SmartGridSolve Manual

1. SmartGridSolve Introduction

SmartGridSolve is an extension of GridSolve which is aimed at higher performance of Grid applications by providing the functionality for collective mapping of a group of tasks on to a network that is fully connected. The SmartGridSolve API allows the user to separate the mapping of tasks from their execution which is one atomic operation in the GridRPC model of GridSolve. This allows a group of tasks to be mapped collectively which can improve the performance of the group by

- more effectively balancing the load of computation of the group of tasks
- more effectively balancing the load of communication of the group of tasks
- reducing the overall volume of communication of the group by eliminating bridge communication either by caching or direct data transfers between servers

In addition the traditional client-server model of GridRPC has been extended so that the group of tasks can be collectively mapped on to a network topology that is fully connected. This is a network topology where all the servers can communicate directly or servers can cache their outputs locally.

The mapping of a group of tasks on a fully connected network not only involves the mapping of tasks to servers but also the mapping of virtual links between tasks (i.e. links representing data dependencies) on to the communication paths of the network. This increases the mapping solution space and allows for further optimization to be achieved by choosing the optimal paths between servers.

2. Using SmartGridSolve

To use SmartGridSolve, one should enable the SmartGridSolve feature both on the GridSolve client and server. Type

```
%  ./configure --enable-smartgridsolve
% make
% make services
```

during the initial configuration of GridSolve. Note that services/tasks require compilation when GridSolve system has been configured with SmartGridsolve extension enabled.

3. SmartGridSolve API

The SmartGridSolve API allows a user to specify the scope of a group of tasks to be mapped collectively.

3.1 gs_smart_map

This function is used for specifying the scope of the group of tasks and the mapping heuristic to implement

```
gs_smart_map(char * mapping_heuristic_name)
```

Parameters :

- mapping_heuristic_name – Name of the mapping heuristic to implement when mapping the group of tasks.

Usage :

```
gs_smart_map(char * mapping_heuristic_name){
  ...
  // group of tasks to map collectively
  ...

}
```

Description :

The gs_smart_map "function" is in fact a macro that inserts a while loop around the code block specified by the parenthesis. When the gs_smart_map function is called the code within its parenthesis will be iterated through twice. On the first iteration each grpc_call and grpc_call_async is discovered but not executed. From these discovered calls a task graph is generated. At the beginning of the second iteration the mapping heuristic specified by the gs_smart_map parameter will generate a mapping solution based on the task graph and the performance model of the network. The mapping solution outlines a task to server mapping and also the communication scheme between tasks.
The communication scheme may implement

- client server communication
    - standard GridRPC communication
- server-server communication
    - server sends a single argument to another server
- client broadcasting
    - client sends a single argument to multiple servers.
- server broadcasting
    - server sends a single argument to multiple servers.
- server caching
    - server stores an argument locally for future tasks.

During the second iteration through the code, the tasks will be executed according to the generated mapping solution.
It should also be noted that handles and sessionids should be created, initialised, destroyed and deleted outside the scope of the parenthesis of the gs_smart_map function.

Example:

In this example the gs_smart_map function is the only addition required to make this code SmartGridSolve enabled. As previously explained the handles and sessionids should be created, initialised, destroyed and deleted outside the scope of the parenthesis of the gs_smart_map function.

```
grpc_function_handle_t*handles=(grpc_function_handle_t*)
                  calloc(iters, sizeof(grpc_function_handle_t ));

grpc_sessionid_t * sessionIDs=( grpc_sessionid_t *)
                   calloc(iters, sizeof(grpc_sessionid_t));

int * status=(int *)
                  calloc(iters, sizeof(int));

for(i=0; i<iters; i++){
  if(grpc_function_handle_default(&handle[i], "dgesv") != GRPC_NO_ERROR) {
    fprintf(stderr,"Error creating function handle1\n");
    die(EXIT_FAILURE);
  }
}

.....
.....

gs_smart_map("ex_map"){
  for(i=0;i<iters;i++){
    status1=grpc_call_async(&handle[i], &sessionID[i], .., .., ..);
  }
  for(i=0;i<iters;i++){
    grpc_wait(sessionID[i]);
  }
}
.....
.....
for(i=0;i<iters;i++)
  if(grpc_function_handle_destruct(handle[i]) != GRPC_NO_ERROR) {
    fprintf(stderr,"Error destroying function handle1\n");
    die(EXIT_FAILURE);
  }
}
```

3.2 gs_smart_map_ft

This function is a fault tolerant version of the gs_smart_map function

```
gs_smart_map_ft(char * mapping_heuristic_name)
```

Parameters :

- mapping_heuristic_name – Name of the mapping heuristic to implement when mapping the group of tasks.

