Journal de l'OSGeo

Le Journal de l'Open Source Geospatial Foundation Volume 2 / Septembre 2007

Dans ce volume

Les bases de la topologie

1Spatial : ***Concepts de qualité des données***

Introduction à MapWindow & GeoNetwork

LizardTech : ***Pourquoi utilise-t-on des logiciels Open Source***

Rapport des Local Chapter : Taiwan, U.K., Francophone, Espagnol ...

Étude de cas : UN FAO, Traçage des navires de pêche ...

Rapport d'événements communautaires : Inde, France

Traitement distribué et GRASS

Actualités et mise à jour des logiciels ...

Table des matières

Cours de programmation **2**
Portage d'un module raster de GRASS pour le calcul distribué 2

Cours de programmation

Portage d'un module raster de GRASS pour le calcul distribué

Exemples avec MPI et Ninf-G

Shamim Akhter, Yann Chemin, Kento Aida, traduit par Yann Chemin

Résumé

L'imagerie satellitaire procure un grand nombre d'informations utiles. Extraire ces informations et les comprendre peut nécessiter une capacité et un temps de calcul très importants. Le calcul distribué peut réduire le temps de traitement en fournissant plus de puissance de calcul. GRASS, un logiciel libre, a été utilisé pour le traitement d'images satellitaires. Pour montrer comment les modules de GRASS peuvent tirer parti du calcul distribué, un module (r.vi) est ici porté, à titre dâĂŹexemple, à lâĂŹaide des composants de programmation distribuée MPI (r.vi.mpi) et Ninf-G (r.vi.grid). Les faĂğons d'implanter ces modules constituent le sujet principal de cet article qui va progressivement présenter les étapes de base requises pour porter tout module raster de GRASS sur une plate-forme distribuée. Une étude comparative des versions modifiées de r.vi, r.vi, r.vi.mpi et r.vi.grid est également présentée.

Introduction

Le traitement de l'imagerie satellitaire joue un rôle vital pour les développements de la recherche en télédétection, en SIG, pour le suivi de l'agriculture, la gestion des désastres et pour d'autres sujets d'étude. Néanmoins, traiter ces images satellites de résolution spatiale toujours croissante nécessite un temps de calcul important en raison de la complexité et de la taille des traitements. Cela semble être une barrière pour la prise de décision en temps réel. Le traitement distribué de données peut être une solution réaliste pour réaliser une telle tâche dans les temps. Les clusters et (ou grappes de machines) et les environnements de type grid (ou grille de calcul) sont deux systèmes distribués bien connus qui ont été associés avec le calcul de haute performance pour des applications ayant des besoins très élevés en termes de CPU. GRASS GIS (Neteler et al. , 2003) est un logiciel libre qui a été utilisé pour traiter des images satellites. Dans GRASS, différents modules ont été développés pour traiter des images satellites. Le module r.vi de GRASS développé par (Kamble et al. , 2006) est utilisé dans cette étude comme un exemple de test. Le développement de la méthode, qui permet d'utiliser l'environnement GRASS GIS pour le traitement d'image satellite sur des systèmes de calcul

distribué est le sujet principal de cet article. Deux différentes méthodes d'implémentation d'un r.vi distribué sont présentées ici pour les plate-formes de programmation MPI (**?**) et Ninf-G (**?**).

L'Index de Végétation (VI) constitue lâĂŹensemble principal d'indicateurs pour la végétation. Le module r.vi de GRASS, est utilisé pour calculer 13 indices différents de végétation pour les images satellite. NDVI (Normalized Difference Vegetation Index (Weier et al. , 2007)) est un d'entre eux. Le NDVI est calculé à partir de ces mesures individuelles : NDVI=(IRP-Rouge)/(IRP+Rouge), où Rouge et IRP sont respectivement les mesures de réflectance spectrales acquises dans les régions du rouge et de l'infrarouge proche. D'autres indices de végétation sont :

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

FIG. 1 – Indices de Végétation

Ils sont dérivés en utilisant différentes méthodes de différenciation et de contraste. La figure 2 montre une capture d'écran d'un résultat de calcul produit par GRASS.

