Porting a GRASS raster module to distributed computing

Examples for MPI and Ninf-G

Shamim Akhter, Yann Chemin, Kento Aida

Abstract

Satellite imagery provides a large amount of useful information. To extract this information and understand them may require huge computing power and processing time. Distributed computing can reduce the processing time by providing more computational power. GRASS, an open source software, has been used for processing the satellite images. To let the GRASS modules benefit from distributed computing, an example module r.vi is ported in both MPI (r.vi.mpi) and Ninf-G (r.vi.grid) programming modules. Their implementation methodologies are the main discussion issue of this paper which will guide the basic way of representing any GRASS raster module in distributed platform. Additionally, a comparative study on modified r.vi, r.vi.mpi and r.vi.grid is presented here.

Introduction

Satellite image processing plays a vital role for research developments in Remote Sensing, GIS, Agriculture Monitoring, Disaster Management and many other fields of study. However, processing those even increasing spatial resolution satellite images require a large amount of computation time due to its complex and large processing criteria. This seems a barrier for real time decision making. Distributed computing can be a suitable solution to complete the job timely. Cluster and Grid are two well-known distributed systems have been associated with high performance computing of massive CPU bound applications. GRASS GIS (Neteler et al. , 2003) is an open source software/tool, which has been used to process satellite images. Inside GRASS, different modules have been developed for processing satellite images. GRASS module r.vi is developed by (Kamble et al. , 2006), and is used as a test example for this study. Developing the methodology, which enables to run GRASS GIS environment for satellite image processing on distributed computing systems, is the main concerning issue of this paper. Additionally, two different implementation methodologies for distributed r.vi are discussed for two different programming platforms MPI (Message Passing Interface , MPI) and Ninf-G (Ninf-G Homepage. , 2007).

Vegetation Index (VI) is the major set of indicators for vegetation. The GRASS module r.vi, is used to process 13 different vegetation indexes for the satellite images. NDVI (Normalized Difference Vegetation Index (Weier et al. , 2007) is one of them. The NDVI is calculated from these individual measurements as NDVI=(NIR-Red)/(NIR+Red), where RED and NIR stand for the spectral reflectance measurements acquired in the red and near-infrared regions, respectively. Other vegetation indices are as follows:

Contents of this volume:

RVI	Ratio Vegetation Index
IPVI	Infrared Percentage Vegetation Index
DVI	Difference Vegetation Index
PVI	Perpendicular Vegetation Index
WDVI	Weighted Difference Vegetation Index
SAVI	Soil Adjusted Vegetation Index
GARI	Green Atmospherically Resistant Vegetation Index
MSAVI	Modified Soil Adjusted Vegetation Index
MSAVI2	Second Modified Soil Adjusted Vegetation Index
GEMI	Global Environmental Monitoring Index
ARVI	Atmospherically Resistant Vegetation Index
GVI	Green Vegetation Index

Figure 1: Vegetation Indexes

They are derived using various methods of differentiations and contrast. Fig. 2 shows a snapshot of an output of GRASS Software.

GRASS module r.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 Infra-Red) band images. However, ARVI uses red, NIR and blue band images, GVI uses red, NIR, blue,green, chan5 and chan 7 of landsat images and GARI uses red, NIR, blue and green bands. 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 sent 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. Fig. 3 presents the structure of serial running r.vi module (for simplicity only two band images have been presented).

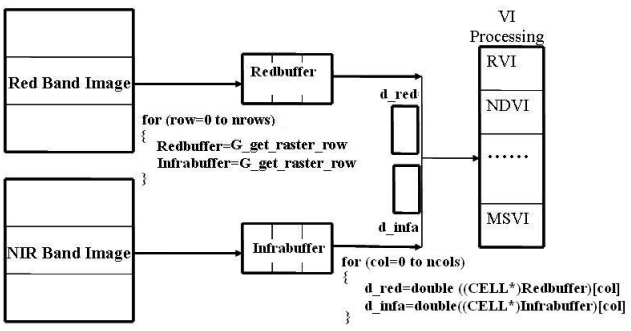

Figure 3: Serially Running r.vi Module Structure

Objectives

Under the assumption that distributed style of computing will remove computational time constraints, new image processing distributed algorithms for remotely sensed images can be considered. Different GRASS modules have been developed for solving different Remote Sensing Image analysis problems. To evaluate the performance of the GRASS modules in distributed computing environment is the major objective of this paper. Additionally, it is a new dimension for the remote sensing users to port their jobs in distributed computing environment. Another issue addressed in this paper is to find out the sufficient amount of workload that is needed to dispatch among the worker nodes for getting better performance than the serial module.

