module is the main module, where the program starts. This module extracts the pixel value and the date for each image from the GRASS environment, and it calls this GASWAP module. The GASWAP module is able to run GA and completes the assimilation process. The Evaluation module runs the SWAP executable for each population and sends the simulated results to the GASWAP module. An example with 15 pixels image, 60 populations and 10 generations gives a clear view of calling procedures inside the SWAP-GA model. Particularly, in this case, the RSImageAccess module will call the GASWAP module 15 times (one for each pixel). For each pixel, the GASWAP module first internally executes the SWAP executable 60 times to initialize the simulated pixel value for every population and then calls the Evaluation module 10 times (one for each generation). For each generation, the Evaluation module executes the SWAP executable 60 times and produces simulated pixel values for 60 populations. A high demand of parallel computing is called for inside the whole SWAP-GA module. Three different strategies are applied to work SWAP-GA in a parallel manner.

The strategies are presented by the following approaches: i) Data Distribution Approach, ii) Task Distribution Approach, iii) Combined Data and Task Distribution Approach.

Implementation Strategy	Computing Nodes		
	1	2	15
Serial SWAP-GA	27,026 (sec)	–	–
Data Distribution	17,382 (sec)	9,276 (sec)	1,218 (sec)
Task Distribution	19,745 (sec)	10,941 (sec)	4,031 (sec)
Combined Distribution	–	18,216 (sec)	3,152 (sec)

Table 1: Execution Time in Single Site

2.3.1 The Data Distribution Approach: The Data Distribution approach is implemented on the Grid using Ninf-G. The Master-worker paradigm is used for parallelization in the distributed SWAP-GA model. Inside the master node, the RSImageAccess module works, whereas in the worker nodes assimilation procedures (GASWAP and Evaluation modules) run. The RSImageAccess module, running as Ninf-G client, dispatches a set of pixels to the remote PC cluster. In the remote PC cluster, the GASWAP module and the Evaluation module, running as a Ninf-G server, perform a computation for each pixel, where the computations inside the PC cluster are distributed among computing nodes through the local batch scheduler in FCFS manner.

2.3.2 The Task Distribution Approach: An evaluation procedure is called for each population to execute the SWAP executable. Thus, the Evaluation module (to evaluate population) can be distributed through Ninf-G. Here, the master node runs both the RSImageAccess and the GASWAP module, and worker nodes only run the Evaluation module. The GASWAP module, running as a Ninf-G client, dispatches a set of populations to the remote PC cluster. In the remote PC cluster, the Evaluation module, running as a Ninf-G server, performs an evaluation for each population. The computation inside the PC clusters is distributed among computing nodes through the local batch scheduler.

2.3.3 The Combined Data and Task Distribution Approach: So far, the above two approaches are running with Ninf-G, whereas this third approach is implemented with combined Ninf-G and MPI and called the Combined Distribution model. The idea of the Combined Distribution approach is to distribute the computation of pixels and populations in a hierarchical way. Here, the master node runs the RSImageAccess module and distributes a pixel to the gateway node, or the master node of the remote PC clusters with Ninf-G to invoke the GASWAP module. The approach is similar to the Data Distribution. After getting the pixel value, the master node (GASWAP) dispatches the populations to worker nodes (inside cluster) using MPI to run the Evaluation modules and this is similar to the Task Distribution Approach.

2.3.4 Experiments: Table 1 presents the running time comparison of the SWAP-GA serial model with the parallel SWAP-GA models with Computing Nodes 1, 2 and 15 on a single cluster. With 15 nodes, the performance of parallel models is improved. According to the distribution approaches, the Data Distribution approach performs better than others. In the Data Distribution approach, the dispatched workload (one whole pixel evaluation) is bigger and the communication overhead is hidden through the computing workload. Whereas, in the Task Distribution approach, Ninf-G calls happen frequently (once to evaluate the assigned populations set for each generation) and the workload to the computing nodes is not sufficient to gain the efficient parallelism. Additionally, Ninf-G takes some time for each RPC to establish and to close the session with computing nodes. On the other hand, to reduce the Ninf-G session establishment cost, the Combined Data and Task Distribution approach is presented where (inside the cluster) MPI reduces the session establishment cost (that was taken by Ninf-G) and the performance is improved. However, the performance of the combined approach is not superior to the Data Distribution approach. The same numbers of Ninf-G calls are conducted in both the Data Distribution and the Combined Distribution approaches. However, the Combined Distribution approach takes sometime for MPI communication.

Table 2 presents the additional experimental results on the real Grid testbed (multiple sites) with the Combined Data and Task Distribution Approach and the Data Distribution Approach with 15 pixels, 10 generations and 60 populations. The best performance for the Data Distribution approach was achieved in the single site experiment (1,378 sec). However, the Grid with more CPU power makes the Combined Distribution approach performance (922 sec) better than the best performance of the Data Distribution approach. Table 2 highlight the major drawback of the Data Distribution approach, when the pixel amount is diminutive compared to the computing nodes number. For this particular workload (with 15 pixels, 10 generations and 60 populations) more than 15 nodes will not create any advanced effects on the Data Distribution approach whereas the door is open for the Combined Distribution approach to use more than 15 computing nodes (at most 60 nodes in each cluster). However, the Combined Distribution approach performance greatly depends on the number of clusters as well as the number of computing nodes in each cluster. So, when the cluster number is equal to the pixel number and the computing node number inside each cluster is equal to the population number, it may provide the best performance for the Combined Distribution approach.

