Experiments with SmartGridSolve: Achieving Higher Performance by Improving the GridRPC Model

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

Abstract

The paper presents SmartGridSolve, an extension of GridSolve, the programming system for high performance computing. The extension is aimed at higher performance of Grid applications by providing the functionality for collective mapping of a group of tasks on to a network topology that is fully connected. This functionality was achieved with only a minor addition to the GridRPC API. The key to the implementation of collective mapping was to separate the mapping of tasks from their execution which is one atomic operation in the GridRPC model of GridSolve.

This paper demonstrates the performance gained by collective mapping with a real-life astrophysical experiment. The presented results show a significant speedup of 2.17 executing this application on a small network of two servers.

1. Introduction

The remote procedure call (RPC) paradigm [1] is widely used in distributed computing. It provides a straightforward procedure for executing parts of an application on a remote computer. To execute a RPC, the application programmer does not need to learn a new programming language but merely use the RPC API. Using the API the application programmer specifies the remote task to be performed, the server to execute the task, the location of the input data on the user's computer required by the task and the location on the user's computer where the results will be stored. The execution of the remote call involves transferring input data from the user's computer to the remote computer, executing the task on the remote server and delivering output data from the remote computer to the user's one.

GridRPC [2] is an emerging standard promoted by the Global Grid Forum which extends the traditional RPC. GridRPC differs from the traditional RPC in that the programmer does not need to specify the server to execute the task. When the programmer does not specify the server, the middleware system which implements the GridRPC API is responsible for finding the remote executing server. When the program 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. A number of grid middleware systems have recently become GridRPC compliant including GridSolve [3], Ninf-G [4] and DIET [5].

This simple extension to the RPC however has some limitations affecting the performance of Grid applications. When using the traditional GridRPC to execute tasks remotely, the mapping and execution of the task is one atomic operation which cannot be separated. As a result, each task is mapped separately and independently of other tasks of the application. This model supports minimization of the execution time of each individual task of the application rather than the minimization of the execution time of the whole application. Mapping tasks individually results in mapping solutions which are far from optimal. If tasks are mapped as they are called, the mapping heuristic is unable to take into account any of the tasks that follow the task being mapped. Consequently, the mapping heuristic does not have the ability to optimally balance the load of computation and communication. Another consequence of mapping tasks in this way is that dependencies between tasks are not known at the time of mapping. Therefore this approach to mapping forces bridge communication. Bridge communication occurs when the output of one task is required as an input to another task. In this case, using the traditional GridRPC, the output of the

first task must be sent back to the client and the client then subsequently sends it to the server executing the second task when it is called. Eliminating bridge communication can significantly decrease the overall communication time of an application and hence improve the overall performance of the application.

In this paper we propose an extension of the traditional GridRPC which would allow a group of tasks to be mapped collectively. Namely, in the execution model of GridRPC we propose to separate the mapping of tasks from their execution. SmartGridSolve, which is an extension to GridSolve is an implementation of this extended model. SmartNetSolve [6], which extended NetSolve [7] was previously implemented to allow for collective mapping of tasks but has since been re-implemented for GridSolve to support the emerging GridRPC model.

There are a number of advantages of mapping tasks collectively. When a group of tasks is mapped collectively the mapping heuristic can improve the performance of that 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

SmartGridSolve has extended GridSolve to support collective mapping of a group of tasks by separating the execution from the mapping of tasks in GridSolve's execution model.

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 which is fully connected. This is a network topology where all servers can communicate directly or server 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 to traverse between servers.

The SmartGridSolve extension is incremental and interoperable with the current version of GridSolve. If SmartGridSolve is installed only on the client side, the system will be extended to allow for collective mapping. If SmartGridSolve is installed on the client side and on any of the servers in the network, the system will be extended to allow for collective mapping on a partially connected network or if it's installed on all servers, the network will be fully connected.