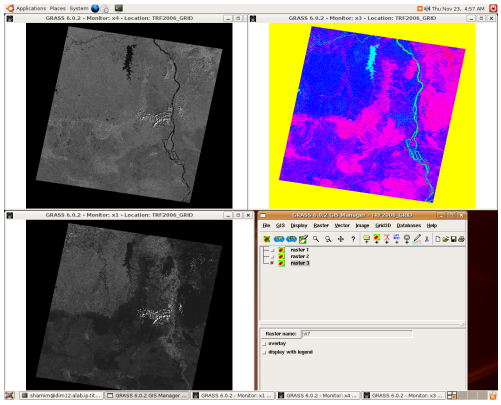

FIG. 2 – Calcul de NDVI avec GRASS

Le module r.vi de GRASS utilise des images de type raster (lignes x colonnes). Plusieurs bandes d'images raster sont nécessaires pour différents indices. Les indices génériques (NDVI, RVI, etc) utilisent les bandes d'images du rouge et de l'IRP (Infrarouge Proche). ARVI utilise quant Ãă lui le rouge, l'IRP et le bleu ; GVI utilise le rouge, l'IRP, le bleu, le vert, la bande 5 et la bande 7 des images Landsat. GARI utilise le rouge, l'IRP, le bleu et le vert. Les fonctions de GRASS sont utilisées pour extraire les lignes de données provenant des bandes spectrales pertinentes et pour les stocker dans des zones de mémoire tampon. Après cela, chaque valeur de colonne est extraite séquentiellement des zones de mémoire tampon et envoyée pour générer les valeurs de VI spécifiques. Ainsi, après avoir traité les valeurs de VI des lignes en mémoire-tampon, les valeurs de VI par ligne sont écrites dans le fichier image résultat. Cette procédure se répètera pour chaque ligne. La figure 3 présente la structure du module r.vi s'exécutant séquentiellement (par soucis de simplicité, seulement deux bandes spectrales sont présentées).

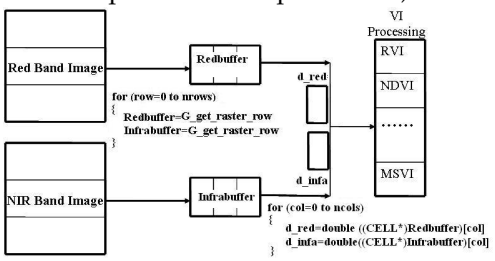

FIG. 3 – Structure du Module r.vi s'exécutant séquentiellement

Objectifs

Dans l'hypothèse où le calcul distribué éliminera les contraintes de temps de calcul, de nouveaux algorithmes de traitement d'image appliqués aux données provenant de la télédétection peuvent être considérés. Divers modules de GRASS ont été développés pour résoudre différents problèmes d'analyse d'images acquises par télédétection. L'objectif principal de cet article est d'évaluer la performance des modules de GRASS dans un environnement de calcul distribué. De plus, le portage dans un environnement distribué des processus créés par les utilisateurs de données de télédétection est une activité nouvelle pour ces derniers. Un autre point abordé dans cet article, porte sur l'évaluation de la charge de travail nécessaire pour que les performances obtenues en répartissant le travail entre différents noeuds de calcul soient meilleures que lors de l'utilisation du

module séquentiel.

Méthodologie

Pour répondre aux objectifs ou aux besoins mentionnés précédemment, le module de GRASS (r.vi) a été parallélisé en utilisant une stratégie maître-esclave. Le processus maître est actif dans l'environnement de GRASS et décompose les images cibles en lignes et les distribue aux multiples processus esclaves. Ces derniers sont indépendants de GRASS, ils effectuent les calculs et renvoient le résultat pour chaque ligne au processus maître. Le module r.vi est implémenté en utilisant MPI sur un système de PCs en cluster (r.vi.mpi) et Ninf-G sur le même système (r.vi.grid) afin de préserver la similarité des environnements expérimentaux. Néanmoins, le module r.vi.grid a une structure capable de travailler sur un système distribué de type GRID (grille de calcul). Des expériences ont enfin été menées pour analyser les résultats des modules r.vi distribués en augmentant le nombre d'opérations, afin de trouver la charge de travail nécessaire pour tirer parti des environnements de calcul distribué.

Dans la figure 4, la structure mettant en place le module r.vi distribué est présentée (par soucis de simplicité, seulement deux bandes spectrales ont été représentées). Ici, S1, S2, ... et Sn sont différents processus esclaves.