Methodology

To fulfill the above objectives or requirements, the GRASS module (r.vi) has been parallelized by using the master-worker model. 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 r.vi is implemented using MPI on a PC cluster system (r.vi.mpi) and Ninf-G on the same PC cluster system (r.vi.grid) to hold the similarity between the experimental environments. However, r.vi.grid module has been structured as it is capable to run in distributed GRID system. Additionally, experiments were done to analyse the results with distributed r.vi modules by increasing the number of operations, to find out the amount of workload that is necessary to get a performance benefit from the distributing environments.

In Fig. 4, the implementing structure of distributed r.vi is presented (for simplicity only two band images have been presented). Here, S1, S2, ...,Sn are different worker processes.

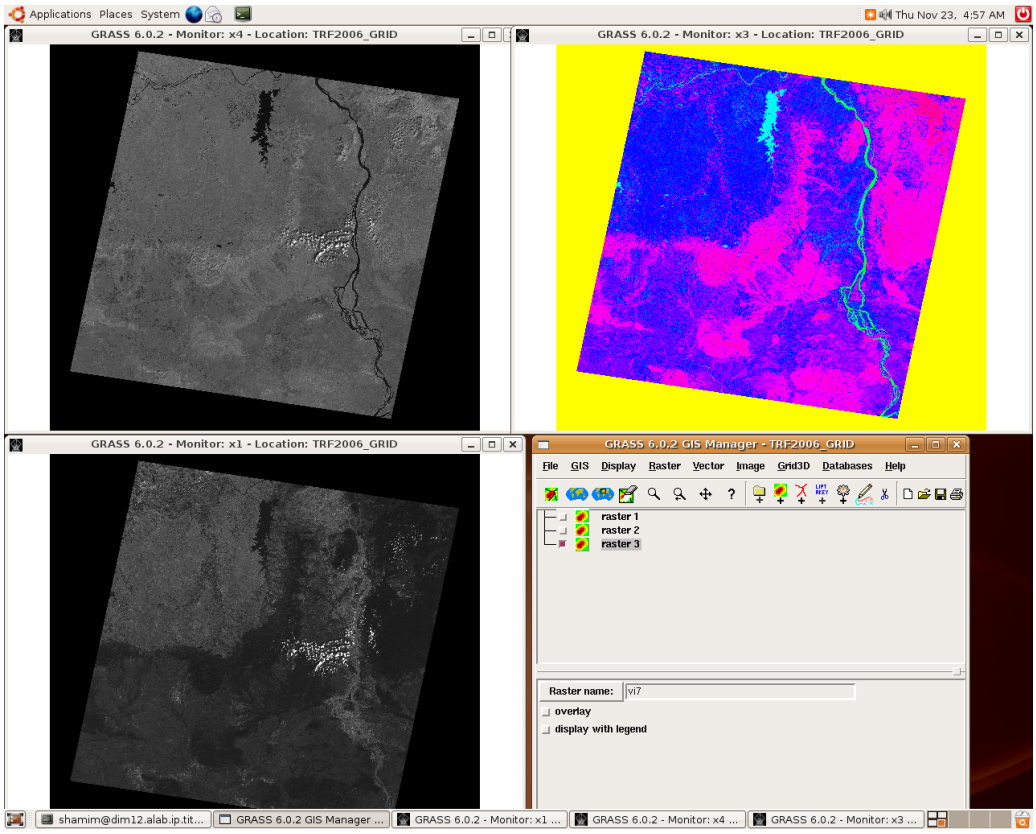

Figure 2: NDVI calculations with GRASS

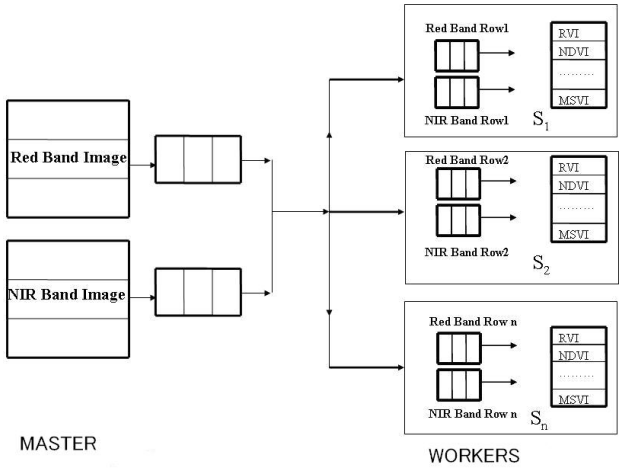

Figure 4: Distributed r.vi Module Structure (r.vi.mpi and r.vi.grid)