[8][9][10] have extended the GridRPC model to enable direct communication to elimate bridge communication. However, these proposed extensions do not exploit the full potential for minimizing the execution time of a group of tasks on a fully connected network. While they eliminate bridge communication by enabling direct communication between servers they have not changed the fundamental approach to mapping of GridRPC, which is to map each task individually and independently. As a result, they have not fully exploited the potential for performance increase of executing a group of tasks on a fully connected network.

This paper demonstrates with a real-life experiment how SmartGridSolve is better able to exploit the potential of executing tasks on a fully connected topology by collectively mapping a group of related tasks. The real-life experiment that was used for testing and evaluation was an astrophysical application that simulates the evolution of clusters of galaxies in the universe. The paper does not pursue the area of research of mapping heuristics as there has been extensive research already done in this area [11]. However, the SmartGridSolve framework is designed so that these mapping heuristics can be plugged-in within the extended GridRPC model, providing a framework for testing and evaluating these heuristics.

The paper is outlined as follows. Section 2 gives an overview of SmartGridSolve's design. Section 3 describes the astrophysical application and how it is implemented in SmartGridSolve. Experimental results of executing both the GridRPC and the SmartGridSolve implementation of the astrophysical application are given in section 4. And finally, section 5 will conclude the paper.

2. SmartGridSolve design

An important design goal of SmartGridSolve is to provide functionality for collective mapping of a group of tasks on a fully connected network, which is both practical and easy to use for the application programmer. In addition, the aim is to achieve this functionality with only a minor change to the API of GridRPC. Therefore an application programmer can gain from the improved performance of collective mapping by only making minor changes to any application that is already GridRPC enabled. Another

essential design goal is to give the application programmer the authority and flexibility to decide which parts of an application should be collectively mapped. The application programmer does this by using parenthesis to specify the scope of the tasks that should be mapped collectively in the application. Indeed, if specified, the entire application can be mapped collectively. The design also supports the mapping of groups of tasks which contain conditional statements (i.e. if, while, for etc.). Moreover, the application programmer can specify the frequency in which a group of tasks are mapped. For example, if a segment of an application is iterated through a number of times, the application programmer can decide whether to map the group on each iteration or to map all iterations at the same time. Both approaches to mapping are beneficial under different circumstances. Using their knowledge of the environment and the application itself the application programmer can decide which tasks get mapped and the optimal frequency of mappings for their circumstance. This will be described in more detail in section 3.2.

2.1. Design overview

An overview of the SmartGridSolve framework is illustrated in Figure 1 and 2. In Figure 1 a mapping heuristic generates a mapping solution for a group of tasks based on its task graph and the network performance model. The network performance model in SmartGridSolve is a representation of the performance of the servers and communication links of the fully connected network. This performance model is similar to that previously implemented for SmartNetSolve, which was outlined in [6]. Figure 2 illustrates how a mapping solution generated by a mapping heuristic is used during the execution of the group of tasks. The mapping solution not only outlines the task-to-server mapping but also the communication between the tasks in the group, specifying both direct communication and caching.

The key to the implementation of this framework is separating the mapping of a group of tasks from their execution. This requires that all the tasks of the group are discovered and a task graph is generated before any members of the group are called for execution. The GridRPC execution model does not support this framework as tasks are discovered and mapped when they are called for execution.

SmartGridSolve solves this problem by iterating two times through the calls that have been specified for collective mapping. On the first iteration, each call is discovered but not executed. When the last call in the scope of the group of tasks has been discovered, a task graph is generated based on the discovery of these tasks.

At the beginning of the second iteration, the group of tasks is mapped by a mapping heuristic (Figure 1). Then as each task is called for the second time, it is executed according to the mapping solution (Figure 2).