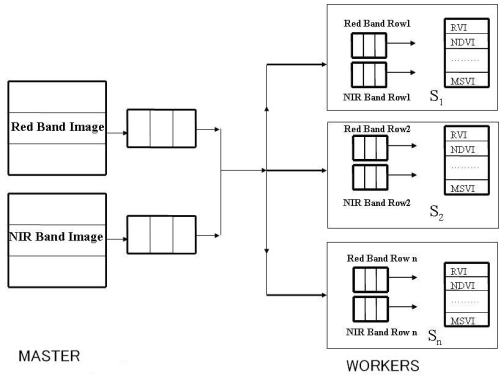

FIG. 4 – Structure du module r.vi Distribuée (r.vi.mpi et r.vi.grid)

Implémentation

MPI et le framework Ninf-G

MPI (Interface de Passage de Message) est une bibliothèque de fonctions (en C) ou de sous-routines (en FORTRAN) que l'on peut insérer dans un fichier source de code informatique pour permettre la communication de données entre processeurs (Sain et al. , 1996). MPI fut développé Ãă des fins de calcul haute performance à la fois sur machines massivement parallèles mais sur des grappes de stations de travail (cluster). MPI fut mis en oeuvre par un comité relativement imposant et regroupant des vendeurs de matériels et de logiciels, des professionnels ainsi que des utilisateurs.

Ninf-G a été développé par AIST (National Institute of Advanced Industrial Science and Technology, Japan) et TITECH (Tokyo Institute of Technology, Japan). Ninf-G est une ré-implémentation du système Ninf au-dessus de la boîte à outils Globus (Foster et al. , 1997). Globus sert de plate-forme robuste et commune pour la mise en place d'outils middleware et de programmation de plus haut niveau, etc., assurant ainsi l'interopérabilité entre de tels éléments de haut niveau, dont Ninf-G. Le système Ninf-G est basé sur une architecture client-serveur. Les ressources de calcul sont disponibles à travers des bibliothèques hébergées sur un hôte de calcul qui peut être appelé au travers du réseau global à partir de programmes clients écrits dans un langage courant comme FORTRAN, C, ou C++.

Détails du Cluster Davinchi

Les noeuds du Cluster utilisés pour les expérimentations sont :

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)

FIG. 5 – Détails du Cluster Davinchi

Module r.vi distribué en MPI (r.vi.mpi)

Pour le module r.vi.mpi, le code source de GRASS est installé dans le noeud qui sera maître et les bibliothèques GRASS sont copiées sur les noeuds esclaves en mémoire locale au même endroit que dans le noeud maître (/usr/local/grass-6.0.2/) et les étapes de configuration suivantes sont nécessaires pour l'environnement de MPI-GRASS.

- le fichier grass.conf est créé à l'intérieur du répertoire /etc/ld.so.conf.d/ et les noms des bibliothèques de GRASS sont écrits dans ce fichier.

- Après cela la commande /sbin/ld config est exécutée.

Les fonctions MPI_Send et MPI_Recv ont été utilisées pour la communication des données.

Le fichier Makefile suivant est créé pour la compilation du code informatique :

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

Pour exécuter le code informatique, la commande shell suivante a été utilisée :

```
mpirun -np 3 location_du_fichier_execution parametres
```

Dans cette experience, la localisation du fichier d'exécution est : $GRASS_COMPILE_DIR$/dist.i686-pc-linux-gnu/bin/r.vi.mpi Le pseudo-code informatique de r.vi.mpi est ci-après :

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


/* main.c: Declare le code MPI suivant   */
/* NUM_HOSTS est le nombre total d'hotes */
/* me est le rang/nombre de CPUs         */

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

/*---------------------------------------*/
/* Commence Code Commandant: Rang 0 (me=0)*/


/* Extrait nombre total de lignes et de  */
/* colonnes et envoie les aux executants */

for(i=1;i<NUM_HOSTS;i++)
/* Commence boucle en i :

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

/* Boucle i est finit.

/* Donnees des lignes sont extraites des images */
/* et distribuees parmis les executant avec */
/* nombres de lignes specifiques en boucle */
/* d'attribution */

for (r = 1; r*(NUM_HOSTS-1) <= nrows;r++ )
/* D\'ebut de boucle en r:

for(k=1;k<NUM_HOSTS;k++)
/* D\'ebut de boucle en k:

  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++)
  /*D\'ebut de boucle en col:

/* Chaque valeur de cellule de colonne venant de
/* toute les bandes sont extraites et mises dans
/* une rangee de memoire 2D
    db[0][col]= d_redchan;
    db[1][col]= d_nirchan;

  /* Boucle en col finit.

  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);

/* Boucle en k finit.