Implementation

MPI and Ninf-G Framework

MPI (Message Passing Interface) is a library of functions (in C) or subroutines (in FORTRAN) that one can insert into the source code to perform data communication between processors (Sain et al. , 1996). MPI was designed for high performance on both massively parallel machines and on workstation clusters and developed by a broadly based committee of vendors, implementors, and users.

The Ninf-G has been developed by AIST (National Institute of Advanced Industrial Science and Technology, Japan) and TITECH (Tokyo Institute of Technology, Japan). Ninf-G is a reimplementation of the Ninf system on top of the Globus Toolkit (Foster et al. , 1997). Globus serves as a robust and common platform for implementing higher level middleware and programming tools, etc., ensuring interoperability amongst such high level components, one of which being NinfG. NinfG system is based on client server computing. The computational resources are available as remote libraries at a remote computation host which can be called through the global network from client program written in existing languages such as FORTRAN, C, or C++.

Specification of the Davinchi Cluster

The Cluster nodes for the experiments are as follows:

Number of hosts	4
Host Spec	Each Host 2 CPUs(Xeon 2.4GHz x 2) 512 KB Cache Size 1 GB Hard Disk
GRASS Version	6.0.2
MPICH Version	1
Ninf-G Version	4.1.0
Globus Toolkit Version	4.0.3
Local Job Manager for r.vi.grid module	SGE (SUN Grid Engine, 2007)

Figure 5: Specifications of Davinchi Cluster

Distributed r.vi Module in MPI (r.vi.mpi)

For r.vi.mpi, GRASS source code is installed in master node and GRASS libraries are copied to the slaves local memory location exactly the same as master (/usr/local/grass-6.0.2/) and the following configuration steps are completed to setup the MPI-GRASS environment.

- grass.conf file is created inside /etc/ld.so.conf.d/ directory and the required grass library names are written inside this file.

- Then the /sbin/ld config command is executed.

MPI_Send and MPI_Recv both functions have been used for data communication.

The following Makefile is created for compilation

```
MODULE_TOPDIR = ../..
CC=mpicc
PGM = r.vi.mpi
LIBES = \$(GISLIB) \$(GMATHLIB)
DEPENDENCIES = \$(GISDEP) \$(GMATHDEP)
include \$(MODULE_TOPDIR)/include/Make/Module.make
default: cmd
```

To run the code the following shell command has been used:

```
mpirun -np 3 location_of_exec_file parameters..
```