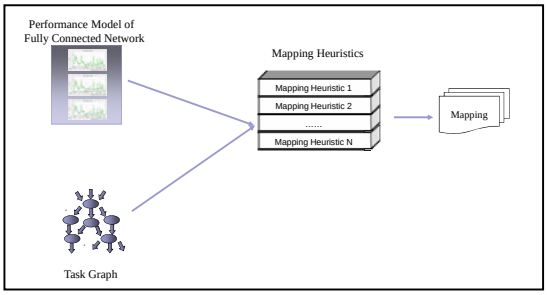

Figure 1. Mapping a group of tasks

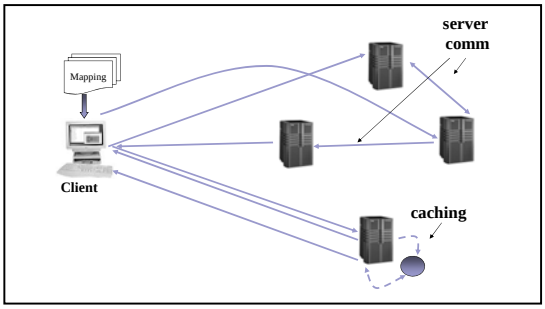

Figure 2. Executing a group of tasks

3. SmartGridSolve test application: The evolution of cluster of galaxies

In this section we introduce Hydropad, which is an astrophysical application written by Claudio Gheller [12], that simulates the evolution of clusters of galaxies in the universe and we show how it was implemented in both GridRPC (section 3.1) and SmartGridSolve (section 3.2) .

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, and the second is dark matter, which accounts for most of the gravitational mass in the Universe. The interaction between the two components is regulated by a gravitational component, which together calculates the movement of clusters of galaxies in the universe.

Figure 3 outlines the work-flow of the Hydropad application. The evolution part of the application 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 gravitation task first calculates the gravitational force and subsequently, the dark matter and the baryonic matter task are executed in parallel, calculating the new position and velocity of their respective types of matter. This evolution step is iterated through a specified number of times, with the output of the previous evolution step being the input to the next.

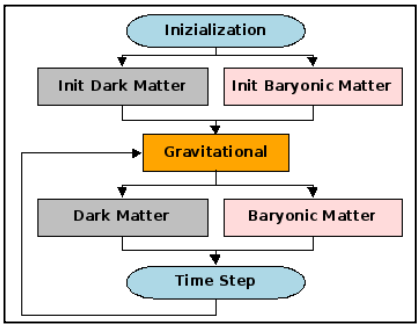

Figure 3. Overview of the Hydropad application

The computational load of the baryonic matter task ($comp_{bm}$) is far greater than the computation load of dark matter task ($comp_{dm}$), where $comp_{bm} >> comp_{dm}$ and in terms of communication, the communication load for the dark matter task is far greater than for the baryonic matter, where $comm_{dm} > comm_{bm}$.

3.1. GridRPC implementation of Hydropad

Figure 4 outlines the GridRPC implementation of an evolutionary step of Hydropad. In the main evolution loop, the gravitational task is executed sequentially with the blocking call "*grpc_call*", while the dark matter and baryonic matter tasks are executed in parallel using the GridPRC call "*grpc_call_async*". The first argument of each call specifies the handle of the remote task to be performed. Each of the subsequent arguments specifies the input and output parameters of the task. When the program runs, each *grpc_call* and *grpc_call_async* function results in the middleware mapping the call to a remote server and then the middleware executing the task on the mapped server.

```
nb_evolutions=2;
for(i=0;i<nb_evolutions;i++) {
   ...
   grpc_call(grav_hndl,phiold,...);
   grpc_call_async(dark_hndl,&sid_dark,x3dm,...);
   grpc_call_async(bary_hndl,&sid_bary,v3bm,...);

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

Figure 4. Hydropad implemented in GridRPC

On each iteration of the loop, the first grpc_call results in the gravitational task first being mapped to a server. The inputs are then sent to the server and the task is executed. The client then waits until the task is finished executing and all the outputs have been returned as it is a blocking task. At this point it proceeds to the next call, which is the dark matter non-blocking call. When this is called, the task is mapped and the execution is initiated. As it is a non-blocking call the client does not wait for the task to finish and proceeds immediately to the next call. When the baryonic matter task is called, it is mapped and again the client initiates the execution but does not wait for the execution to finish. 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.

It is evident from the aforementioned description of the execution of these tasks that the tasks are mapped individually, whether they are executed in sequence or parallel and that all outputs of tasks are sent back to the client, whether dependencies exist or not. Where dependencies do exist, which is the case when the output of one task is the input to another, the output is sent back to the client and then it is sent from the client to the subsequent dependent remote task when it is called.

3.1. SmartGridSolve implementation of Hydropad

Figure In this section we present implementations of the Hydropad application in SmartGridSolve. The first is shown in Figure 5. This shows the SmartGridSolve implementation of the code fragment shown in Figure 4. One can see that it differs from the GridRPC code by one extra call to *gs_smart_map*, which is part of the SmartGridSolve API.

