Grid-Enabled Hydropad: a Scientific Application for Benchmarking GridRPC-Based Programming Systems

Michele Guidolin, Alexey Lastovetsky
School of Computer Science and Informatics
University College Dublin
Belfield, Dublin 4, Ireland
{ michele.guidolin, alexey.lastovetsky }@ucd.ie

Abstract

GridRPC is a standard API that allows an application to easily interface with a Grid environment. It implements a remote procedure call with a single task map and client-server communication model. In addition to non-performance-related benefits, scientific applications having large computation and small communication tasks can also obtain important performance gains by being implemented in GridPRC. However, such convenient applications are not representative of the majority of scientific applications and therefore cannot serve as fair benchmarks for comparison of the performance of different GridRPC-based systems. In this paper, we present Hydropad, a real life astrophysical simulation, which is composed of tasks that have a balanced ratio between computation and communication. While Hydropad is not the ideal application for performance benefits from its implementation with GridRPC middleware, we show how even its performance can be improved by using GridSolve and SmartGridSolve. We believe that the Grid-enabled Hydropad is a good candidate application to benchmark GridRPC-based programming systems in order to justify their use for high performance scientific computing.

1. Introduction

A typical numerical simulation needs a lot of computational power and memory footprint to solve a physical problem with a high accuracy. A single hardware platform that has enough computational power and memory to handle problems of high complexity is not easy to access. Grid computing provides an easy way to gather computational resources, whether local or geographically separated, that can be pooled together to solve large problems. GridRPC [8] is a standard API promoted by the Open Grid Forum that allows the user to smoothly design an application to interface with a Grid environment. Currently a number of Grid middleware systems are GridRPC compliant including GridSolve [10], Ninf-G [9] and DIET [4]. Performance improvements are not the only goals of these systems, however they are designed to achieve high performance in execution of scientific applications. A good GridRPC-based programming system permits a typical scientific application to gain non-performance-related benefits, like ease of development and control of the application, while not compromising or even improving its performance.

A GridRPC middleware works by individually mapping the application's tasks to appropriate servers in the Grid and communicating the data between the servers and the client computer. In a remote execution all the data used by a task has to be available on the chosen server, consequently for each task there is a high quantity of data communication. A scientific application, that obviously benefits from the use of GridRPC, consists of tasks that are highly computationally intensive and low in data communication. These applications, which are the best suited to run on a Grid environment, are not representative of many real-life scientific applications. Unfortunately they are typically chosen, or artificially created, to test and show the performance of a GridRPC middleware system. We believe that to justify the use of GridRPC for a wide range of applications, we should not use an extremely suitable application as a benchmark but a real life application that shows the eventual limits and benefits of the GridRPC middleware systems tested.

In this work, we present Hydropad, a real-life astrophysical application that simulates the evolution of clusters of galaxies in the universe [6]. This application is composed of tasks that have a balanced ratio between computation and communication. Hydropad requires high processing resources because it has to simulate an area comparable to the dimension of the universe and simultaneously try to achieve a high enough resolution to show how the stars developed.

In section 3, we introduce the motivations and benefits behind the use of GridRPC in Hydropad and how it is implemented. We also present experimental results obtained for the GridSolve version of Hydropad demonstrating that in many realistic situations this GridRPC implementation will outperform the original sequential Hydropad. In section 4, we introduce SmartGridSolve [2], a new middleware that extends the execution model of GridRPC to overcome its limitations. We demonstrate that SmartGridSolve can significantly improve the performance of Hydropad even in situations where GridSolve fails to do it.

2. Hydropad: a Simulator of Galaxies' Evolution

Hydropad is a cosmological application, originally written by Claudio Gheller, which simulates the evolution of clusters of galaxies in the universe [6]. 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 only be described by treating both components at the same time, looking at all of their internal processes, while their mutual interaction is regulated by a gravitational component. Figure 1 shows an example of a typical output generated by Hydropad.