/* Attend les resultats...

for(k=1;k<NUM_HOSTS;k++)
/* Debut de boucle en k :

  MPI_Recv(R,ncols+1,MPI_DOUBLE,k,1,\
  MPI_COMM_WORLD,&status);
  row_n=R[ncols];
  for (cn=0;cn<ncols;cn++)
  /* D\'ebut de boucle en cn:

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

  /* Fin de boucle en cn.

/* Fin de boucle en k.

/* Les lignes traitees sont remises dans */
/* les images resultats */
for(k=0;k<(NUM_HOSTS-1);k++)
/* Debut de boucle en k :

 for(j=0;j<ncols;j++)
 /* Debut de boucle en j :

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

 /* Fin de boucle en j.

/* Fin de boucle en k.
```

```
/* Fin de boucle en r.

/* Si il y a quelconque lignes en plus */
/* (row_number%slaves!=0), le reste de */
/* l'attribution des lignes (moins que n) */
/* sont distribuees d'executant 1 a n une */
/* autre fois */

MPI_Finalize();
G_free(inrast_redchan);

/* Fin du code Commandant                    */
/*-----------------------------------------

/*-----------------------------------------
/* Debut de code executant: Rang non 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++;

/* Reception de donnees du commandant et traitement
for(i=0;i<n_rows;i++)
/* Debut de boucle en i :

  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++)
  /* Debut de boucle en col :

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

  /* Fin de boule en col.

  r[ncols]=I[ncols];

   /* resultats sont dans r[] et
   /* renvoit au Commandant
  MPI_Send(r,ncols+1,MPI_DOUBLE,0,1,\
  MPI_COMM_WORLD);

/* Fin de boucle en i.

 MPI_Finalize();

/* Fin de code ex\'ecutant.                  */
/*-----------------------------------------
```

Module r.vi distribué en GRID (r.vi.grid)

r.vi est plus aisé à transformer en environnement Ninf-G que MPI. Ici, il n'y a nul besoin de copier les fichiers des bibliothèques de GRASS, car les procédures d'exécution sont totalement indépendantes de la procédure maîtresse et de l'environnement GRASS. L'API d'appel GridRPC (Tanaka et al. , 2003) est utilisée pour communiquer entre le maître et les exécutants. Le fichier Makefile suivant est créé pour la compilation du code informatique :

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

Comme tout autre module GRASS, r.vi.grid peut démarrer à partir de la commande

```
r.vi.grid  param\`etres
```

Le fichier IDL d'un exécutant est comme suit :

```
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);
```

Le code informatique d'un porcessus exécutant est comme suit :

```
#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++)
 //Debut de boucle col

    /*Valeur par colonne a envoyer pour traiter par
    /* index specifique, mettant le resultat en r[]
/* traitement du ndvi....
r[col]=(a[col]-b[col])/(a[col]+b[col]);

 //Fin de boucle col.

} //function call finish
```

Le code informatique du module maître est comme suit :

```
#include "gis.h"
#include "glocale.h"
#include "grpc.h" // inclure entete grpc.h
#define NUM_HOSTS 5 //Combien d'hotes disponibles

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);
}


//creation d'une poign\'ee pour chaque h\^ote
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++)
/* Debut de boucle en row :


  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++)
  /* D\'ebut de boucle en col:


     /* Chaque valeur de cellule de colonne pour
     /* chaque bande d'image est extraite d'une
     /* fois et mises dans des rangees memoires 2D
    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;
     /* Les valeurs d'index de 'egetation sont
     /* remplies en rangees de memoires I[ncols]


  /* Fin de boucle en col.


  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);
  }


   /* Tous les sortants sont remis dans des raster
  for(j=0;j<ncols;j++)
  /* Debut de boucle j :


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


  /* Fin de boucle j.


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


  host_n++;


/* Fin de boucle en row.


