Proceedings of the International Parallel & Distributed Processing Symposium, IEEE Computer Society Press, pp. x-xx, Nice, France, April 2003 (submitted).

# Distributed Geo-rectification of Satellite Images using Grid Computing

Y.M. Teo[*], S.C. Tay[**], and J.P. Gozali[*]
[*]Department of Computer Science
[**]Centre for Remote Imaging, Sensing and Processing
National University of Singapore
3 Science Drive 2
Singapore 117543
email: teoym@comp.nus.edu.sg

**Abstract**

Grid computing seeks to aggregate computing resources within an enterprise and leverage on resources you don't own for compute-intensive applications. Geo-rectification is a process for correcting spatial location and orientation of a satellite image. This paper focuses on the parallelization of the compute-intensive satellite image geo-rectification problem on a cluster grid. We discuss our approach to data and task partitioning, visualization technique and the archival of data. The computational tasks include wrapping satellite positional data to compensate the earth curvature, and consist of several steps such as image re-sampling, resolution conversion and image matching. Experimental results obtained using commodity PCs are discussed.

## 1. Introduction

Grid computing [11, 13] seeks to efficiently coordinate the sharing of geographically distributed computing resources, thereby bringing supercomputing power to its users. Unlike cluster computing [4, 5, 26] that is more constrained to computation on a local area networked of processors, grid computing enables applications to utilize resources that are spread across wide area networks. Globus [12] is a grid computing toolkit for the deployment of grid applications and systems. Nimrod/G [1, 3] is a resource management system for scheduling of grid applications built on top of Globus. Other grid systems such as Condor [24] and its Globus enabled variant, Condor/G [14], enable the harnessing of idle cycles. Grid-solvers have also been developed on these systems. In [15, 18], a web-based problem solving environment is introduced to simplify the submission, monitoring and steering of Master-Worker [17] based grid computing applications. In [7], architecture for matching grid application requirements to a set of heterogeneous grid resources is proposed. Our grid middleware, called ALiCE (Adaptive and scaLable Internet-based Computing Engine), is a portable software technology for developing and deploying general-purpose grid applications and systems. ALiCE aggregates and virtualises computer resources on the Internet/intranet into one computing environment through a platform-independent consumer-producer resource-sharing model, and harnesses idle resources for computation to increase the usable power of existing systems on the network.

Grid computing can be exploited for computationally intensive tasks such as protein folding, production simulation, operation research, climate modeling, etc. This paper discusses the application of grid computing in satellite remote sensing [6], specifically the *geo-rectification* of earth images. Geo-rectification is the correction of skew caused by the earth's curvature in raw satellite images. It establishes the image in the correct spatial location and orientation. Geo-rectification represents the projective transformation of a tilted photograph (model) to an output print assumed to be tilt free (assumes the correction is orthogonal to the surface, or a vertically corrected model). Pixels are relocated onto a new grid by extrapolating their true location from a source map or corrected image in which the new plane will represent a map projection system with a defined array of coordinates [25]. Figure 1 illustrates the geo-rectification process. A satellite image must be geo-rectified before it can be used in various geographic and scientific applications [22].

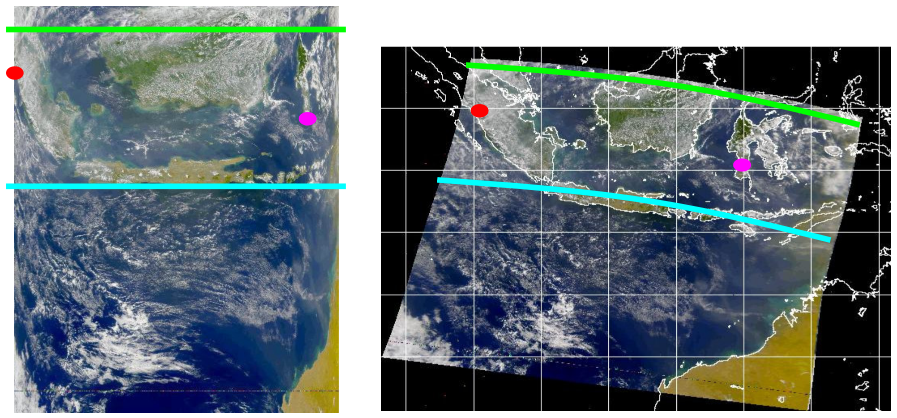

**Figure 1:** Geo-rectification of Satellite Image

Several projects have attempted to speed up remote sensing processes using parallel and distributed computing techniques. An ATM based wide area network is used in [18, 19] and a Beowulf cluster [27] is used in [30]. The challenge of using grid computing techniques for geo-rectification of satellite image data includes the partitioning of tasks and data, visualization methods for displaying of results and the final archival of data on secure storage. Partitioning of tasks and data is crucial since grid resources are usually heterogeneous in nature.