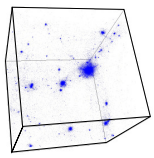

Figure 1. Example of Hydropad Output

The dark matter computation can be simulated using *N-Body* methods [7]. These methods utilise the interactions between a large number, N_p, of collision-less particles. These particles, subjected to gravitational forces, can simulate the process of the formation of galaxies. The accuracy of this simulation depends on the quantity of particles used. Hydropad utilises a *Particle-Mesh* (PM) N-Body algorithm, which has a linear computational cost and depends on the number of particles $O(N_p)$. In the first part this method transforms the particles, through an interpolation, into a grid of density values. Afterwards the gravitational potential is calculated from this density grid. In the last part the particles are moved depending on the gravitational forces of the cell where they were located.

The baryonic matter computation utilises a *Piecewise-Parabolic-Method* (PPM) Hydrodynamic algorithm [5]. This is a higher order method for solving partial differential equations. PPM reproduces the formation of pressure forces and the heating and cooling processes generated by the baryonic component during the formation of galaxies. For each time step of the evolution, the fluid quantities of the baryonic matter are estimated over the cells of the grid by using the gravitational potential. The density of this matter is then retrieved and used to calculate the gravitational forces for the next time step. The accuracy of this method depends on the number of cells of the grid used, N_g, and its computational cost is linear $O(N_g)$. The application computes the gravitational forces, needed in the two previous algorithms, by using the Fast-Fourier-Transform (FFT) method to solve the Poisson equation. This method has a computational cost of $O(N_g \log N_g)$. All the data, used by the different components in Hydropad, are stored and manipulated in three-dimensional grid-like structures. In the application, the uniformity of these base structures permits easy interaction between the different methods.

Figure 2 shows the work-flow of the Hydropad application. It is composed of two parts: the initialisation of the data and the main computation. The main computation 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 (FFT method), the dark matter task (PM method) and the baryonic matter task (PPM method). 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.

The initialisation part is also divided in two independent tasks. The main characteristic of dark matter initialisation is that the output data is generated by the external application *grafic*, a module of the package COSMICS [1]. *Grafic*, given the initial parameters as an input, generates the position and velocity of the particles that will be used in the N-Body method. The output data is stored in two files which information has to be read by the application during the initialisation part. Like the main application, *grafic* has a high memory footprint.

An important characteristic of Hydropad is the difference in computational and memory load of its tasks. Despite both algorithms being linear, the computational load of the baryonic matter component is far greater than the dark matter one, $C_{bm} \gg C_{dm}$, when the number of particles is

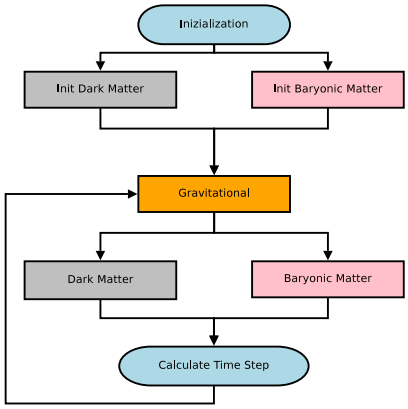

Figure 2. Internal structure of Hydropad

equal to the number of cells in the grid, $N_p = N_g$. Furthermore the quantity of data used by the dark matter computation is greater than the baryonic matter one, $D_{dm} \gg D_{bm}$.

As previously indicated Hydropad utilises three dimensional grid structures to represent the data. In the application code these grids are represented as vectors. In the case of the dark matter component, the application stores the position and velocity in three vectors for each particle, one for each dimension. The size of these vectors depends on the number of particles, N_p, chosen to run on the simulation. For the gravitational and baryonic components the different physical variables, such as force or pressure, are stored in vectors, with the size depending on the given number of grid cells N_g. In a typical simulation the number of particles is of the order of billions, while the number of cells in a grid can be over 1024 for each grid side. Given that for the values of $N_g = 128^3$ and $N_p = 10^6$ the total amount of memory used in the application is roughly $500MB$, the memory demand to run a typical simulation is very high.