In this experiment the location of the executable file is: $GRASS_COMPILE_DIR$/dist.i686-pc-linux-gnu/bin/r.vi.mpi The pseudo-code of r.vi.mpi is appended here:

```
#include "gis.h"
#include "glocale.h"
#include "mpi.h"


/* main.c: Declare the following MPI code */
/* NUM_HOSTS is total host number         */
/* me is defined processor rank/number    */

MPI_Status status;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&NUM_HOSTS);
MPI_Comm_rank(MPI_COMM_WORLD,&me);

/*---------------------------------------*/
/* Master Code Begin: Rank 0 (me=0)      */


/* Extract total row and column number   */
/* and sends them to the slaves          */


for(i=1;i<NUM_HOSTS;i++)
/* i Loop Started:

 MPI_Send(&nrows,1,MPI_INT,i,1,MPI_COMM_WORLD);
 MPI_Send(&ncols,1,MPI_INT,i,1,MPI_COMM_WORLD);

/* i Loop Finished.

/* Row data are extracted from images and   */
/* dispatched them among slaves with specific*/
/* row number by Round Robin Fashion        */

for (r = 1; r*(NUM_HOSTS-1) <= nrows;r++ )
/* r Loop Started:

 for(k=1;k<NUM_HOSTS;k++)
 /* k Loop Started:

   row=(r-1)*(NUM_HOSTS-1)+k-1;
   G_get_raster_row(infd_redchan,...
   G_get_raster_row(infd_nirchan,...
   for (col=0; col < ncols; col++)
   /*col Loop Started:

      /*Each column cell values form all bands
      /* are extracted and put them in a 2D array
     db[0][col]= d_redchan;
     db[1][col]= d_nirchan;

   /* col Loop Finished.

   row_n=k-1;
   I[ncols]=row_n;
   MPI_Send(I,ncols+1,MPI_INT,k,1,\
   MPI_COMM_WORLD);
   MPI_Send(db,6*ncols,MPI_DOUBLE,k,1,\
   MPI_COMM_WORLD);

 /* k Loop Finished.

 /* Waiting for the result...

 for(k=1;k<NUM_HOSTS;k++)
 /* k Loop Start:

   MPI_Recv(R,ncols+1,MPI_DOUBLE,k,1,\
   MPI_COMM_WORLD,&status);
   row_n=R[ncols];
   for (cn=0;cn<ncols;cn++)
   /* cn Loop Started:

outputImage[row_n][cn]=R[cn];
```

```
   /* cn Loop Finished.

 /* k Loop Finished.

  /* Processes row put back to the result images
 for(k=0;k<(NUM_HOSTS-1);k++)
 /* k Loop Start:

  for(j=0;j<ncols;j++)
  /* j Loop Start:

  ((DCELL *) outrast)[j] = outputImage[k][j];
  G_put_raster_row(outfd,outrast,data_type_output);

  /* j Loop Finished.

 /* k Loop Finished.

/* r Loop Finished.

/* If any row left when (row_number%slaves!=0), */
/* the round robin fashion rest rows (lets n)   */
/* are distributed from slave 1 to slave n again*/

MPI_Finalize();
G_free(inrast_redchan);

/* Master code Finished                         */
/*----------------------------------------

/*----------------------------------------
/* Slave Code Begin: Rank not 0 (me!=0)       */

MPI_Recv(&nrows,1,MPI_INT,0,1,MPI_COMM_WORLD,\
&status);
MPI_Recv(&ncols,1,MPI_INT,0,1,MPI_COMM_WORLD,\
&status);

n_rows=nrows/(NUM_HOSTS-1);
modv=nrows%(NUM_HOSTS-1);
if(modv>=me)
n_rows++;

/* Data Receiving from master and process
for(i=0;i<n_rows;i++)
/* i Loop Started:

  MPI_Recv(I,ncols+1,MPI_INT,0,1,MPI_COMM_WORLD,\
  &status);
  MPI_Recv(db,6*ncols,MPI_DOUBLE,0,1,MPI_COMM_WORLD,\
  &status);

  for (col=0; col<ncols; col++)
  /* col Loop Started:

   ndvi(db[0][col],db[1][col]);// Process ndvi()

  /* col Loop Finished.

  r[ncols]=I[ncols];

   /* results are in r[] and
   /* sends back to master
  MPI_Send(r,ncols+1,MPI_DOUBLE,0,1,\
  MPI_COMM_WORLD);

/* i Loop Finished.

 MPI_Finalize();

/* Slave Code Finished                        */
/*----------------------------------------
```

Distributed r.vi Module in GRID (r.vi.grid)

r.vi is easier to port in Ninf-G environment than MPI. Here, no need to copy the library files, because the worker procedure is totally independent from the master procedure and GRASS environment. GridRPC (Tanaka et al. , 2003) calling API is used for communication between the master and workers. The following Makefile is created for compilation

```
MODULE_TOPDIR = ../..
CC=ng_cc
PGM = r.vi.grid
LIBES = \$(GISLIB) \$(GMATHLIB)
DEPENDENCIES = \$(GISDEP) \$(GMATHDEP)
include \$(MODULE_TOPDIR)/include/Make/Module.make
default: cmd
```

Like any other GRASS modules r.vi.grid can run by the command

```
r.vi.grid  parameters
```

Worker IDL file is as follow:

```
Module VI_Server;
Define VI_CALC (IN int n,IN double I[n], \
IN double a[n], IN double b[n], IN double c[n],\
IN double d[n], IN double e[n],\
IN double f[n], OUT double r[n])
Required "VI_ServerC.o"
Calls "C" VI_CALC(n,I,a,b,c,d,e,f,r);
```

Worker code is as follow:

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<unistd.h>

void VI_CALC(int n, int *I, double *a,\
 double *b,double *c, double *d, double *e,\
 double *f,double *r){
 int col;


 for (col=0; col<n; col++)
 //col Loop Started

    /*coloum wise value going to process for
    /* specific indexing, put results in r []
```

```
/* as for ndvi processing....
r[col]=(a[col]-b[col])/(a[col]+b[col]);

 //col Loop Finished.