The rest of the paper is as follows. In section 2, we introduce ALiCE, our prototype Java-based grid computing system. In section 3, we discuss the geo-rectification process. Section 4 describes the distributed geo-rectification method on a grid computing system. In section 5 we present an analysis of the experiments conducted, and section 6 contains the concluding remarks.

## 2. ALiCE Grid Computing

ALiCE, developed at the National University of Singapore, is a Java-based grid computing middleware that supports the development and execution of generic Grid applications over a geographically distributed, heterogeneous collection of resources. The ALiCE middleware is written in Java and uses Sun Microsystems' Jini [29] and JavaSpaces [28] technologies for resource discovery and communication. Jini provides an adaptive network architecture that is scalable, evolvable and flexible suitable for a dynamic distributed computing environment. JavaSpaces is a tuplespace-based [10] object repository. ALiCE objects such as computation tasks, data, code and results are stored in JavaSpaces. As shown in Figure 2, ALiCE consumer-producer architecture consists of three entities: consumers, a resource broker, and producers. Grid applications are launched through a consumer GUI for execution at producers that share its compute cycles through the resource broker. The resource broker manages application execution and resource management.

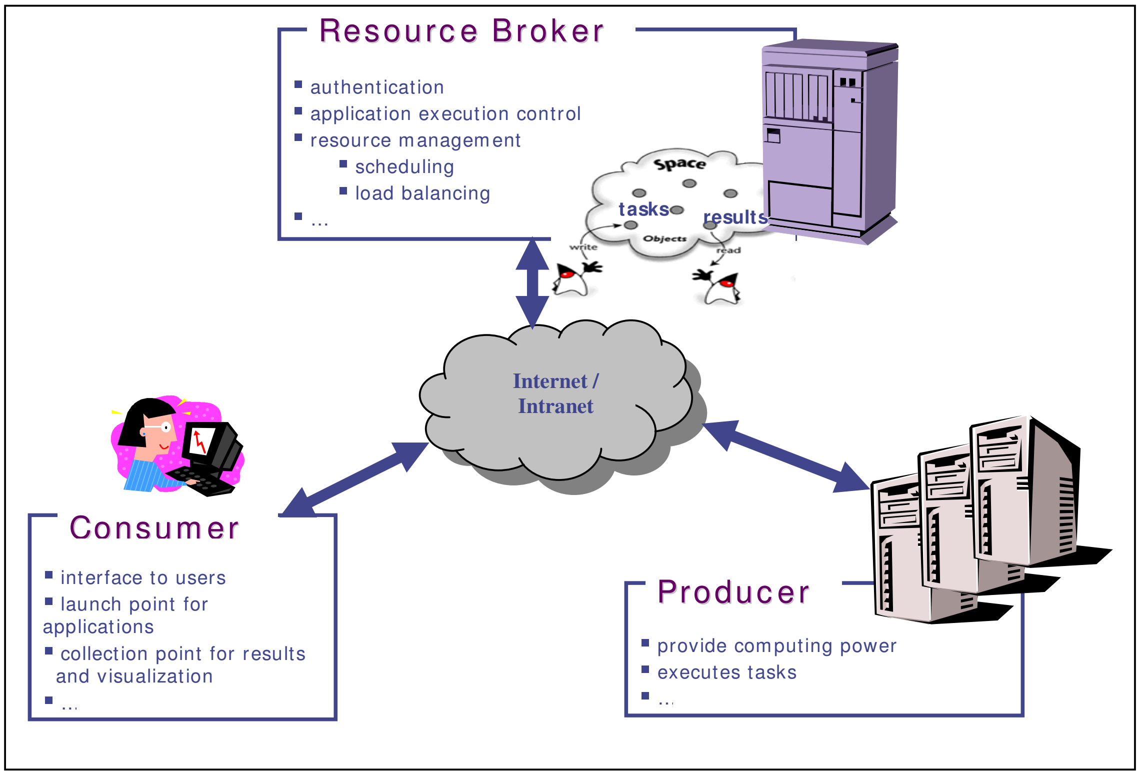

**Figure 2:** ALiCE Consumer-Producer Grid

ALiCE supports job parallelism to maximize throughput and object (task) parallelism to maximize performance. To exploit object parallelism, ALiCE Object-based Programming Template (AOPT) hides the complexities of parallel programming. The four main components of AOPT are shown in Table 1.

| Template | Function |
|---|---|
| TaskGenerator | • Allows application to be invoked at the resource broker by invocation of the user's main method.<br>• Provides methods for user applications to use to send tasks to ALiCE<br>• Hides dynamic code linking mechanisms. |
| ResultCollector | • Allows application to be invoked at the Consumer node, waiting for results to be returned from the Resource Broker.<br>• Provides methods for user applications to retrieve the results from Result Listener object residing in the consumer node. |
| Task | • Allows the producer nodes to return a Result object to the Resource Broker upon completing the execution.<br>• Allows the user to specify what functions to execute at the producer nodes. |
| Result | • Provides an interface for producer to instantiate and returns any evaluated or intermediate data.<br>• Allows user to store the result of execution. |

**Table 1:** ALiCE Programming Template

ALiCE uses the TaskGenerator-ResultCollector execution model as shown in Figure 3. When an ALiCE application is launched, its Java Archive (JAR) file containing the classes that implement the template is sent to the resource broker. The TaskGenerator at the Resource Broker initiates the application and produces a pool of tasks. These tasks are distributed for execution at Producers. Producers return result objects to the Resource Broker. To support visualization of data, the Result Collector is started at the consumer when the application is submitted. During the execution of the application, the ResultCollector at the Consumer received result objects for visualization from the Resource Broker. Alternatively, results are collected by the resource broker and returned to the consumer as a file.