3. Enabling Hydropad for Grid Computing

GridRPC provides a simple remote procedure call (RPC) to execute, synchronously or asynchronously, a task in a Grid environment. GridRPC differs from the traditional RPC method since the programmer does not need to specify the server to execute the task. When the Grid-enabled application runs, each GridRPC call results in the middleware mapping the call to a remote server and then the middleware is responsible for the execution of that task on the mapped server. As a result, each task is mapped separately and independently of other tasks of the application. Another important aspect of GridRPC is its communication model. For each task, the GridRPC middleware sends all the input data from the client machine to the remote server. Then, after the remote task has finished its execution, the middleware retrieves the output data back to the client machine. Therefore, remote execution of each task results in significant amount of data communicated between the client machine and servers.

Hydropad is not the ideal application for execution in a Grid environment because of the relatively low complexity of its tasks (log-linear at maximum) and the large amount of input and output data moved between tasks. In this work, we study how such an application can benefit from implementation in GridRPC. The performance related benefits include the potential for faster solution of a problem of a given size and solution of problems of larger sizes.

Faster solution of a given problem. Grid-enabled Hydropad has the potential to perform the simulations of the same given size faster than the original Hydropad on the client machine. There are two main reasons for this:

- The Hydropad application includes two independent tasks, the baryonic matter task and the dark matter task, that can be executed in parallel. The non-blocking GridRPC task call API allows us to implement their parallel execution on remote servers of the Grid environment. This parallelisation will decrease the computation time of the application.

- If the Grid environment contains machines more powerful than the client machine, then remote execution of the tasks of this application on these more powerful machines will also decrease the computation time of the application.

However, this decrease of the computation time does not come for free. The application will pay the communication cost due to remote execution of the tasks. If communication links connecting the client machine and the remote servers are relatively slow, than the acceleration of computations will be compensated by the communication cost resulting in the total execution time of the application higher than in the case of its sequential execution on the client machine. For example, experiments with Hydropad in section 3.2 show that with a 100 Mbit/sec connection between the client machine and the servers the Grid-enabled Hydropad is slower than the original serial one. At the same time, for a 1 Gbit/sec connection the Grid-enabled Hydropad was faster than its sequential counterpart. Thus, in many realistic Grid environments, the Grid-enabled Hydropad can outperform its original sequential version.

Solution of larger problems. Grid-enabled Hydropad has the potential to perform larger simulations resulting in their higher accuracy. Indeed, the baryonic and dark matter tasks allocate temporary memory during their execution. Remote execution of these tasks will decrease the amount of

memory used on the client machine as the temporary memory is now allocated on remote machines. Therefore, within the same memory limitations on the client machine (say, the amount of memory that can be used by the application without heavy paging), the Grid-enabled Hydropad will allow for larger simulations.

The use of GridRPC for scientific applications does not only bring performance related advantages. Other benefits may be more difficult to notice but are equally important.

More control over the application. Hydropad potentially can be executed not only in a Grid environment but also in a high performance computer (HPC) system. Unfortunately in a HPC system, where applications are executed in batch mode, the user will not have much control over the execution. Grid-enabled Hydropad allows the user to have a high control over its execution because, although the tasks are being computed in remote servers, the main component of the application is running on the client machine. This can be important for many types of applications, some examples are:

- Applications that need a direct interaction with the data produced. For example the user could visualise directly in the client machine the evolution of the universe, while Hydropad is running on the Grid. Furthermore while the user is checking the simulation evolution, he could decide on the fly to change some parameters of the simulation or restart the application. This is possible since in Grid-enabled Hydropad the main data and the main execution is on the client machine.
- Applications that have a task that is inherently remote. For example in the case of Hydropad, if *grafic* cannot be executed on the client machine because it needs a specific hardware, the user has to generate the initial data on the remote server and then manually retrieve it. The use of GridRPC can simplify this situation by allowing a special task to interface with *grafic* directly on the remote server. This task can communicate immediately the initial data generated by *grafic* to the application.