} //function call finish
```

Master module code is as following:

```
#include "gis.h"
#include "glocale.h"
#include "grpc.h" // we need grpc.h header file
#define NUM_HOSTS 5 //how many hosts available

char* hosts[]= {"davinchi1.alab.ip.titech.ac.jp",\
"davinchi1.alab.ip.titech.ac.jp", ... , ... , ... ;
grpc_function_handle_t handles[NUM_HOSTS];
grpc_sessionid_t ids[NUM_HOSTS];
int ret;


if((ret=grpc_initialize("/.../raster/r.vi.grid \
 /vi.conf")!=GRPC_NO_ERROR)){
  fprintf(stderr, "Error in grpc_initialize, \
  %d\n",ret);
  exit(2);
}


//creating one handle for each host node
for(i = 0; i < NUM_HOSTS; i++)
 grpc_function_handle_init(&handles[i], hosts[i],\
    "VI_Server/VI_CALC");


for(row = 0; row < nrows; row++)
/* row Loop Started:


  host_n=host_n%NUM_HOSTS;
  if(G_get_raster_row(infd_redchan,\
      inrast_redchan,row,data_type_redchan)<0)
    G_fatal_error(_("Could not read from <%s>"),\
      redchan);
  if(G_get_raster_row(infd_nirchan,\
      inrast_nirchan,row,data_type_nirchan)<0)
    G_fatal_error(_("Could not read from <%s>"),\
      nirchan);


  for(col=0; col < ncols; col++)
  /* col Loop Started:


     /* each column cell values for all
     /* band images are extracted together
     /* and put them in 2D arrays
    db[0][col]= d_redchan;
    db[1][col]= d_nirchan;
    db[2][col]= d_greenchan;
    db[3][col]= d_bluechan;
    db[4][col]= d_chan5chan;
    db[5][col]= d_chan7chan;
    /* vegetation indexing array  valuess are
    /* filled in I[ncols] array


  /* col Loop Finished.


  if(grpc_call(&handles[host_n],ncols,I,db0,db1,\
  db2, db3,db4,db5, R) != GRPC_NO_ERROR){
    fprintf(stderr,"grpc_call ERROR\n");
    exit(2);
  }


   /* All outputs are put back into raster
  for(j=0;j<ncols;j++)
  /* j Loop Started:


    ((DCELL *) outrast)[j] = R[j];


  /* j Loop Finished.


   if(G_put_raster_row(outfd,outrast,\
      data_type_output) < 0)
   G_fatal_error(_("Cannot write to output\
      raster file"));


  host_n++;


/* row Loop Finished.