We have developed several applications using ALiCE. These applications include the distributed Mandelbrot Set Generator, computational genomics applications such as protein alignment and sequence comparator, distributed equation solver, DES key search, N-body simulation, satellite image processing for red tide monitoring, etc.

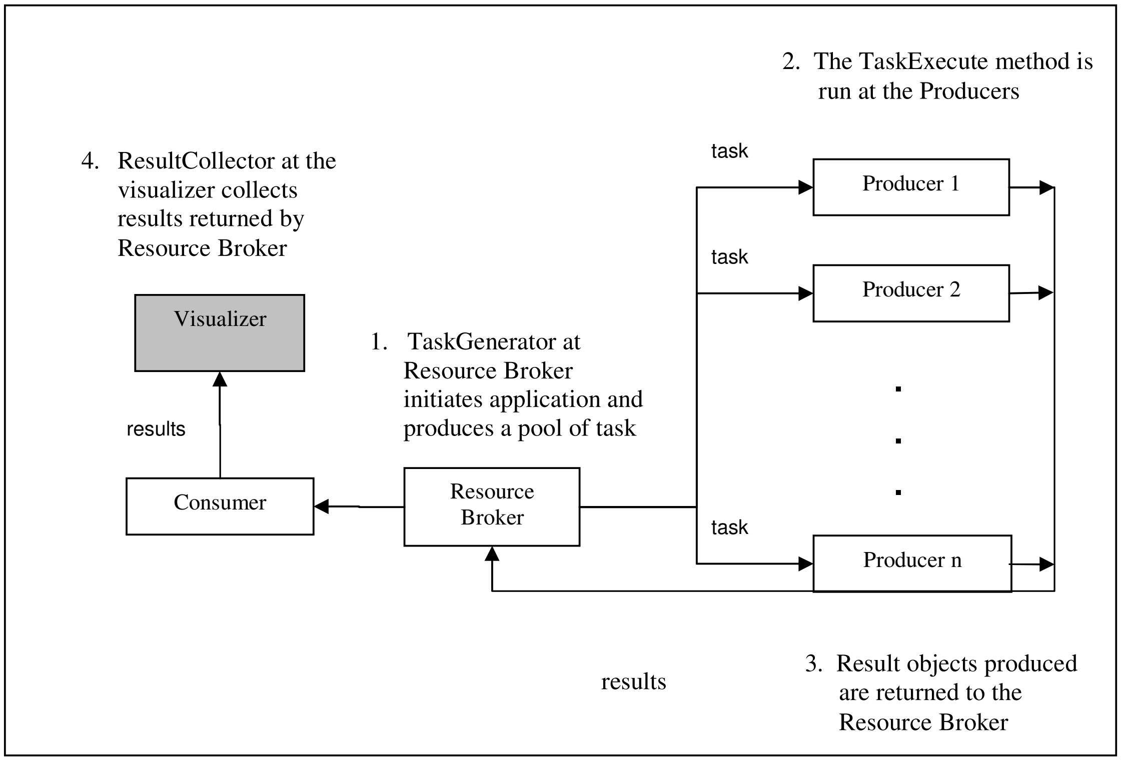
2. The TaskExecute method is run at the Producers
4. ResultCollector at the visualizer collects results returned by Resource Broker
task
Producer 1
task
Producer 2
Visualizer
1. TaskGenerator at Resource Broker initiates application and produces a pool of task
results
Consumer
Resource Broker
task
Producer n
results
3. Result objects produced are returned to the Resource Broker

**Figure 3:** ALiCE Execution Model

## 3. Sequential Geo-rectification

Figure 4 shows the seven main steps in the geo-rectification of satellite images. A geo-rectification process starts with the setting up of *geometric parameters* and *processing parameters*. Geometric parameters supply the sun and view zenith angles on the ellipsoid relative to a normal to that surface, as well as azimuth angles relative to local North. Processing parameters provide information about the area of interest, the level of detail, and the type of spectral band.