Usage :

```
gs_smart_map_ft(char * mapping_heuristic_name){
  ...
  // group of tasks to map collectively
  ...

}
```

Example:
Implementation is the same as gs_smart_map, just change the function from gs_smart_map to gs_smart_map_ft

Description :

This is the same as gs_smart_map function, except that the mapping solution generated does not implement server-server communication. The mapping solution outlines a task to server mapping and a communication scheme which only implements communication between client and server.
The communication scheme may implement

- client-server communication
    - standard GridRPC communication
- client broadcasting
    - client sends a single argument to multiple servers.

If any server that is part of the mapping solution fails, then the tasks mapped to those servers will be mapped to the next server which is estimated to give lowest execution time for that task.

3.3 gs_smart_local_region

This function is used to specify the code that should be ignored during on the first iteration through the scope of gs_smart_map (i.e. code that should be ignored when building the task graph).

```
int gs_smart_local_region()
```

Usage:

```
gs_smart_map(char * mapping_heuristic_name){

   //reset variables which have been updated
   //task discovery during

  if(gs_smart_local_region()){
    //code to ignore when generating task graph
  }
  ...
  // group of tasks to map collectively
  ...

}
```

Description :

Any segment of client code that is not part of the GridRPC API should not be executed during task discovery. To achieve this, such code must be enclosed in the conditional that tests the gs_smart_local_region function. This function will return false during task discovery and true during execution.

There is one exception to this rule, when the client code directly affects any aspect of the task graph. For example, if a variable is updated on the client that determines which remote tasks get executed or the size of inputs/outputs of any task, then the operations on this variable should not be encapsulated by gs_smart_local_region. If any variables or structures are updated during the task discovery cycle then they should be restored to their original values before the execution cycle begins.

Example:
In this example the variable x determines which tasks get executed and therefore any computation on x should not be encapsulated by the gs_smart_local_region. However the variable y does not affect the task graph therefore computations on y should be encapsulated by the gs_smart_local_region.

```
grpc_function_handle_t*handles=(grpc_function_handle_t*)
                  calloc(iters, sizeof(grpc_function_handle_t ));

grpc_sessionid_t * sessionIDs=( grpc_sessionid_t *)
                   calloc(iters, sizeof(grpc_sessionid_t));

int * status=(int *)
                  calloc(iters, sizeof(int));

for(i=0; i<iters; i++){
  if(grpc_function_handle_default(&handle[i], "dgesv") != GRPC_NO_ERROR) {
    fprintf(stderr,"Error creating function handle1\n");
    die(EXIT_FAILURE);
  }
}

.....
.....

x_old=x;
gs_smart_map("ex_map"){
  x=x_old;
  for(i=0;i<iters;i++){
    x=func1(x);

    if(x==1){
      status1=grpc_call_async(&handle[i], &sessionID[i], .., .., ..);
    }
    if(gs_smart_local_region()){
      y=func2();
    }
  }

  for(i=0;i<iters;i++){
    grpc_wait(sessionID[i]);

  }

}
.....
.....
for(i=0;i<iters;i++)
  if(grpc_function_handle_destruct(handle[i]) != GRPC_NO_ERROR) {
    fprintf(stderr,"Error destroying function handle1\n");
    die(EXIT_FAILURE);
  }
}
```

4. Application of SmartGridSolve

In this section we present implementations of the Hydropad application in SmartGridSolve. This application can be downloaded on the Heterogeneous Computing Laboratory website http://hcl.ucd.ie/project/SmartGridSolve .

Hydropad is an astrophysical application that simulates the evolution of clusters of galaxies in the universe. Hydropad is a cosmological application, originally written by Claudio Gheller, which simulates the evolution of clusters of galaxies in the universe. The cosmological model that this application is based on has the assumption that the universe is composed of two different kinds of matter. The first is baryonic matter, which is directly observed and forms all bright objects. The second is dark matter, which is theorised to account for most of the gravitational mass in the Universe. The evolution of this system can be simulated by examining the mutual interaction between these components which is regulated by a gravitational component. Figure1 shows an example of a typical output generated by Hydropad