 /*Destruct the handles
for(i = 0;i < NUM_HOSTS; i++)
  grpc_function_handle_destruct(&handles[i]);
grpc_finalize();
G_free(inrast_redchan);
G_close_cell(infd_redchan);
...etc...(Free memory)
```

Experimental Results

Fig. 6 is generated by running r.vi and r.vi.mpi GRASS module with increasing the operation numbers. To calculate different vegetation indices the workload in the slaves is too small to get benefit from r.vi.mpi module. Thus, the purpose of this experiment is to find out the sufficient amount of workload that will make the parallel version faster. In Fig. 6, to calculate the NDVI takes only 3 operations (subtraction, addition and then division) and this amount of workload takes just few seconds to execute, where the serial version takes less processing time then parallel version. However, increas-

ing the number of operations provide better performance in parallel version. It is clear from Fig. 6 that to get the benefit from the parallel version the total workload needs to be around 300 operations or more so that the communication overhead will be overpassed by the computational workload.

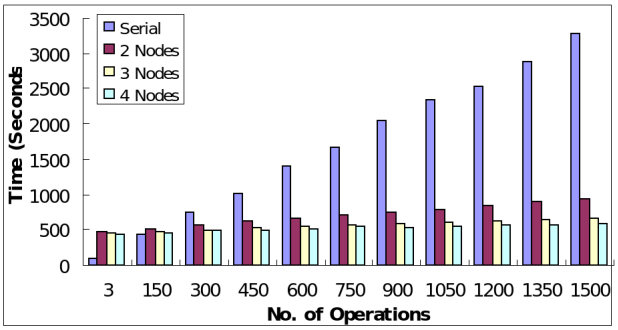

Figure 6: Performance Evaluation of Serial (r.vi) and MPI (r.vi.mpi) Version

When master with one worker node (slave) are working together to solve a particular problem, the master node will dispatch the whole job to his slave for processing. When there will be 2 slaves, the workload will be distributed among two salves. Similarly for 3 slaves, the workload will be distributed to 3 slaves equally. So the time performance increase for master with 2 slaves will be 2 times than master with one slave and the ratio will be 3 times for master with 3 slaves with one slave and so on... This would highlight a perfect parallelism.

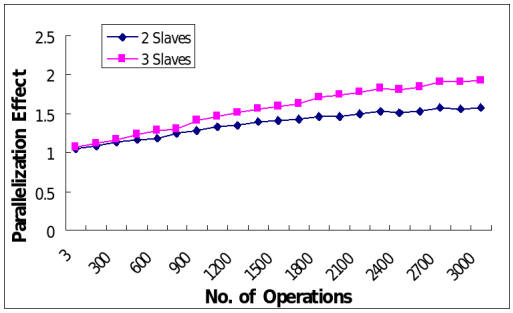

Figure 7: Parallelization Effect of r.vi.mpi Module with DTT

Concerning the above philosophy and experimenting on r.vi.mpi module, Fig. 7 and Fig. 9 have been generated. In Fig. 7 and Fig. 9, the *2 Slaves* curve presents the ratio of the total running time between master with two slaves and master with one slave. The *3 Slaves* curve represents the ratio of the total running time between master node with 3 slaves and master (one slave) with increasing the operation numbers. Increasing the operations, reflects to increase the workload in slaves and improving the performance. However, in Fig. 7, the performances are not satisfying with the desire values (as perfect parallelism) due to the regular and high communication overhead.

In equation 1, 2 and 3 the following terms have been used.

Term Name	Term Meaning
DTT	Data Transfer Time (Sec)
VDS	Volume of Data Send (each time)
VDR	Volume of Data Receive (each time)
NC	Number of Columns = 8519
NR	Number of Rows = 7630
NB	Number of Band Images = 6
DTS	Data Type Size = 8 Bytes
NBW	Network Band Width = 100Mb

Figure 8: Terms used and their definitions

$$DTT = \{NR \times (VDS + VDR)\}/(NBW) \quad (1)$$

$$VDS = NB \times DTS \times (NC + 1) \quad (2)$$

$$VDR = DTS \times (NC + 1) \quad (3)$$

From equation 2 and 3, equation 1 has been derived as

$$DTT1 = (6 \times 8 \times (8519 + 1)) + (8 \times (8519 + 1)) \quad (4)$$

$$DTT2 = 100 \times 1024 \times 1024 \quad (5)$$

$$DTT = (7630 \times 8 \times DTT1)/DTT2 = 277.74Sec \quad (6)$$

To evaluate the parallel version performance more precisely, Fig. 9 has been generated. Only the execution time (Data Transfer Time has been reduced from the total running time) is concerned here. Due to a constant amount of data need to transform between master and slaves, the transfer time (DTT) is derived from the above equations. Fig. 9 shows that the performance in the slaves curves are improving and nearly to the desirable values. It concludes that r.vi.mpi module is working well in distributed manner.

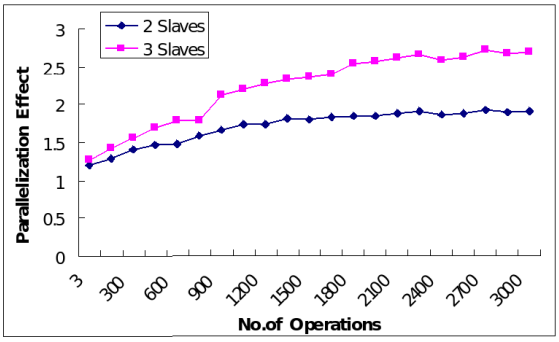

Figure 9: Parallelization Effect of r.vi.mpi module without DTT