```
nb_evolutions=2;
gs_smart_map("ex_map"){
  for(i=0;i<nb_evolutions;i++) {
    ...
    grpc_call(grav_hndl,phiold,...);
    grpc_call_async(dark_hndl,&sid_dark,x3dm,...);
    grpc_call_async(bary_hndl,&sid_bary,v3bm,...);

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

Figure 5. Hydropad implemented in SmartGridSolve

In this example, two evolutions steps are mapped at the same time using the mapping heuristic specified by the application programmer, which in this case is the exhaustive mapping heuristic.

When the *gs_smart_map* function is executed each call within its parenthesis 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 have been discovered, a task graph is generated that is based on the discovery of this group of tasks. The task graph generated for the code fragment in Figure 5 is illustrated in Figure 6.

The rectangles in the graph represent remote tasks, the diamonds represent the client and the circles represent the input/outputs of the remote tasks. The incoming arrows of these circles indicate their source, whether it's the client or another remote task and the outgoing arrows indicate their destination.

The task graph highlights the order of tasks, their synchronisation (whether they are executed in sequence or parallel), the dependencies between tasks, the load of computation and communication of each task in the group. It is evident from this task graph that there are numerous dependencies between remote tasks in the same evolution step but also between remote tasks in subsequent evolution steps. Using this graph and the performance model of the fully connected network, the mapping heuristics can generate a mapping solution for the group. This mapping solution eliminates bridge communication, either by caching or direct communication and more effectively balances the load of computation and communication of the group over the fully connected network.

This mapping solution is subsequently implemented on the second iteration through the tasks. When each task is called on the second iteration, the task is executed on the server specified in the task-to-server mapping of the mapping solution. The inputs to each task are received from the location specified by the communication scheme of the mapping solution, which is either another remote server or the client. When the task has finished its execution, the output is then sent to the location specified in the communication scheme of the mapping solution. Again, this location may be another server or the client.

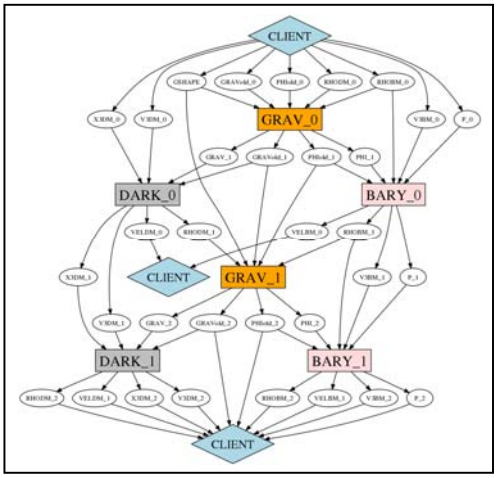

Figure 6. Task graph for two evolution steps

Figure 6 is a task graph generated for only two cycles of the evolution step. It is also possible to map a significantly larger number of evolution steps, by increasing the value of the *nb_evolutions* variable in Figure 5. This type of coarse mapping would be more favourable on a distributed environment which is highly stable, for example a distributed environment that consisted of dedicated servers or servers that are idle. However if the environment is highly changeable, which would be the case if the distributed environment consisted of workstations currently being used, then it might be more advantageous to have a higher frequency of mappings. It may also be necessary to increase the frequency of mappings, if the task graph is altered as a result of the execution of one of the remote tasks in the task graph. For example, this may be the case if there is a conditional statement in the group of tasks that is based on an output of a remote task in the group (task A). If this conditional statement determines whether another remote task (task B) gets executed then the shape of the task graph depends on the output of task A. When the shape of a task graph is determined by the outputs of a remote task in the group then it is important to increase the frequency of mappings and perform mappings whenever the task graph is altered. To ensure the shape of the task graph is accurate in the

aforementioned case, the task graph should be generated and mapped every time task A is executed.

