Experiments with a Software Component Enabling NetSolve with Direct Communications in a Non-Intrusive and Incremental Way

Xin Zuo, Alexey Lastovetsky

School of Computer Science and Informatics
University College Dublin
Belfield, Dublin 4, Republic of Ireland
{ Xin.Zuo, Alexey.Lastovetsky }@ucd.ie

Abstract

The paper presents a software component that enables NetSolve with direct communications between servers in a non-intrusive and incremental way. Non-intrusiveness means that the software component is supplementary, working on top of the original system, which does not change at all. Increment means that the software component does not have to be installed on all computers to enable applications with the new feature. It can be done incrementally, step by step, and the new feature will be enabled in part, with the completeness dependent on how many nodes have been upgraded with the software component. The paper describes the design and implementation of the software component. The paper also reports on experiments with three typical scientific NetSolve applications having different communication structures: (i) protein tertiary structure prediction, (ii) image processing using sequential algorithms, and (iii) the matrix chain product. The presented experimental results show that the performance of these Grid applications can be easily and significantly improved by using the proposed supplementary software component.

1 Introduction

High performance Grid programming systems have reached a certain level of maturity. Scientific researchers and programmers can develop reliable Grid applications by using systems such as NetSolve/GridSolve [1-3] and Ninf [4]. The systems are quite easy to install and use. They also demonstrate high level of stability and reliability achieved over years of testing and maintenance. On the other hand, the constantly growing number of users and applications results in the need of further development of such systems in terms of functionality and quality.

Traditionally, the addition of a new feature to a Grid programming system is achieved by changing the code of the system and producing its new version. This new version of the system has to replace the previous one in order to enable Grid applications with the new feature.

In [5], we formulated another approach to enabling an existing Grid system with a new feature. Its main difference from traditional ones is that it is non-intrusive and incremental. *Non-intrusiveness* means that the original system does not change and the new feature is provided by a supplementary software component working on the top of the system. *Increment* means that the software component does not have to be installed on all computers to enable applications with the new feature. It can be done incrementally, and the new feature will be enabled in part. In [5], we demonstrated how this approach could be applied. In particular, we briefly outlined the design and principles of implementation of a software component, which could enable NetSolve with direct communications between remote tasks in a non-intrusive and incremental way. Since then, the software component has been fully implemented and widely used for experiments with various NetSolve applications. In this paper, we present some details of the implementation and results of experiments with three typical real-world applications having different communication structures.

The rest of the paper is structured as follows. Section 2 describes the design and implementation of the software component enabling direct communications in NetSolve. Section 3 presents experiments with the applications. Section 4 outlines related work, and Section 5 concludes the paper.

2 Enabling NetSolve With Direct Communications

NetSolve is positioned as a programming system for high performance distributed computing on global networks based on GridRPC [6]. In NetSolve, output data of remote tasks are typically sent back to the client upon completion of each remote task even if the data are only needed as input for some other remote tasks, resulting in so-called bridge communications when data between remote tasks are sent through the client machine. More generally, bridge communication can be defined as follows. Two separate tasks on the remote servers make the use of a third party (either the client machine or some other machine) to transfer data between the tasks. Compared with direct communication between the tasks, the bridge communication can seriously increase the cost of communication time during the execution of the corresponding Grid application and hence its total execution time.

2.1 Non-Intrusive and Incremental Approach

Traditionally, addition of a new feature to a Grid programming system is achieved by changing the code of the system and producing its new version. This new version of the system has to replace the previous one in order to enable Grid applications with the new feature. This approach to the evolution of Grid programming systems has two serious disadvantages. First, the change of the system's code may introduce bugs resulting in the situation when some applications, which have been developed, tested and successfully executed with the previous version of the system, will not run properly with the new one. Secondly, the new version of the system has to replace the old version on *all* computers of the Grid in order to support the development and execution of applications enabled with the new feature. Such simultaneous and total replacement can have very high organizational overhead and sometimes be simply unrealistic as different computers on the Grid are managed and administered by independent and, very often, loosely connected users.

Our research focuses on developing a *non-intrusive* and *incremental* approach to enabling new features in an existing Grid programming system.

Non-intrusiveness means that the original system does not change and the new features are provided by a supplementary software component working on the top of the system. Correspondingly, all applications not requiring those new features will only use the basic original software and be developed and executed in the same way both in the original and modified systems.

Increment means that the supplementary software component does not have to be installed on all computers to enable applications with the new features. It can be done incrementally, step by step, and the new features will be enabled in part, with the completeness dependent on how many nodes participating in the execution of the application have been upgraded with the supplementary software component.

2.2 Design and Implementation of Software Component

By using the *non-intrusive* and *incremental* approach, we have developed such a supplementary software component that enables direct communications in NetSolve. It consists of three parts: *Client API & Argument Parser, Server Connector* and *Job Name Service (JNS).*

Client API provides a uniform interface for the client to make remote procedure calls. Despite the modification on the remote side, the wrapper API allows the calls to be made in the same manner. The only difference is in the arguments that can be not only variables storing real data but also handlers. Like in NetSolve, we parse the list of arguments to construct the handler array. For each argument, the relevant *communication information* is generated. For each input argument, which is a variable storing real data, the local IP address and the port number are used as such communication information. If this input argument is a handler, then a request is sent to the JNS to get the IP address and the port number of the remote resource and this information is used as communication information for this handler. For each output argument, which is a variable storing real data, the client wrapper function will set up a socket to download output data from computational servers. If this output argument is a handler, the returned result information from computational servers is sent to JNS and registered there.