An easy and powerful development paradigm. A numerical method, to be executed remotely, has to avoid internal state changes, like a function with isolated computation and no global variable. This method of development creates tasks that have a specific interface for input/output values. Therefore, the GridRPC tasks can be easily reused in other Grid applications because their execution with the same input always produces the same output. This situation can reduce the programmer effort on developing a Grid application. For example the programmer can use already existing tasks that he would not have the time or skill to write. Additionally if the application needs to use tasks that are inherently remote because they are made of proprietary code or bound to a specific hardware, like *grafic* in the previous example, the programmer can easily include them.

3.1. GridRPC Implementation of Hydropad

Hydropad was originally a sequential *Fortran* code, we upgraded this program to take advantage of the GridRPC API and to work with the GridSolve middleware. Table 1 shows the original Hydropad code of the main loop, written in the C language. Three functions, *grav*, *dark*, and *bary*, are called in this loop to perform the three main tasks of the application. In addition, at the first iteration of this loop, a special task, *initvel* is called to initialise the velocities of the particles. The dark and baryonic tasks compute the general velocities of the respective matter. At each iteration, these velocities are used by a local function, *timestep*, to calculate the next time step of the simulation. The simulation will continue until this time becomes equal to the present time of the universe, $t_{sim} = t_{univ}$.

Table 1. Hydropad evolve loop

```
t_sim=0;
while(t_sim<t_univ) {
    grav(phi,phiold,rhoddm,rhobm,...);
    if(t_sim==0){ initvel(phi,...); }
    dark(xdm,vdm,...,veldm);
    bary(nes,phi,...,velbm);
    timestep(veldm,velbm,...,t_step);
    t_sim+=t_step;
}
```

The GridRPC implementation of Hydropad application uses the APIs *grpc_call* and *grpc_call_async* to execute respectively a blocking and an asynchronous remote call of the Fortran routines. The first argument of both APIs is the handler of the task executed, the second is the session ID of the remote call while the following arguments are the parameters of the task. Furthermore, the code uses the method *grpc_wait* to block the execution until the chosen, previously issued, asynchronous request has completed. When the program runs, the GridSolve middleware maps each *grpc_call* and *grpc_call_async* functions singularly to a remote server. Then, the middleware communicates the data from the client computer to the chosen server and then executes the task remotely. At the end of the task execution, the data is communicated back to the client. In the blocking call method, the client cannot continue the execution until the task is finished and all the outputs have been returned. Instead, in the asynchronous method, the client does not wait for the task to finish and proceeds immediately to execute

the next code. The output of the remote task is retrieved when the respective wait call function is executed.

Table 2. Hydropad implementation in GridRPC

```
t_sim=0;
 while(t_sim<t_total) {
  grpc_call( grav_hndl,phiold,...);
  if(t_sim==0){ grpc_call(initvel_hndl,phi,...); }
  grpc_call_async( dark_hndl,&sid_dark,x1,...);
  grpc_call_async( bary_hndl,&sid_bary,nes,...);

  grpc_wait( sid_dark); /*wait for non blocking*/
  grpc_wait( sid_bary); /*calls to finish*/
  timestep(t_step,...);
  t_sim+=t_step;
}
```

Table 2 outlines the GridRPC implementation of the main loop of Hydropad that simulates the evolution of universe. At each iteration of the loop, the first *grpc_call* results in the gravitational task being mapped and then executed. When this task is completed, the client proceeds to the next call, which is a non-blocking call of the dark matter task. This call returns after the task is mapped and its execution is initiated. Then, the baryonic matter call is executed in the same way. Therefore, the baryonic and dark matter tasks are executed in parallel. After this, the client waits for the outputs of both these parallel tasks using the *grpc_wait* calls.

3.2. Experiments with the GridSolve-Enabled Hydropad

In this section, we compare the execution times and memory footprints of the GridSolve implementation of Hydropad against its sequential execution on the client machine. The hardware configuration used in the experiments consists of three machines: a client and two remote servers, S1 and S2. The two servers are heterogeneous however they have similar performance, respectively 498 and 531 MFlops, and they have an equal amount of main memory, 1GB each. The client machine is a computer with low hardware specifications, 256MB of memory and 248MFlops of performance, which is not suitable to perform large simulations. The bandwidth of the communication link between the two servers is 1Gb/s. The client-to-server connection varies depending on the experimental setup. We use two setups, C1 with a 1Gb/s connection and C2 with a 100Mb/s communication link. These hardware configurations represent a situation when a user having a relatively weak computer can access more powerful machines. For each conducted experiment, table 3 shows the initial problem parameters and the corresponding data sizes (the total memory used during the execution of Hydropad on a single machine).