It is also possible to make this mapping frequency more dynamically adaptive. In Figure 7, the value assigned to the variable *nb_steps* indicates how many evolution steps should be mapped collectively at the next point of execution of the application. This value can be fine-tuned during the execution of the application to determine the optimal number of evolutions to map as a group. In this example, the value for *nb_steps* is updated and fine-tuned using an evaluation function *func()*. This may be a function that changes the value of the variable *nb_steps* based on an evaluation of the performances of previous executions of collective mappings.

```
nb_evolutions=1000;
for(i=0;i<nb_evolutions;i++) {
  gs_smart_map("ex_map"){
    nb_steps=func(...);    //assign dynamically
    for(j=0;j<nb_steps;j++,i++){
      ...
      grpc_call(grav_hndl, phiold, ...);
      grpc_call_async(dark_hndl,&sid_dark,x3dm,.);
      grpc_call_async(bary_hndl,&sid_bary,v3bm,.);
      grpc_wait(sid_dark);
      grpc_wait(sid_bary);
      ...
    }
  }
```

Figure 7. Dynamically determining the optimal group size to map

This approach can be used to find the optimal mapping for an application on any given distributed environment. Once determined, this optimal number can then be assigned statically for each subsequent execution of the application on this environment without the need for an evaluation function.

However, in the case were the environment is highly changeable this optimal number of evolutions may vary throughout the execution of the application and therefore it may be more beneficial to maintain this dynamic update of *nb_steps* variable at run-time.

5. Experimental Results

In this section we compare execution times of the GridSolve and SmartGridSolve implementations of the Hydropad application. The underlying network consisted of three machines: a client and two remote servers, S1 and S2. In terms of performance S1, which performed 140Mflop/s, was slower than S2, which performed 523Mflop/s. The bandwidth from the client to S1 was 1Gb/s while the bandwidth from the client to S2 was 100Mb/s. The bandwidth between both servers was 100Mb/s. This configuration represented a realistic representation of a grid environment in which a client is connected to two remote servers, one that is slow but in close proximity and one which is fast but is at a further distance from the client.

Figure 8 shows a comparison of the performance of the GridSolve and the SmartGridSolve implementation of Hydropad performing six iterations of the evolution step. In this experiment, tasks were mapped in the standard way for GridSolve (i.e. individually) and all six iterations were mapped collectively in SmartGridSolve. Figure 8a shows the difference in execution times for increasing input data sizes. The input data size proportionally increased both the computation and communication load of each remote task. Figure 8b shows the speedup SmartGridSolve achieved over GridSolve for increasing input data sizes. The speedup was consistently above 2 and reached 2.17 when the input data size was above 500MB.

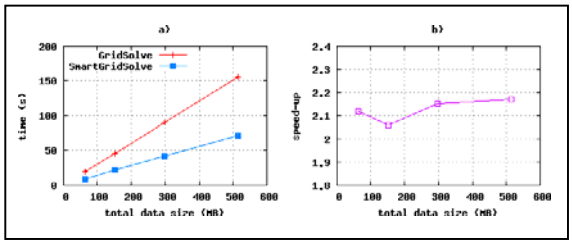

Figure 8. a) Execution times of SmartGridsolve and GridSolve implementations of Hydropad. b) Speedup obtained by SmartGridSolve over GridSolve.

GridSolve based its mapping on each individual task and the performance model of a star network. As a result, it mapped each task on each iteration to the fastest server S2 as this yielded the lowest execution for each individual task. For each task executed, every input was sent to S2 from the client and every output was sent back from S2 to the client, despite there being dependencies between tasks. In addition, this mapping resulted in dark matter and baryonic matter both executing on S2, despite being executed in parallel. That meant that server S2 was heavily load with computation and its client-server link was heavily loaded with communication, which increased the overall execution time of both parallel tasks.