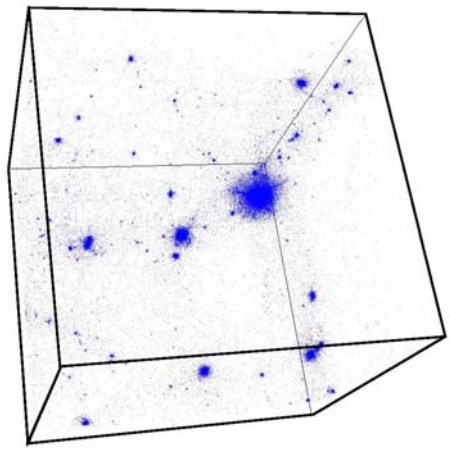

Figure 1: Example of Hydropad Output

Figure 2 shows the work-flow of the Hydropad application. It is composed of two parts: the initialisation part which initialises the initial state of the universe and the evolution part. The evolution part of the application consists of a number of iterations that simulate the discrete time steps used to represent the evolution of the universe from the Big Bang to present time. This part consists of three tasks: the gravitational task the dark matter task and the baryonic matter task. For every time step in the evolution of the universe, the gravitational task generates the gravitational field using the density of the two matters calculated in the previous time step. Hence the dark and baryonic tasks use the newly produced gravitational forces to calculate the movement of the matter that happens during this time step. Then the new density is generated and the lapse of time in the next time step is calculated from it. It is possible to see in figure 2 that the dark matter task and baryonic matter task are independent of each other.

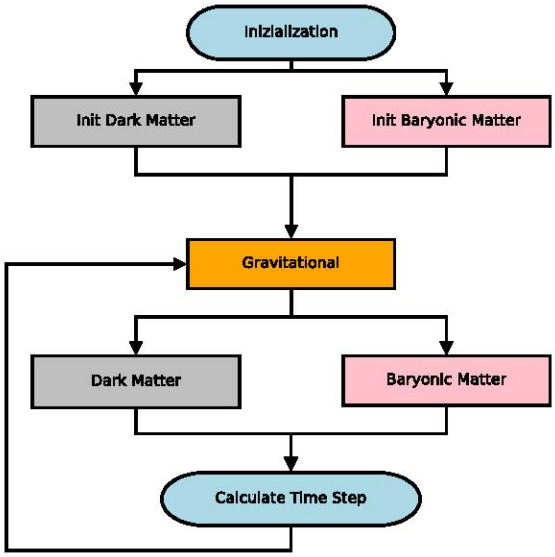

Figure 2: Internal structure of Hydropad application

The following sections demonstrate how the GridRPC implementations of this application were extended to be SmartGridSolve enabled.

4.1 Example 1: Hydropad Initialisation

This section describes how to extend the GridRPC implementation of the **initialisation** part of the Hydropad application to be SmartGridSolve enabled.

The initialisation part of Hydropad consists of four tasks, the initgrav task, the usegrafic, the densitydm task and the initbm. The initgrav is executed in parallel with the other three tasks. In the GridRPC implementation, the tasks will be individually mapped to servers. And the execution involves the client sending inputs of each task to the assigned server and receiving the outputs from the assigned server. **Therefore the GridRPC implementation supports the minimisation of the execution time of each individual task.**

In the SmartGridSolve implementation, the group of tasks will be mapped collectively. There only addition to the GridRPC implementation is the gs_smart_map function. When the gs_smart_map function is called the group of tasks is iterated through twice on the first iteration the group is discovered. At the beginning of the second iteration the mapping heuristic generates a mapping solution for the group. In this case the exhaustive mapping heuristic generates the solution. This solution outlines a task to server mapping and communication scheme that will minimize the execution time of the group of tasks. The mapping heuristic will balance the load of computation and communication of the group and also reduce the overall communication by removing bridge communication. On the second iteration through the group each task will be executed according to the mapping solution of the group.

Therefore the SmartGridSolve implementation supports the minimization of the execution time of the group of tasks.

The execution in the SmartGridSolve implementation may involve

- client server communication
  - standard GridRPC communication
- server-server communication
  - server sends a single argument to another server
- client broadcasting
  - client sends a single argument to multiple servers.
- server broadcasting
  - server sends a single argument to multiple servers.
- server caching
  - server stores an argument locally for future tasks.