Table 3. Input values and problem sizes for the Hydropad experiments

Problem ID	N_p	N_g	Data Size
P1	120^3	60^3	73MB
P2	140^3	80^3	142MB
P3	160^3	80^3	176MB
P4	140^3	100^3	242MB
P5	160^3	100^3	270MB
P6	180^3	100^3	313MB
P7	200^3	100^3	340MB
P8	220^3	120^3	552MB
P9	240^3	120^3	624MB

Table 4 shows the average computation time of one evolution step achieved by the local computation (in subtable (a)), and by the GridSolve version of Hydropad (in subtable (b) and (c)). This table also introduces the results obtained by the SmartGridSolve version of Hydropad, they will be discussed in section 4.

Table 4 also presents the scale of paging that occurs in the client machine during the executions. It is possible to see in table 4(a) that for the local computation the paging is taking place when the problem size is equal or greater than the machine memory, 256MB. For the GridSolve version the paging is occurring later, when the problem size is around 310MB, as shown in table 4(b)/(c). The GridRPC implementation can save memory thanks to the temporary data allocated remotely in the tasks and consequently increase the problem size that will not cause the paging. In the sequential local execution the paging is taking place during a task computation, while for the GridSolve version the paging occurs during a remote task data communication. Hence for the Grid-enabled Hydropad the paging on the client machine does not negatively affect the execution time of the experiments.

The experiments in table 4(b) were executed using C1 as client. This machine has a fast network link to the servers S1 and S2. The results in this subtable show that the speedup (S_p) obtained by GridSolve is around 2 until the client machine starts paging, then the local computation receives a heavy penalty from the paging. Figure 3 shows the execution times of the evolution step for the local computation and for the GridSolve version of Hydropad.

Table 4(c) shows the results obtained by the GridSolve version when the client machine used, C2, has a slow client-to-servers connection of 100Mb/s. The GridSolve version is slower than the local computation when the client machine is not paging. This is happening because there is a large amount of data communication between tasks. So for this configuration, the time spent communicating the data compensates the time gained by computing tasks remotely. However as the problem size gets larger and the client ma-

Table 4. Experimental results

(a) Local

P.ID	Time Step	Paging
P1	14.09s	No
P2	29.95s	No
P3	35.29s	No
P4	55.13s	Light
P5	61.63s	Light
P6	83.66s	Yes
P7	128.55s	Yes
P8	227.89s	Heavy
P9	280.07s	Heavy

(b) GridSolve C1 - 1GB/s

Time Step	Paging	S_p v Local
7.20s	No	1.96
15.51s	No	1.93
16.48s	No	2.14
29.11s	No	2.14
29.07s	No	2.12
36.74s	Light	2.28
48.06s	Yes	2.67
77.91s	Heavy	2.92
91.75s	Heavy	3.06

(c) GridSolve C2 - 100MB/s

Time Step	Paging	S_p v Local
18.01s	No	0.78
35.02s	No	0.86
43.09s	No	0.82
55.66s	No	0.97
58.17s	No	1.06
72.50s	Light	1.15
80.05s	Yes	1.61
133.47s	Heavy	1.71
155.36s	Heavy	1.81

(d) SmartGridSolve C1 - 1GB/s

P.ID	Time Step	Paging	S_p v Local	S_p v GS (C1)
P1	6.99s	No	2.02	1.03
P2	14.69s	No	2.04	1.06
P3	15.52s	No	2.27	1.06
P4	27.22s	No	2.03	1.07
P5	27.13s	No	2.27	1.07
P6	27.22s	No	3.07	1.35
P7	29.13s	Light	4.41	1.65
P8	49.21s	Light	4.63	1.58
P9	50.82s	Light	5.52	1.81