However, SmartGridSolve evaluated the task graph and the performance model of the fully connected network when mapping. Therefore it could eliminate bridge communication by exploiting the dependencies between the tasks and proportionally distribute the

computational load and communication load of parallel tasks over the fully connected network.

As a result, on every iteration it mapped the larger task (baryonic matter) to the faster server (S2) and the smaller task (dark matter) to the slower server (S1). Since the computational load was proportionally distributed over the network, the overall computation time of executing both parallel tasks was decreased. In addition, as a result of this mapping there was improved load balancing of client-server communication, which was distributed over two communication links as opposed to heavily loading a single link. This decreased the overall communication time of both parallel tasks.

To reduce the overall volume of communication, SmartGridSolve eliminated bridge communication by exploiting both the dependencies between tasks in the same evolution step and between tasks of subsequent evolution steps. Figure 9 shows the communication scheme that SmartGridSolve implemented for two evolution steps.

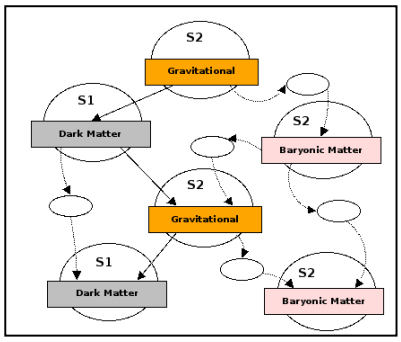

Figure 9. Caching and direct communication between servers for two evolution steps.

The solid line indicates direct communication between servers and the broken line with the ellipses indicates caching. For simplicity the diagram has omitted any communication with the client. On each evolution step, outputs of the gravitational task that were required by the dark matter task were sent directly from S2 to S1. Outputs of the gravitational task that were required by the baryonic matter task were cached locally on S2.

It is evident from Figure 6 that there are also dependencies that existed between subsequent evolutions. Outputs of baryonic matter and dark matter of one evolution step were required by the gravitation task of the next evolution step. Also the outputs of dark matter and baryonic matter of one evolution step were required by the same tasks on the next evolution step. These dependencies between subsequent evolution steps were also exploited, either by direct communication or caching as shown in Figure 9.

Figure 10 illustrates the speedup of SmartGridSolve over GridSolve for mapping increasing number of evolutions collectively when the input data size was 150MB. It is apparent from this graph that the speedup increased sharply from 1 to 2 iterations then there was moderate increase from 2 to 6 and a minimal increase thereafter.

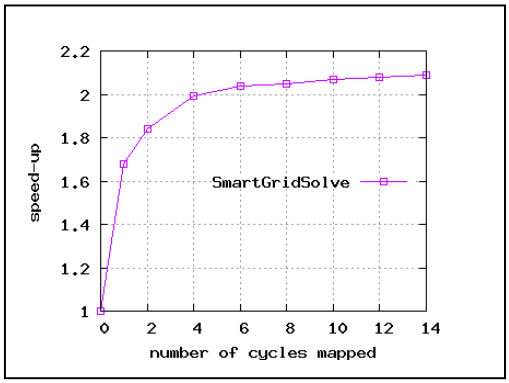

Figure 10. Speedup of SmartGridSolve over GridSolve for increasing number of evolution steps mapped collectively.

The shape of this curve can be explained as follows. When only one evolution was mapped collectively, dependencies between subsequent evolutions could not be exploited. That meant that bridge communication between evolutions was forced and this dependent data was sent between the servers via the client. However, when two evolution steps were mapped collectively then all the dependencies between two evolution steps could be exploited, either through direct communication or caching. Therefore bridge communication between evolution steps occurred after every 2nd evolution step and therefore occurred ½ as often than when one evolution step was mapped collectively. Therefore the speedup of the application increased sharply. When three evolution steps were mapped collectively then all the dependencies between three evolution steps could be exploited. Therefore bridge communication between evolution steps occurred after every 3rd evolution step and therefore occurred 1/3 as often as when one evolution step was mapped collectively but only a 1/6 as often as when two evolution steps were mapped collectively. Therefore there was only a moderate increase in the speedup over GridSolve by increasing the number of evolutions mapped from 2 to 3. This continued until

the speedup from increasing the number of evolutions steps mapped collectively became insignificant.