GridRPC Implementation

```
grpc_call_async(initgrav_hndl, &sid_initgrav, ...);
grpc_call(usegrafic_hndle, ...);
grpc_call(densitydm_hndl, ...);
grpc_call(initbm_hndl, ...);
grpc_wait(sid_initgrav);
```

SmartGridSolve Implementation

```
gs_smart_map("ex_map"){
  grpc_call_async(initgrav_hndl, &sid_initgrav, ...);
  grpc_call(usegrafic_hndle, ...);
  grpc_call(densitydm_hndl, ...);
  grpc_call(initbm_hndl, ...);
  grpc_wait(sid_initgrav);
}
```

4.2 Example 2 : Hydropad Evolution

This section describes how to extend the GridRPC implementation of the **evolution** part of the Hydropad application to be SmartGridSolve enabled. The evolution part of Hydropad consists of four tasks, the gravitational task, the initialise velocity task, the dark matter task and the baryonic matter task. The initialise velocity task is only executed on the first evolution cycle. On each evolution step, the dark matter and the baryonic matter are both executed in parallel. Also, in each evolution step, the following variables get updated on the client ga, gb->nsteps, gb->bmvelmax, gb->dmvelmax.

When mapping a group of task in SmartGridSolve the variables that are updated on the client need to be assessed as to whether they affect any aspect of the task graph. If a variable follows any of the following conditions then it should not be encapsulated by the gs_smart_map function

- The variable affects which remote tasks get executed
- The variable affects the computational load of any task
- The variable size of inputs/outputs of any task.

In this case the variables gb->nsteps determines whether the task initvel gets executed. Therefore operations on this variable should be executed during the task discovery phase and therefore these operations are not encapsulated by the gs_smart_local_region function. If any variables or structures are updated during the task discovery cycle then they should be restored to their original values before the execution cycle begins. The variable gb->nb_steps is restored back to its original value (nb_steps_old) once task discovery is finished.

GridRPC Implementation

```
nb_evolutions=2;

 while(gb->nb_steps < nb_evolutions) {

   ga = gb->gconst/gb->at;

   grpc_call(grav_hndl,... , ga, ... , gb->nsteps, ...);

   if(gb->nb_steps==0){
     grpc_call(intivel_hndl, ... );
   }

   grpc_call_async(dark_hndl,&sid_dark, ... , gb->dmvelmax, ... );
   grpc_call_async(bary_hndl,&sid_bary, ... , gb->nsteps, ... , gb->bmvelmax);

   /* wait for non blocking calls to finish */
   grpc_wait(sid_dark);
   grpc_wait(sid_bary);

   timestep(gb->dmvelmax, ...);
   gb->bmvelmax=0;
   gb->dmvelmax=0;

   gb->nb_steps++;

}
```

SmartGridSolve Implementation

```
nb_evolutions=2;

//store value of gb->nb_steps before
//task discovery
nb_steps_old=gb->nb_steps;

gs_smart_map("ex_map"){

   //restore variables changed during task discovery
   gb->nb_steps=nb_steps_old;

   while(gb->nb_steps < nb_evolutions) {

      if(gs_smart_local_region()){
        ga = gb->gconst/gb->at;
      }

      grpc_call(grav_hndl,... , ga, ... , gb->nsteps, ...);

      if(gb->nb_steps==0){
        grpc_call(intivel_hndl,...);
      }


      grpc_call_async(dark_hndl,&sid_dark, ... , gb->dmvelmax, ... );
      grpc_call_async(bary_hndl,&sid_bary, ... , gb->nsteps, ... , gb->bmvelmax);

      /* wait for non blocking calls to finish */
      grpc_wait(sid_dark);
      grpc_wait(sid_bary);

      if(gs_smart_local_region()){
        timestep(gb->dmvelmax, ...);
        gb->bmvelmax=0;
        gb->dmvelmax=0;
      }
      gb->nb_steps++;

   }
}
```

5. Installation of Hydropad

The installation procedure in Hydropad uses the GNU auto-tools (autoconf, automake and libtools) and Makefile to help the user to compile and install properly the application. The auto-tools generate a *configure* shell script that automatically check if the computer contains all the necessary programs and libraries to compile the application. At this point of development Hydropad was tested only in a x86 platform with a Linux environment. Hydropad computational code is written in Fortran 90 while the kernel is written in C language. To be able to compile Hydropad the host machine needs to have these two compilers installed. Hydropad uses the library FFTW, in the gravitational task, to compute the discrete Fourier transform. The *configure* script check if this library is installed in the host machine. If this is not the case the package contains a x86 version of the library. Hydropad application is composed of three executable files: *hydropad seq*, *hydropad gs*, *hydropad smart*. The first file is the original Hydropad application, it executes the computation sequentially in a local computer. The other two executables use the GridRPC protocol to compute the different tasks of Hydropad in a Grid environment. The difference between the two files is that the *hydropad gs* uses the standard middleware GridSolve while the *hydropad smart* utilises SmartGridSolve middleware that introduces new extensions in GridRPC. The last two executables need to be compiled with the GridSolve
and SmartGridSolve libraries, if the host machine does not contain these libraries only the sequential executable will be generated. The configure shell script will automatically check the presence of these libraries. The various computational tasks, to be used inside GridSolve or SmartGridSolve, need to be compiled using the special GridSolve problem compiler. The problem compiler, to include a task inside the Grid environment, need the libraries that contain the code of the task and a gsIDL file that describe the task. The Hydropad package contains the libraries and gsIDL file necessary to do this operation furthermore it contains a specific makefile command to simplify this operation.
The installation procedure is composed of the following steps:

1. Retrieve the package `hydropad-1.tar.gz` from the hcl website repository (hcl.ucd.ie) and unpack the files in a local directory with the shell command:

```
$cd /path/local/
$tar zxfv hydropad-1.tar.gz
```

2. Execute the configure shell script to generate the makefile:
`$./configure --prefix=/path`
With the argument *–prefix=/path* is possible to choose the directory where to install Hydropad. The default location is `/usr/local`, the user need to have the right to write in this directory to install the application. The GridSolve and SmartGridSolve libraries have to be installed in the machines to generate the respective executables. If the configure script find the libraries it will print the following message:

```
checking for grpc.h... yes
checking for GridSolve library... yes
checking for GridSolve... yes
checking for gs_smart_clib.h... yes
checking for SmartSolve library... yes
checking for SmartSolve... yes
```

3. Compile the application with the command:
`$make`

4. Install the Hydropad libraries and executables with the command:
`$make install`
To run the application, the directory that contain the executables file need to be included in the environment variable "$PATH" while the directory that contains the libraries has to be included in the variable "$LD LIBRARY PATH".

5. Compile the tasks problems to include the Hydropad tasks inside GridSolve or SmartGridSolve with the command:
`$make services`

Other than the three version of the Hydropad executable, the package contains also another application, *grafic*. This application is used to generate the initial computational value and it is utilised by the task *usegrafic*. To be able to run Hydropad with GridSolve/SmartGridSolve, the *grafic* file need to be executable in each server included in the Grid environment.
Consequently for each server the *grafic* file has to be installed and the directory that contains it has to be inserted in the environment variable "$PATH". After this procedure the Hydropad application is ready to be executed sequentially or with GridSolve/SmartGridSolve.

The Hydropad application also needs a specific input file that contains the initial value of some of the physics variables used in the simulation. The name of this file is passed at command line by using the specific argument "*–input*". Table 1 shows an example of the input file. The most important values are the third and forth one. The Third value indicate the number of particles utilised in the dark matter *N-Body* method while the fourth value indicate the number of cells for grid used for the baryonic matter in the simulation. Bigger are these two values and more precise is the simulation but more memory is used and the computation take more time. Hydropad cannot work with any values for these two variables. The values have to be even and the number of particles does not have to be more than double the number of cell for side grid. Table 2 shows an example of acceptable value and the quantity of memory used to run the application.

Table 1. Hydropad input file

LCDM	Cosmological model
Michele Guidolin	Owner of the simulation
64	Numper of particles, Nparmax = Np^3
64	Ng Grid sides size, X Y Z
30.0	Box size (Mpc/h)
0.71	Huble parameter
0.226	Omega dark matter
0.044	Omega baryonic matter
0.73	Cosmological constant
0.01	Present BM average temperature
0.75	Hydrogen mass fraction
0.948	Long-wave spectral index
0.772	Desired normalization
314159265	Random number seed (9-digit integer)

Table 2. Example of input values and problem sizes for Hydropad

Np	Ng	Data Size
120	60	73MB
140	80	142MB
160	80	176MB
140	100	242MB
160	100	270MB
180	100	313MB
200	100	340MB
220	120	552MB
240	120	624MB

Hydropad uses other arguments in the command line to change the behaviour of the application. The arguments are:
–input chose the input file;
–notsc use faster CIC for interpolation instead of the slower triangular shaped clouds;
–nmap number of SmartGridSolve map;
–cycles number of cycles for SmartGridSolve map or total number of cycles in the simulation.

The following example execute a local sequential computation of Hydropad with the input file of table 1 and 10 cycles of evolution in the simulation.

```
$hydropad_seq --input input.in --cycles 10
```