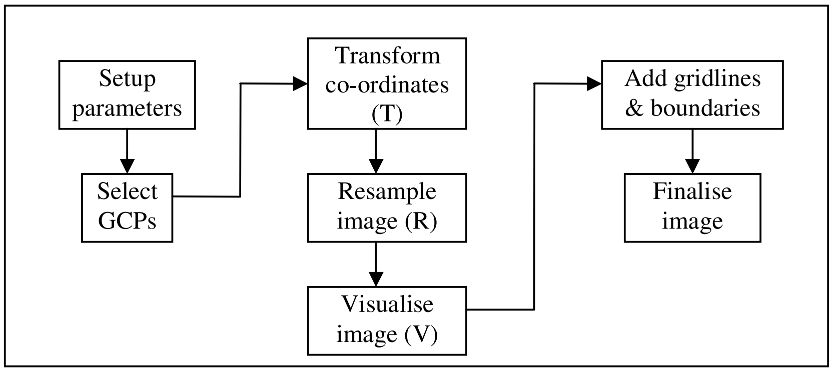

**Figure 4:** Sequential Geo-rectification Process

The next step is the selection of Ground Control Points (GCPs). These points are specific pixels in an image from which the output map coordinates are known. GCPs consist of two pairs of coordinates. The *source coordinates* refers to the coordinates in the image being rectified, while the destination *or known reference coordinates* are the coordinates of *the reference image* to which the source image is being registered. After identifying several well-distributed GCP pairs, the coordinate information is processed to determine the transformation equations to apply. The transformation maps the original (row and column) image coordinates onto their new ground coordinates.

After coordinate transformation, a procedure called *re-sampling* is used to determine the digital values to place in the new pixel locations of the corrected output image. The re-sampling process calculates the new pixel values from the original digital pixel values in the uncorrected image [8, 9, 23]. Three common methods for re-sampling are nearest neighbor, bilinear interpolation, and cubic convolution.

Following image re-sampling is the image generation step where *contrast enhancement* is applied. Contrast enhancement involves changing the original values so that more of the available range is used, thereby increasing the contrast between targets and their backgrounds. By manipulating the range of digital values in an image, graphically represented by its histogram in Figure 5, we can apply various enhancements to the data.

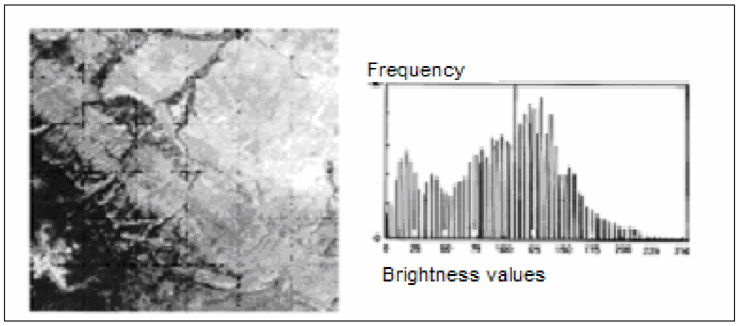

**Figure 5:** An Image and its Brightness Histogram

Next, grid lines and land boundaries are added to provide ground features and annotations to the image. This allows the clear demarcation of land and sea bodies as well as for marking out different land areas. In addition, the overlaying of boundaries allows areas that are not of interest to be masked out, thus focusing the attention to selected areas [2]. The addition of such lines enables the measurement of distance of an observed phenomenon from a fixed locale. An example of this application is the measurement of oil spills [25] and their distance from neighboring countries.

The last step of the geo-rectification process is the finalization of the geo-rectified image to permanent storage. In the Portable PixMap (PPM) format, an image is represented in terms of color values but exclude representation for various geophysical phenomena. When information of geophysical phenomena needs to be stored, the Hierarchical Data Format (HDF) format is used.

## 4. Distributed Geo-rectification using ALiCE Grid

Repetitive steps in the geo-rectification process can be executed simultaneously, thereby reducing the total execution time. ALiCE facilitates the parallel execution of the applications on a pool of networked computing resources.

We observe that the steps for transformation, re-sampling and visualization of image (TRV) are repeated many times for the entire image. Measurements on a single processor system showed that these three steps account for 60% to 90% of execution time. Figure 6 shows the seven steps grouped into three phases: setup, execution, and termination. *Setup* and *select* correspond to the two steps in the setup phase: setting up of geo-rectification parameters and the selection of GCPs. *T, R* and *V* correspond to the three steps in the execution phase: co-ordinate transformation, image re-sampling and visualization. *Draw* and *finalize* correspond to the two steps of the termination phase: drawing of boundary lines and the finalization of the output image. Both the setup and the termination phases are sequential.