In Figure 7, we demonstrated the possibility of having a function which could dynamically determine the optimal number of tasks to be mapped collectively for a given application on a given environment. An example for a basic implementation of this function could be a function that dynamically generates a graph of the performance of mappings similar to the one shown in Figure 10. The function could then determine the optimal number of tasks to map by evaluating this graph at any given point in an application's execution.

5. Conclusion

SmartGridSolve's simple and easy to use API provides the functionality for collective mapping of groups of tasks in GridRPC enabled applications on to fully connected networks. The key to the implementation of collective mapping was to separate the mapping of tasks from their execution which is one atomic operation in the GridRPC model of GridSolve. This functionality was achieved with only a minor addition to the GridRPC API. Therefore an application programmer can gain from the improved performance of collective mapping by only making minor changes to any application that is already GridRPC enabled. With the API the application programmer can indicate which tasks get mapped collectively and determine the frequency in which iterative sections of an application get mapped. This gives them the means to use their knowledge of the application and the executing environment to further increase the performance of the application. An approach was also presented that gave the application programmer the flexibility to dynamically update and fine tuned this frequency.

The experimental results presented in this paper show that SmartGridSolve significantly improved the performance of Grid enabled applications. Through collective mapping, the astrophysical application achieved a speedup of approximately 2.17 over GridSolve.

6. References

[1] A. D. Birrell, B. J. Nelson, "Implementing remote procedure calls", *ACM Trans. Comput. Syst.*, ACM, New York, NY, USA, 1984, 2(1) pp 39-59

[2] K. Seymour et al, "An Overview of GridRPC: A Remote Procedure Call API for Grid Computing", *Third International Workshop on Grid Computing*, Springer-Verlag, London UK, 2002, pp. 274-278.

[3] A. YarKhan, K. Seymour, K. Sagi, Z. Shi, and J. Dongarra. "Recent Developments in GridSolve", *International Journal of High Performance Computing Applications*, Sage Science Press, 2006, pp 131-141.

[4] Y. Tanaka, H. Nakada, S. Sekiguchi et al: "Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing", *Journal of Grid Computing*, 2003, pp 41-51.

[5] E. Caron, F. Desprez, "DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid", *International Journal of High Performance Computing Applications*, Sage Science Press, 2006, pp 335-352.

[6] Brady T., Konstantinov E., and Lastovetsky A., "SmartNetSolve: High Level Programming System for High Performance Grid Computing", *IPDPS*, IEEE Computing Society, 2006, pp 8-18.

[7] H. Casanova, J. Dongarra, "NetSolve: A Network Server for Solving Computational Science Problems", *The International Journal of Supercomputer Applications and High Performance Computing*, Sage Science Press, 1997, pp. 212-223.

[8] F. Desprez, E. Jeannot. "Improving the GridRPC Model with Data Persistence and Redistribution", *ISPDC*, IEEE Computing Society, 2004, pp. 193-200.

[9] Y. Tanimura, H. Nakada, Y. Tanaka, S. Sekiguchi. "Design and implementation of distributed task sequencing on gridrpc", *International Conference on Computer and Information Technology*, IEEE Computing Society, 2006, pp. 67.

[10] X. Zuo, A. Lastovetsky, "Experiments with a Software Component Enabling NetSolve with Direct Communications in a Non-Intrusive and Incremental Way", IPDPS, IEEE Computing Society, 2007, pp 1-8.

[11] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, et al., "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems", *J. Parallel Distrib. Comput.*, Academic Press, Inc., 2001, pp 810-837.

[12] C. Gheller, O. Pantano, L. "A cosmological hydrodynamic code based on the piecewise parabolic method", *Monthly Notices of the Royal Astronomical Society*, Blackwell Science, 1998, pp. 519–533