 /*Destruction des poignees
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...(Libere la memoire)
```

Résultats Expérimentaux

La figure 6 provient de l'exécution des modules r.vi et r.vi.mpi de GRASS en utilisant un nombre d'opérations croissant. Pour calculer différents indices de végétation la charge de travail au sein des esclaves est trop petite pour tirer parti du module r.vi.mpi. Donc, le but de cette expérimentation est de déterminer la charge de travail nécessaire pour accroître les performances d'une version parallèle. Dans la figure 6, pour calculer le NDVI seulement

3 opérations sont nécessaires (soustraction, addition et division) ce qui prend seulement quelques secondes à exécuter, la version sur CPU unique nécessite au final moins de temps de calcul que la version parallèle. Néanmoins, une augmentation du nombre d'opérations permet une meilleure performance de la version parallèle. Il est clair dans la figure 6 que pour avoir un bénéfice en version parallèle la charge totale de calcul doit être d'au moins 300 opérations pour que le surcoût de communication soit dépassé par la charge de calcul elle-même.

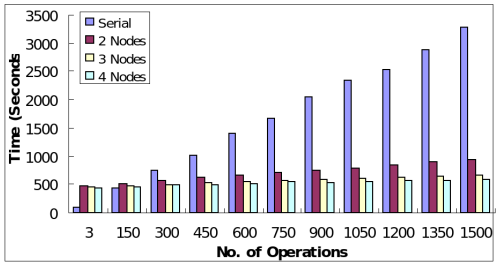

FIG. 6 – Évaluation de la performance des versions séquentielles (r.vi) et MPI (r.vi.mpi)

Quand le maître et un noeud travaillent ensemble pour résoudre un problème particulier, le-noeud maître va allouer tout le travail à ce seul noeud esclave pout traitement. S'il y a deux noeuds esclaves, la charge de travail sera distribuée entre ces deux noeuds. De même pour 3 exécutants, la charge de travail sera distribuée en 3 parts égales. Donc la performance en temps augmente : pour un maître avec 2 noeuds esclaves le temps sera 2 fois moindre qu'avec un seul noeud et le ratio sera de 3 pour un maître ayant 3 noeuds esclaves au lieu d'un seul, etc... Cela caractériserait un parallélisme parfait.

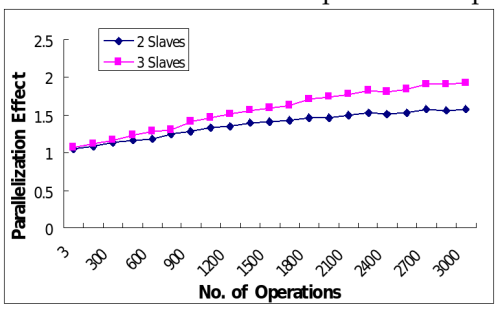

FIG. 7 – Effet de la parallélisation du module r.vi.mpi avec Temps de Transferts de Données (DTT)

En adoptant la démarche précisée ci-avant et en s'appuyant sur le module r.vi.mpi, les figures 7 et 9 ont été produites. Dans celles-ci, les deux courbes des représentent le ratio du temps total d'exécution avec 2 et 3 noeuds esclaves sur le temps total d'exécution avec un seul noeud ceci pour un nombr croissant d'opérations. L'augmentation des opérations reflète l'accroissement de la charge de travail des noeuds esclaves et une meilleure performance. Pourtant, dans la figure 7, les performances ne sont pas si satisfaisantes avec les valeurs désirées (c'est à dire, pas aussi satisfaisante qu'un parallélisme parfait) à cause du surcoût régulier et élevé de communication entre les noeuds.

Dans les équations 1, 2 et 3, les termes suivants ont été utilisés :

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

FIG. 8 – Termes utilisés et leurs définitions

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

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

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

En s'appuyant sur les équations 2 et 3, l'équation 1 est dérivée en :

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

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

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

Pour évaluer précisÃl'ment la performance de la version parallèle, la figure 9 a été générée. Seul le temps d'exécution est adressé ici, car le Temps de Transfert des Données (DTT) a été soustrait du temps total d"exécution. Due à la quantité constante de donnÃl'es qu'il est nécessaire de transformer entre le maître et les esclaves, le DTT est dérivé des équations ci-dessus. La figure 9 montre que la performance des noeuds esclaves s'améliorent et atteint presque les valeurs souhaitées. Cela conclut que le module

r.vi.mpi remplit bien sa fonction de distribution des tâches de calcul.

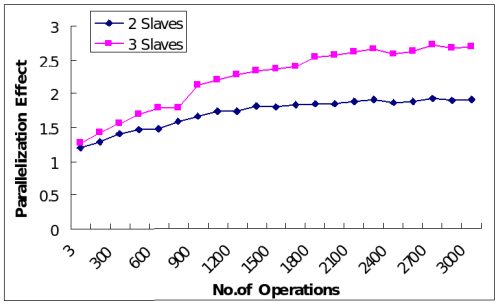

FIG. 9 – Effet de parallèlisation du module r.vi.mpi sans DTT