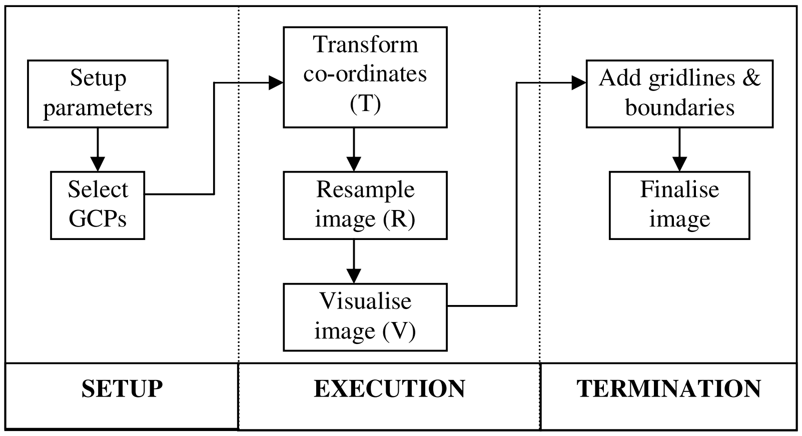

**Figure 6:** Three Main Phases in Geo-rectification

Figure 7 shows a sequential and a Grid-based geo-rectification. In Grid-based geo-rectification, the TRV steps are executed in parallel as task objects. This application is amendable to geometric partitioning (single program multiple data) as it performs the same set of TRV operations on different sets of data. This homogeneity can be exploited by executing different partitions of the image in parallel on different processors.

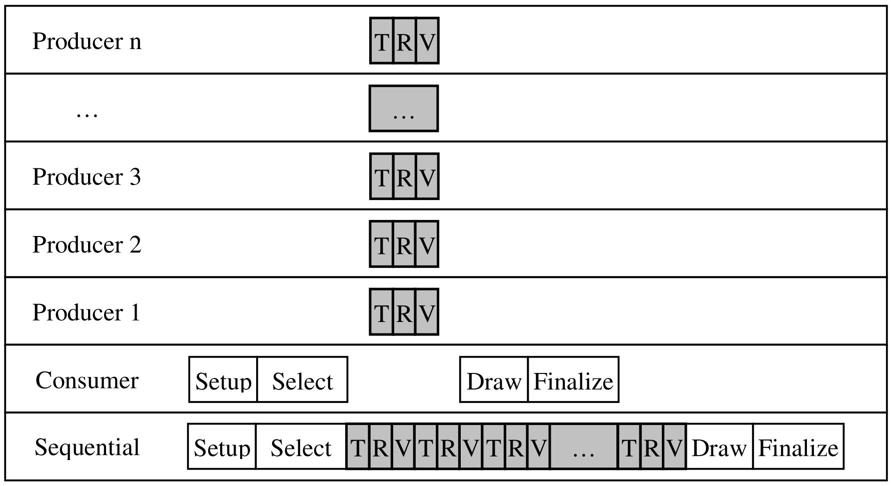

**Figure 7:** Sequential versus Grid-based Geo-rectification

A typical image file size is of the order of hundreds of megabytes. To avoid having to send large image chunks over the network, we employ a variant of *geometric partitioning* [16, 21]. A task object contains only parameters that describe an image chunk; it does not contain the image chunk data. Each partition contains only the parameters for calculating the matrix to map pixels from the source image to the output image. A task execution produces a result object consisting of a *mapping matrix* for a particular image partition. As shown in Figure 8, a mapping matrix for an image partition provides pixel locations that are used to construct the geo-rectified image. In step 1, a mapping matrix is obtained from a result object. In step 2, pixel values are read from the source image based on the pixel locations that are provided in the mapping matrix. In step 3, the pixel values are read from the source image, and are written to the output image as the pixels for the geo-rectified image chunk in step 4.

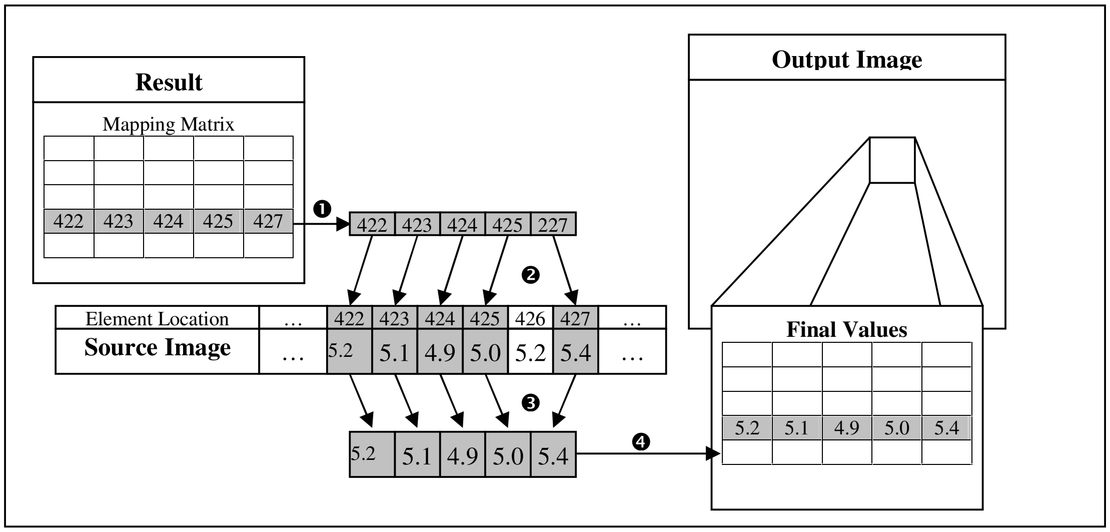

**Figure 8:** Result Mapping Matrix Processing to Produce a Geo-rectified Image Partition

Setup and termination phases of the geo-rectification process involve interaction with the user; therefore, they are implemented as part of the Result Collector. Parallel execution is achieved by partitioning the image data into task objects. The partitioning takes place at the Task Generator. Each task will apply co-ordinate transformation and image re-sampling on the image chunk that it is assigned to.