(e) SmartGridSolve C2 - 100MB/s

Time Step	Paging	S_p v Local	S_p v GS (C2)
7.9s	No	1.78	2.28
15.68s	No	1.91	2.75
17.36s	No	2.03	2.48
28.56s	No	1.93	1.98
28.77s	No	2.14	2.02
30.09s	No	2.78	2.41
31.63s	Light	4.06	2.53
52.30s	Light	4.36	2.55
55.47s	Light	5.06	2.80

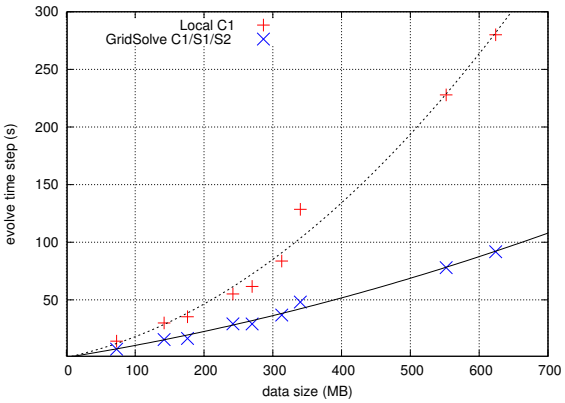

Figure 3. Evolution time step of the local and GridSolve computation with client C1

chine starts paging, the GridSolve version becomes faster than the local computation, even in the case of slow communication between the client and server machines.

4. SmartGridSolve and Hydropad

In this section we introduce SmartGridSolve [2], previously implemented as SmartNetSolve [3]. SmartGridSolve is an extension of GridSolve that has been designed to bypass the limitations of the GridRPC model of execution. The GridRPC implementation of Hydropad has some advantages over the sequential local computation, however it is evident that the model of execution utilised by GridRPC is not optimal. In a GridRPC system all tasks are mapped individually. The mapper will always choose the fastest available server at the instant that a task is called, regardless of the computational size of the task and regardless of whether the task is to be executed sequentially or in parallel.

A drawback of this behaviour is highlighted by the Hydropad application. The parallel tasks in Hydropad are not computationally balanced. The baryonic task is computationally far larger than the dark matter one, $C_{bm} \gg C_{dm}$. When a GridRPC system goes to map these two tasks, it does so without the knowledge that they are part of a group to be executed in parallel. Its only goal is to minimise the execution time of an individual tasks as it is called by the application. If the smaller dark matter task is called first it will be mapped to the fastest available server. With the fastest server occupied, the larger baryonic task will then be mapped to a slower server and the overall execution time of the group of tasks will be sub-optimal.

Another constraint of the GridRPC model, which influences the performance of Hydropad or any other application, is that all the data computed remotely and communicated between remote tasks has to pass through the client machine. Servers computing tasks with data dependencies on each other cannot communicate with each other directly. It is possible for the application programmer to avoid this issue by implementing data caching in his tasks. However it requires the programmer to make heavy modification to the tasks and this is a clear drawback. It also means that remote tasks passing data to each other must all run on the same server, where the data they need is cached.

SmartGridSolve addresses all these issues. It expands the single task map and client-server model of GridRPC by implementing the mapping of groups of tasks, the automatic

data caching on servers and the server to server communication. Collective mapping of groups of tasks, using a fully connected network, allows SmartGridSolve to find an optimal mapping solution for an application that fully exploits a Grid environment. Furthermore the direct server to server communication and automatic data caching that SmartGridSolve implements minimises the amount of memory used on the client and the volume of communication necessary between client and server. Data objects can reside only on the servers where they are needed and they can be moved directly between servers without having to pass through the client. The main goal of SmartGridSolve is to provide these functionalities to the user in a practical and simple way. To achieve this it requires only minor changes and addition to the APIs of GridRPC. An application programmer can gain from the improved performance using SmartGridSolve by making only minor modifications to any application that is already GridRPC enabled.