La figure 10 rend compte de la performance des modules r.vi, r.vi.mpi, r.vi.grid de GRASS en terme de temps de calcul. Trois charges de travail test (faible, moyenne et haute) ont été assignées à chacun de ces modules. Quand la charge de travail est faible, la version sur CPU unique (r.vi) est la meilleure à cause du faible parallèlisme pour lequel le temps de communication est plus grand que le temps d'exécution. Pour atteindre un parallélisme plus important, la charge de travail doit être augmentée afin que le temps de communication devienne mineur vis à vis du temps d'exécution. Plus la charge de travail augmente de 1800 vers 3000 opérations, plus les versions parallèles sont performantes par rapport à la version simple séquentielle.

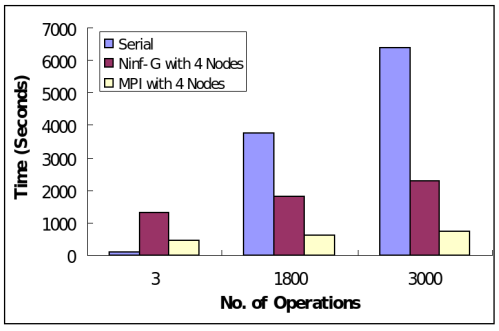

FIG. 10 – Évaluation de la performance des modules GRASS : r.vi, r.vi.mpi et r.vi.grid

La version MPI, r.vi.mpi produit les meilleures performances parmi les trois modèles. Néanmoins, MPI est utilisé principalement dans les systèmes de type cluster où les noeuds ont des spécifications techniques homogènes. De plus, les noeuds de calcul disponibles dans un cluster sont limités et cela constitue un obstacle dans la répartition des charges de travail lors de traitements lourds. Dans ce cas précis, une infrastructure de type "Grid computing" (grille de calcul) peut s'avérer nécessaire. En effet, l'environnement gère l'hétérogénéité de même que la distribution des réseaux connectés. Jusqu'à présent, Ninf-G est plus performant que la version à s'équentielle (c'est à dire, à CPU unique) pour les exemples de charges de travail importante.

Dans cette expérience, Ninf-G n'est pas plus performant que MPI à cause du surcoût de communication (pour établir la session avec les hôtes distants) qui avec Ninf-G est plus important qu'avec MPI. Ninf-G est spécifiquement conçu pour les environnements de calcul de type Grid (grille de calcul), et non pas pour être utilisé dans un cluster comme cela a été fait dans les expérimentations illustrées précédemment. Quand la charge de calcul est plus grande que celle de communication, l'amélioration réelle de performance avec Ninf-G se fait sentir. Pour l'instant, r.vi.grid est développé et testé. Dans un futur proche, un banc d'essai dans un véritable environnement de type grid (grille de calcul) dans lequel l'expérience sera reproduite sera réalisé afin de mettre en exergue les apports d'une telle infrastructure.

Conclusion

GRASS (Geographic Resources Analysis Support System) est utilisé pour des tÃćches dâĂŹanalyse et de visualisation de données SIG et de télédétection. Aujourd'hui, GRASS est capable de manipuler de larges jeux de données. La performance et les capacités de GRASS à traiter de grands jeux de données peuvent être améliorées de beaucoup en intégrant GRASS dans des environnements de calcul parallèle et distribué. Le principal objectif de cette recherche était de donner à un utilisateur de données de télédétection un exemple compacte de programmation à l'aide de Ninf-G et MPI, afin de réaliser des traitements avec GRASS GIS de manière distribuée. Ce type de recherche devrait de surcroît, contribuer à rapprocher les communautés de la télédétection et des SIG avec celle du calcul en haute performance (HPC).

Remerciements

Les auteurs voudraient remercier spécialement Osawa Kiyoshi (doctorant au Laboratoire AIDA), SunHao et Nishimura Motokazu (étudiants en Masters au laboratoire AIDA) pour leur support dans la création de banc d'essai (Davinchi Cluster) ayant permis de mener cette expérience. Les auteurs voudraient aussi remercier tous les membres du labora-

toire AIDA pour leur soutien moral.

Bibliographie

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
Tokyo Institute of Technology
http://www.alab.ip.titech.ac.jp/~shamim
shamimakhter@gmail.com

Remerciements
Divers relecteurs & le projet actualités de GRASS

Le *Journal de l'OSGeo* est une publication de *la Fondation OSGeo*. La base de ce journal, le source du style LaTeX 2ε a été généreusement fournie par l'équipe éditoriale de l'actualité de GRASS et R.

ISSN 1994-1897