3. CONCLUSION

The three case studies concern on the three different implementation phases of GRASS GIS on HPC. The foremost case study concerns on the interoperability framework for GRASS and HPC and that has been successfully implemented with three different programming frameworks. OpenMP performance seems better than MPI and Ninf-G implementations. However, the performance on parallel implementation differs on the data distribution and task distribution issues. The second case study focuses on the image level data distribution to multiple computing nodes. Partitioning in the image data creates more process to work in parallel and that reflects more communication overhead. Additional computing power hides the communication overhead by increasing the processing efficiency. The third case study concerns on the processing level distribution issues. The processing codes or software itself can work in parallel by distributing independent coding modules to process with different computing nodes. Additionally, single or combined programming frameworks can improve the whole performance however, that needs to be chosen according to the coding structures, processing time, available computing resources, communication overhead etc.

Execution Strategy	# of Computing Nodes			Total Time (sec)
	Cluster One	Cluster Two	Cluster Three	
Combined Distribution	10	5	-	3,189
	5	10	-	3,329
	21	31	-	1,193
	16	16	8	1,004
	16	21	8	922
Data Distribution	0	15		3,519
	5	10		2,986
	10	5		2,750
	15	0		1,378

Table 2: Execution Time on Multiple Sites

ACKNOWLEDGEMENTS

This article is based on work supported in part by Japan Society for the Promotion of Science (JSPS) Post Doctoral Fellowship for Foreign Researchers Program Grant (P 09260).

REFERENCES

API Reference, 2008. “Ninf-G Client API Reference”, http://ninf.apgrid.org/documents/ng/api.html, (accessed 28 Jun, 2008)

Akhter, S. and Roberts, J., 2006. *Multi-Core Programming – Increasing Performance through Software Multi-threading*. Intel Press. ISBN 0-9764832-4-6

Akhter, S., Jangjaimon, I., Chemin, Y., Uthayopas, P., Honda, K, 2006. Development of a GRIDRPC Tool for Satellite Images Parallel Data Assimilation in Agricultural Monitoring, *International Journal of GeoInformatics*, Vol 2, No 3.

Akhter, S., Sarkar, I., Rabbany, K.G., Akter, N., Akhter, S., Chemin, Y., and Kiyoshi, H., 2007. Adapting the LMF Temporal Splining Procedure From Serial toMPI/Linux Clusters. *Journal of Computer Science}3 (3): 130-133*, ISSN 1549-3636, Science Publications.

Akhter, S., Osawa, K., Nishimura, M., and Aida, K., 2008a. Experimental Study of Distributed SWAP-GA Models on the Grid. *IPSJ Transactions on Advanced Computing Systems*, Vol.1 No.2, pp.193-206.

Akhter, S., Aida, K. and Chemin, Y. 2008b. Asynchronous Grid LMF Processing of Vegetation Index for Asia, *International Journal of GeoInformatics*, Vol 4,No 4, pp. 39-45.

GRASSWiki, 2010, “OpenMP support in GRASS 7”, http://josef.fsv.cvut.cz/WIKI/grass-osgeo/index.php/OpenMP, (accessed 27 May, 2010)

Kamble, B. and Chemin, Y.H., 2006. “GIPE in GRASS Raster Add-ons”, http://grass.gdf-hannover.de/wiki/, GRASSAddOns, RasterAdd-ons, (accessed 26 Oct, 2007)

MPI, 2007. A tutorial, “Message Passing Interface”, http://www-unix.mcs.anl.gov/mpi/, (accessed 28 Oct, 2007)

Neteler, M. and Mitasova, H., 2003. *Open Source GIS: A GRASS GIS Approach*. Second Edition, Kluwer Academic Publishers.

Ninf-G, 2007. “Ninf-G Information Web Site”, http://ninf.apgrid.org/, (accessed 28 Oct, 2007)

Sawada, H., Sawada, Nagatani, Y. I., and Anazawa, M., 2001. In: *Proceeding for the 1st regional seminar on geo-informatics for Asian eco-system management*

Sawada, H., and Sawada, Y., 2002. Modeling of vegetation seasonal change based on high frequency observation satellite, *Environmental Information Science Paper*s, Vol. 16.

Sawada, Y., Mitsuzuka, N., and Sawada, H., 2005. Development of A Time-Series Model Filter for High Revisit Satellite Data, In: *Proceedings of the Second International Vegetation User Conference*

Tanaka, Y., Nakada, H., Sekiguchi, S., Suzumarn, T., Matsuoka, S., 2003. Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing. *Journal of Grid Computing* 1: 41-51.
.