Fig. 10 evaluates the performance of the GRASS modules (r.vi, r.vi.mpi, r.vi.grid) in the time domain. Low, middle and high, three testing workloads have been set in the modules. When the workload is low, serial version (r.vi) is performing best because of the fine-grain parallelism where communication time is bigger than the execution time. To get the coarse-grain, the workload need to be increased so that the communication time will be hidden by the execution time. As the workload increased to 1800 and 3000 operations, parallel versions are performing better then the serial version.

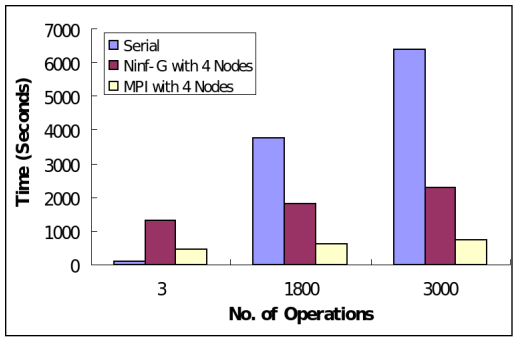

Figure 10: Performance Evaluation of GRASS Modules (r.vi, r.vi.mpi and r.vi.grid)

MPI version, r.vi.mpi provides the best performance among the three models. However, MPI is used mainly in Cluster Computers where the nodes are homogeneous in their specification. Additionally, available nodes in Clusters are limited up to a specific range and this will be an obstacle for load balancing for big processing jobs. In this case, Grid computing infrastructure is necessary. Indeed, (the environment maintains heterogeneity as well as distributed connecting networks). So far Ninf-G is performing better than the serial version for high work load examples.

Particularly, in this issue, Ninf-G is not performing better than MPI for the reason that, the communication overhead (to establish the session with remote hosts) in Ninf-G is larger than MPI. Ninf-G is specially made for Grid-Computing environment (not for running inside Cluster computing environment as have been done in above experiments). When the processing workload is much higher than the communication workload, the real performance improvement with Ninf-G platform will show up. So far, r.vi.grid is developed and tested. In near future, the experiment on real Grid testbed will be highlighted.

Conclusion

Geographic Resources Analysis Support System (GRASS) has been used for RS and GIS data analysis and visualization. Currently, GRASS handles large datasets and the performance and capabilities of GRASS for large datasets can be greatly improved by integrating GRASS with parallel and distributed computing. The major objective of this research was to provide the Remote Sensing user a compact example with Grid and MPI programming for GRASS GIS distributed processing. Additionally, this type of research will merge the Remote Sensing and GIS with High Performance Computing communities.

Acknowledgements

The authors would like to acknowledge specially, to Osawa Kiyoshi San (PhD Candidate in AIDA Lab), SunHao San (Masters Student in AIDA Lab) and Nishimora Motokazu San (Masters Student in AIDA Lab) for their support to create the testbed (Davinchi Cluster) ready for this experiment. Authors also like to thanks to all the members in AIDA Lab for the moral supports.

Bibliography

M. Snir, S. Otto, S. Huss-Lederman, D. Walker and J . Dongarra (1996) MPI: The Complete Reference. Massachusetts Institute of Technology. http://www.netlib.org/utk/papers/mpi-book/mpi-book.html.

I. Foster and C. Kesselman (1997) Globus: A Metacomputing Infrastructure Toolkit. International Journal of Supercomputer Applications.

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

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

B. Kamble, Y.H. Chemin (2006) GIPE in GRASS Raster Add-ons. http://grass.gdf-hannover.de/wiki/, GRASSAddOns, RasterAdd-ons Internet.

MPI(2007) http://www-unix.mcs.anl.gov/mpi/ Internet.

Ninf-G(2007) http://ninf.apgrid.org/ Internet.

J.Weier and D.Herring. (2007) Measuring Vegetation (NDVI/EVI) http://earthobservatory.nasa.gov/Library/MeasuringVegetation/ Internet.

SUN Grid Engine(2007) http://www.lesc.ic.ac.uk/projects/epic-gt-sge.html Internet.

Shamim Akhter, Kento Aida
Tokyo Institute of Technology
http://www.alab.ip.titech.ac.jp/~shamim
shamimakhter@gmail.com *Yann Chemin*
yann.chemin@gmail.com