As shown in Figure 9, a TaskGenerator is sent to the Resource Broker; Task generated are sent to Producers; Producers execute the Task and package output in Result; Result objects sent back to the Resource Broker; Result objects sent to the ResultCollector on the Consumer; Result objects retrieved and visualized at the Consumer. The code segments for Task, Task Generator and Result Collector are shown in Figure 10.

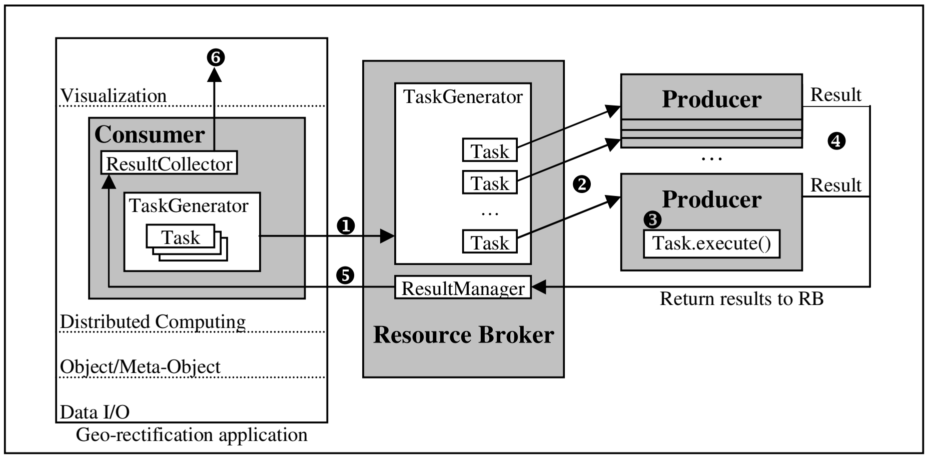

**Figure 9:** Geo-rectification on ALiCE

```
public class swTask implements Task {

  public Result execute()  {
     // .. initialize data structures

     for (int j=0;j<chunkHeight;j++) {
       for (int i=0;i<chunkWidth;i++)  {
          /* perform coordinate
               transformation
             & image resampling */
       }
       return result;
     }
  }// end execute()

   // .. other methods
   //
}// end class
```

```
public class swGenerator extends TaskGenerator {

 public void generateTasks() {
    // ... initialize variables

    for (int h=0;h<cHeight;h++) {
      for (int w=0;w<cWidth;w++)       {
        swTask _swTask=new swTask(); // create task
        process(_swTask); // send task to Res. Broker
        taskGenerated++;
      }
    }
  } // end generateTask()

   // .. other methods
   //
} // end class
```

```
public class swTaskListener extends ResultCollector {

  // .. initialize data structure

  public void run() {
    while (resultsCollected<resultsExpected) {
      swTask result=((swTask)swGUI.collectResult()); // collect results
      vResults.add(result); // store result for visualization

      // .. datastructure maintenance code
    }
  } // end run()

  // .. other methods
  //
}// end class
```

**Figure 10:** Code Segments for Task, Task Generator and Result Collector

## 5. Experiments and Analysis

Our experiments were carried out on a low cost commodity PC cluster consisting of eight nodes connected using a 100 Mbps Ethernet switch. Each node is an Intel P3-866Mhz with 256 MB of memory and runs RedHat Linux 7.0. The resource broker and producers are installed with Hotspot Server Virtual Machines and client node runs the Hotspot Client Virtual Machine.

Two image files of varying size were used – image-file 1 (108 MB), and image-file 2 (136 MB). Two different resolutions, 0.025 and 0.01 are used on each image-file. A smaller resolution means a bigger map, and thus, a bigger problem size. The elapsed time for each run ($T_p$) is defined as the total time for the generation of tasks at the resource broker, communication time in shipping tasks to producers and producer's execution time, and the time incurred in collecting results by the resource broker. We define the speedup ($S_p$) and efficiency ($E_p$) using the following ratios:

$$S_p = T_s / T_p$$

$$E_p = (S_p / p) * 100\%$$

Table 2 shows the sequential geo-rectification execution time on an Intel P3-866MHz computer. Based on our partitioning strategy, the image is partitioned into tasks of various sizes. For example, a task size of 1000x1000 will results in 25 tasks with an estimated execution time of 25 seconds per task. Selection of task granularity and number of tasks are important in load balancing and scalability experiments.