4.1. SmartGridSolve Implementation of Hydropad

The code in table 5 shows the modifications required to use the new SmartGridSolve features in Hydropad, in contrast to those shown in table 2 where we illustrate the changes required for GridSolve/GridRPC. One can see that the difference between the examples is the minor additions of: the *gs_smart_map* block and *gs_smart_local_region* condition. These belong to the SmartGridSolve API.

The code enclosed in the *gs_smart_map* block will be iterated through twice. On the first iteration, each *grpc_call* and *grpc_call_async* is discovered but not executed. At the beginning of the second iteration, when all the tasks within the scope of the block have been discovered, a task graph for them is generated. The discovered tasks are then executed remotely using this task graph to aid their mapping [2]. The *gs_smart_local_region* function, in conjunction with a conditional statement, is used by the application programmer to indicate when a local computation is executed. At run time on the first discovery iteration the code within this conditional statement is not executed. This is to mimic the behaviour that the remote calls have on the discovering iteration. On the second iteration, the code inside the statement is executed normally.

The mapping in the code of table 5 is performed at every iteration of the main loop, this can generate a good mapping solution if the Grid environment is not a stable one. For example, where there are other applications' tasks running on the Grid servers. If the Grid environment is dedicated, where only one application executes at a time, a better mapping solution may be generated if the area to map contains more tasks, i.e. two or more loop cycles. A simple solution could be including an inner loop within the *gs_smart_map* code block. The application programmer could increase the number of tasks mapped together by changing increasing the number of iterations of the inner loop.

Table 5. Hydropad implementation in SmartGridSolve

```
t_sim=0;
 while(t_sim<t_univ) {
   gs_smart_map("ex_map"){
    grpc_call( grav_hndl,phiold,...);
    if(t_sim==0){ grpc_call(initvel_hndl,phi,...);}
    grpc_call_async( dark_hndl,&sid_dark,x1,...);
    grpc_call_async( bary_hndl,&sid_bary,nes,...);
    /* wait for non blocking calls to finish */
    grpc_wait( sid_dark);
    grpc_wait( sid_bary);
     if( gs_smart_local_region()){
       timestep(t_step,...);
       t_sim+=t_step;
     }
  }
}
```

4.2. Experimental Results Using SmartGridSolve

In this section we show the results obtained by the SmartGridSolve version of Hydropad and we compare them with those from the GridSolve and local versions shown in section 3.2. The problem sizes utilised in the experiments (table 3) and the hardware configurations are the same as in previous experiments. As mentioned before, one of the primary improvements of SmartGridSolve is its communication model, use of which minimises the amount of data movement between the client and servers. This advantage is most prominent when the client connection to the Grid environment is slow. Table 4(e) shows the results obtained by the SmartGridSolve version of Hydropad using C2 as the client machine which has a slow network connection of 100Mb/s. One can see that the SmartGridSolve version is much faster than the GridSolve, table 4(c), and the sequential versions, table 4(a). The increase of speed is over twice that of GridSolve, which is primarily due to the improved communication model of SmartGridSolve.

Another important feature of SmartGridSolve is the superior mapping system. Table 4(d) shows results obtained from experiments using C1 as the client machine. This machine has a higher speed network connection of 1Gb/s. The results show the performance gain obtained due to the improved mapping method. The advantage gained by using the communication model of SmartGridSolve is minimised by the faster communication links (experiments with a single server were performed to confirm this). Despite Hy-

dropad having only two parallel tasks the SmartGridSolve mapper can produce a faster execution than the GridSolve one, table 4(b).

A secondary advantage of the direct server to server communication implemented in SmartGridSolve is that the quantity of memory used in the client machine is lower than that of the GridSolve version. Therefore the SmartGridSolve version of Hydropad can execute larger problems without the paging on the client machine. This can influence the execution time for larger problems as it is shown in table 4(d). The speed-up of SmartGridSolve over GridSolve, when the client machine pages, increases as the problem gets larger. This trend is also seen in figure 4.