Test Data file = s2000144031151.l2 (108,316,813 bytes)
Resolution = 0.01

| No. of tasks | Task Size | Execution Time (sec) | Execution Time per task (sec) |
|---|---|---|---|
| 5 | 2000x2000 | 654.4 | 130.9 |
| 25 | 1000x1000 | 636.1 | 25.4 |
| 60 | 1000x400 | 609.1 | 10.2 |
| 132 | 400x400 | 608.7 | 4.6 |
| 253 | 400x200 | 648.8 | 2.7 |
| 506 | 200x200 | 836.9 | 1.7 |

**Table 2:** Image-File 1 – Sequential Time versus Number of Tasks

Table 3 shows the performance for varying number of tasks and producers. The execution time increases when the number of tasks is decreased due to insufficient number of tasks to keep all the producers occupied. ALiCE uses eager task- scheduling. In addition, a task buffer at each producer masked the round trip time overhead in task scheduling.

| Test Data | Task Size | No. of Tasks | Sequential Time, $T_s$ (sec) | No. of Producers | Execution Time, $T_p$ (sec) | Speedup, ($S_p$) | Efficiency ($E_p$) |
|---|---|---|---|---|---|---|---|
| Image-file 1 (108 MB) Resolution: 0.025 | 400x400 | 25 | 106.4 | 2 | 56.1 | 1.9 | 94.8 |
| | | | | 4 | 42.2 | 2.5 | 63.0 |
| | | | | 6 | 40.2 | 2.7 | 44.2 |
| | 400x200 | 45 | 101.5 | 2 | 61.8 | 1.6 | 82.1 |
| | | | | 4 | 47.7 | 2.1 | 53.2 |
| | | | | 6 | 32.0 | 3.2 | 52.9 |
| Image-file 1 (108MB) Resolution 0.01 | 1000x1000 | 25 | 636.1 | 2 | 421.3 | 1.5 | 75.5 |
| | | | | 4 | 252.8 | 2.5 | 63.0 |
| | | | | 6 | 198.1 | 3.2 | 53.5 |
| | 1000x400 | 60 | 609.1 | 2 | 308.4 | 1.99 | 98.8 |
| | | | | 4 | 176.0 | 3.5 | 86.5 |
| | | | | 6 | 128.6 | 3.7 | 78.8 |
| Image-file 2 (136 MB) Resolution: 0.025 | 1000x400 | 12 | 140.7 | 2 | 79.3 | 1.8 | 88.5 |
| | | | | 4 | 59.2 | 2.4 | 59.5 |
| | | | | 6 | 42.2 | 3.3 | 55.5 |
| | 400x400 | 30 | 124.4 | 2 | 64.9 | 1.9 | 96.0 |
| | | | | 4 | 52.9 | 2.4 | 58.8 |
| | | | | 6 | 50.8 | 2.5 | 40.8 |

**Table 3**: Varying Number of Tasks and Number of Producers

The experiments above show the distributed processing of one satellite image on a grid of processors. Typically, each satellite transmits about ten satellite images to a ground station per visit. A ground station receives images for up to ten satellites per day. To process all the images received in a day, multiple satellite images can be distributed over the ALiCE grid to exploit job-level parallelism.

To further demonstrate the capability of the grid on low cost commodity processors, we conducted an experiment using a 50MB image file partitioned into thirty tasks. Using eight slow Pentium II (400MHz) PCs connected via 10Mbps Ethernet, an execution time of 0.8 minute was recorded versus the original application written in C/C++ that took 13.1 minutes on a Sun UltraSPARC machine.

## 6. Conclusions

As many real-world applications for computing have high demand for processing powers, it is critical to have a platform that readily supply such computing powers with the ease of use. Grid computing, by pooling un-used CPU cycles together, has good potential in meeting the needs of high performance computing. We have implemented a grid-based geo-rectification system using our Java-based Grid computing system ALiCE (Advanced and scaLable Internet-based Computing Engine). We focus on the parallelization of the problem and its mapping onto a grid system. In the area of exploiting parallelization, we have already observed the percolation of parallelization techniques to the processor level, with superscalar pipelining processors, symmetric multi-processor (SMP) systems and recently, symmetric multi-threading (SMT) processors pushing the envelope of performance even further. We see great gains in studying how parallelization at various hardware and software levels can be used for maximization of performance. More importantly, grid computing can readily bring high performance computing to its users by harnessing idle compute cycles from existing IT infrastructure without additional costly hardware investment.

Ongoing work on system problem includes the scalability of ALiCE on wide-area network, load balancing algorithm with quality of service, and cluster-based resource broker with data server to support datagrid applications. In the area of grid programming problem, we are developing grid programming model and environment to support computational genomics.

**Acknowledgement**
We thank S.C. Low for implementing the first version of this platform and his valuable contribution to the project.

## References

1. Abramson D., Giddy J., and Kotler L., High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?, *Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS) 2000* , Mexico, IEEE CS Press, USA, 2000.
2. Bugayevskiy L.M., and Snyder J.P., Map Projections: A Reference Manual, Taylor and Francis, London, United Kingdom, 1995.
3. Buyya R., Abramson D., and Giddy J., Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid, *Proceedings of the 4th International Conference and Exhibition on High-Performance computing in the Asia-Pacific Region (HPCASIA'2000)*, China, IEEE CS Press, USA, 2000.
4. Buyya R., High Performance Cluster Computing: Programming and Applications Vol.2, Prentice Hall PTR, New Jersey, USA, 1999.
5. Buyya R., High Performance Cluster Computing: System and Architectures Vol.1, Prentice Hall PTR, New Jersey, USA, 1999.
6. Campbell J.B., Introduction to Remote Sensing 2nd ed., Guildford Press, New York, USA, 1996.
7. Czajkowski K., Foster I., and Kesselman C., Resource coallocation in computational grids. *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HDPC-8)*, 1999.
8. ERDAS, ERDAS Field Guide, 4th Ed., ERDAS Inc. Atlanta, USA, 1997.
9. ESRI, *Fifteenth Annual ESRI User Conference Proceedings 1995*, 1995.
10. Fenwick J.B., Jr. and Pollock L.K., Issues and experiences in implementing a distributed tuplespace. Technical Report TR 9706, University of Delaware, 1996.
11. Foster I., and Kesselman C., editors. The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers, 1999.
12. Foster I., and Kesselman C., Globus: A Metacomputing Infrastructure Toolkit, *Proceedings of the Workshop on Environments and Tools for Parallel Scientific Computing*, SIAM, Lyon, France, 1996.

13. Foster I., Kesselman C., and Tuecke S., *The* Anatomy of the Grid: Enabling Scalable Virtual Organizations, *International Journal on Supercomputing Applications 2001*, 2001.
14. Frey J., Tannenbaum T., Livny M., Foster I., Tuecke S., Condor-G: A Computation Management Agent for Multi-Institutional Grids, *Proceedings of the 10th International Symposium on High Performance Distributed Computing (HPDC-10)*, IEEE Press, 2001.
15. Good M., and Goux J.P., iMW : A web-based problem solving environment for Grid computing applications. *Technical report*, Department of Electrical and Computer Engineering, Northwestern University, 2000.
16. Goodrich M.T., Geometric Partitioning Made Easier, Even in Parallel, *Proceedings of the 9th ACM Symposium on Computational Geometry*, 1993.
17. Goux J.P., Kulkani S., Linderoth J., and Yoder M., An enabling framework for master-worker applications on the computational grid., *Proceedings of 9th IEEE International Symposium on High Performance Distributed Computing (HDPC-9)*, 2000.
18. Hawick K.A., and James H.A., A Web-based Interface for On-Demand Processing of Satellite Imagery Archives. *Proceedings of the Australian Computer Science Conference (ACSC) '98*, Perth, Australia, 1998.
19. Hawick K.A., and James H.A., Distributed High-Performance Computation for Remote Sensing, *Proceedings of Supercomputing '97*, San Jose, USA, 1997.
20. Heng W.A. and Lim H., Oceanic phenomena observed in satellite data collected at CRISP, *Oceanology International '97 Pacific Rim*, 1997.
21. Hu Y.C., Teng S.H., and Johnsson S.L., A Data-Parallel Implementation of the Geometric Partitioning Algorithm, *Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing*, Minneapolis, USA, 1997.
22. Jenson J.R., Introductory Digital Image Processing: A Remote Sensing Perspective. Prentice Hall,, New Jersey, USA, 1986.
23. Lillisand T.M., and Kiefer R.W., Remote Sensing and Image Interpretation 3rd ed., John Wiley and Sons Inc, 1994.
24. Litzkow M.J., Livny M., and Mutka M.W., Condor -- A Hunter of Idle Workstations. *Proceedings of the 8th International Conference on Distributed Computing Systems*, 1988.
25. Lu J., Lim H., Liew S.C., Bao M., and Kwoh L.K., Ocean oil pollution mapping with satellite imagery in Southeast Asia (2000). *28th International Symposium on Remote Sensing of Environment*, Cape Town, South Africa, 2000.
26. Pfister G.F., In Search of Clusters, Prentice Hall PTR, New Jersey, USA, 1998.
27. Sterling T.L., Salmon J., Backer D.J., and Savarese D.F., How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters 2nd Printing, MIT Press, Cambridge, Massachusetts, USA, 1999.
28. Sun Microsystems. JavaSpace Specification, March 1998.
29. Waldo J., The Jini Architecture for Network-centric Computing. Communications of the ACM, pages 76--82, July 1999.
30. Yang C.T., Using a Beowulf Cluster for a Remote Sensing Application, *Proceedings of the Asian Conference on Remote Sensing (ACRS) 2001*, Singapore, 2001.