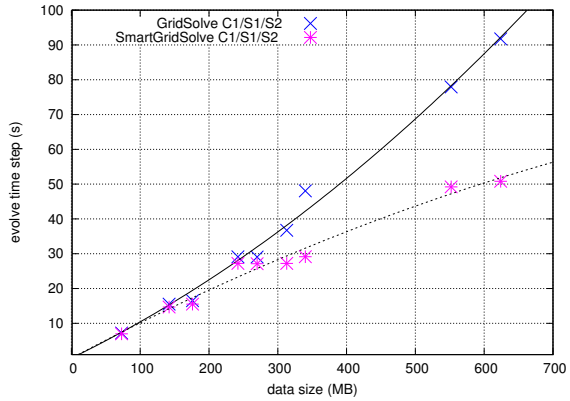

Figure 4. Execution times of the GridSolve and SmartGridSolve versions of Hydropad

5 Conclusions

Grid-enabled Hydropad was originally a sequential code that we upgraded to utilise the GridRPC API to interface with a Grid environment. The main loop of this application is composed of three tasks of which two can be executed in parallel. These tasks have at maximum a log-linear complexity and there is a high amount of data communication between them. Despite the fact that these types of tasks are not the best suitable to be executed on a Grid because of the high magnitude of communication involved, Hydropad can obtain many benefits from being Grid-enabled. These benefits can be related to performance gains or to the management and development aspects of the application.

The experimental results presented in this paper show that Grid-enabled Hydropad, when is executed over GridSolve middleware, can achieve better performance than the original sequential code. However these performance gains are correlated to the link speed of the connection between the client machine and sever machines. Additional experiments show that SmartGridSolve middleware allows Hydropad to obtain quite significant performance gains in comparison to the GridSolve version and to the sequential one. Furthermore the experiment shows that these gains are not influenced negatively by a slow client-servers connection as much as with the GridSolve version.

Grid-enabled Hydropad is a freely available application that could represent a good benchmark for GridRPC-based programming systems because it exemplifies typical real-life scientific applications, which are not perfectly suitable for execution in a Grid environment, that push the limits of a GridRPC middleware. This work was supported by Science Foundation Ireland. A package containing the Hydropad application can be found at the UCD Heterogeneous Computing Laboratory web site: *hcl.ucd.ie*.

References

[1] E. Bertschinger. COSMICS: Cosmological Initial Conditions and Microwave Anisotropy Codes. *ArXiv Astrophysics e-prints*, June 1995.

[2] T. Brady, M. Guidolin, and A. Lastovetsky. Experiments with SmartGridSolve: Achieving Higher Performance by Improving the GridRPC Model. In *Proceedings of the 9th IEEE/ACM International Conference on Grid Computing (Grid 2008)*, Tsukuba, Japan, 29 September - 01 October 2008. IEEE Computer Society.

[3] T. Brady, E. Konstantinov, and A. Lastovetsky. SmartNetSolve: High Level Programming System for High Performance Grid Computing. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, Rhodes Island, Greece, 25-29 April 2006.

[4] E. Caron and F. Desprez. DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid. *International Journal of High Performance Computing Applications*, 20(3):335–352, 2006. Sage Science Press.

[5] P. Colella and P. Woodward. The piecewise parabolic method (PPM) for gas-dynamical simulations. *Journal of Computational Physics*, 54:174–201, 1984.

[6] C. Gheller, O. Pantano, and L. Moscardini. A cosmological hydrodynamic code based on the Piecewise Parabolic Method. *Royal Astronomical Society, Monthly Notices*, 295(3):519–533, 1998. Blackwell Publishing.

[7] R. Hockney and J. Eastwood. *Computer Simulation Using Particles*. McGraw Hill, New York, 1981.

[8] K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee, and H. Casanova. Overview of GridRPC: A Remote Procedure Call API for Grid Computing. In *GRID '02: Proceedings of the Third International Workshop on Grid Computing*, pages 274–278, London, UK, 2002. Springer-Verlag.

[9] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, and S. Matsuoka. Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing. *Journal of Grid Computing*, 1(1):41–51, 2003. Springer.

[10] A. YarKhan, K. Seymour, K. Sagi, Z. Shi, and J. Dongarra. Recent Developments in GridSolve. *International Journal of High Performance Computing Applications*, 20(1):131–142, 2006. Sage Science Press.