Server Connector is on the server side, which is a proxy program responsible for interacting with clients and other Server Connectors to enable direct communications. The Server Connector has two main functions. The first one is to pass handler information between clients and servers. This allows servers to know how to get the data without bridge communication. The second function is the extraction of the handlers' information and using it to download needed data through direct communication. After all the needed data have been acquired; the Server Connector calls the procedure to re-submit to the local host to perform computations that the user exactly requested for. There is no difference in the way the client and computational servers download the result of the computations. The Server Connector firstly returns the result's communication information to the client. Then it sets up a socket waiting for the client or the server to connect in to download the result of computations.

Job Name Service (JNS) is responsible for registration of a procedure upon its invocation during RPC call. Other procedures may send requests to the JNS to search for the registered procedure. JNS is set up on the client side automatically. During the execution of the application, it contains all information about every handler. Only the client has the permission to register or access a handler on the JNS. There is no communication and interaction between JNS and computational servers. Because JNS is designed as a system-independent system on the client side, it can be applied to different RPC-based systems and not influenced by any fault or crash on the server side.

3 Application and Experiments

The software component presented in Section 2 has been fully implemented. In order to use it, client programmers have to install the wrapper API and Job Name Service on the client side and then compile the client program with the wrapper library. The wrapper API allows the programmers to explicitly specify the dataflow between remote tasks. They only need to slightly modify their client code. The principle is easy: the programmer just replaces the input/output arguments with handlers as the input/output data. For example, let two tasks, A and B, be performed on remote nodes. Let the output of task A be the input of task B. Normally, the corresponding NetSolve client code would look as follows:

errno=netsl("A", inputA, outputA);
errno=netsl("B", outputA, inputB, outputB);

The extended API allows the programmer to explicitly specify the optimal dataflow as follows:

errno=mynetsl("A", inputA, hdlA);
errno=mynetsl("B", hdlA, inputB, outputB);

Here, the output of A is represented by handler *hdlA* specifying that it is stored on the remote server. In addition, *hdlA* replaces *outputA* as a B's input argument. The use of *hdlA* tells B where it can get this input. The other modification is the use of *mynetsl* instead of *netsl*. This helps distinguish between our wrapper API and the native NetSolve API.

On the server side, the procedure programmers should do nothing to enable direct communications. They develop their procedures as usual. The supplementary software component has no effect on both existing procedures and newly added procedures. To enable direct communication control on the server side, the server administrator needs to install a Server Connector. No re-installation and re-compilation of either NetSolve itself or registered NetSolve procedures are needed.

Next, we present three typical applications with different communication structures and demonstrate the performance improvement achieved due to the use the software component for elimination of bridge communications. Experiments are conducted using six servers, interconnected via a 100 Mbit Ethernet network with a switch enabling parallel communications. The specifications of the servers are shown in Table 1.

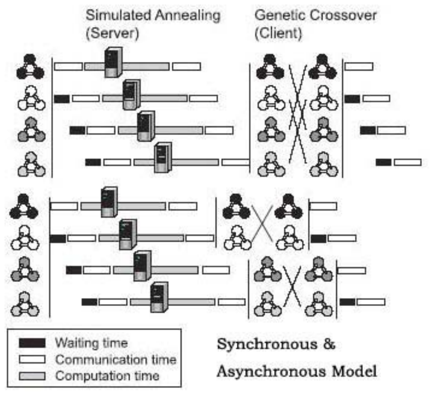
3.1 Genetic Crossover in Protein Tertiary Structure Prediction System

For progress of the bioinformatics, the protein tertiary structure prediction systems are proposed, which is mainly performed by the protein energy minimization. However, a large-scale computing environment would be valuable for this system. In the system, Parallel Simulated Annealing using Genetic Crossover (PSA/GAc) [7] is a minimization engine. To use the Grid resource, NetSolve is a basic tool and implementations are already prepared [8] to improve the computing performance. Their approach reduces critical overhead due to large communication delay over the Internet by using an asynchronous Crossover model. However, bridge communications still exist, and these unnecessary communications can be eliminated by using our software component. Figure 1(a) shows both synchronous and asynchronous Master-slave models for Genetic crossover in the protein tertiary structure prediction system.

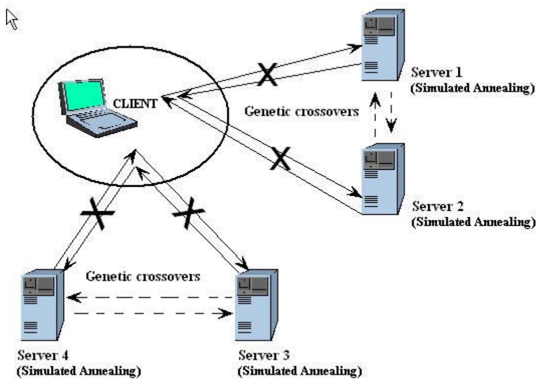
By enabling direct communication using our software, Genetic Crossovers are executed between servers directly. The original approach depends on the client side to do Genetic Crossovers. Direct communication is enabled between Server 1 and Server 2, and between Server 3 and Server 4. Simulated Annealing is executed on each server separately. So the exchanging data is not returned back to the client while direct communications are enabled. This reduces communication links between the client and servers. Figure 1(b) depicts how bridge communications between the client and NetSolve servers are replaced by direct communications between NetSolve servers while performing Genetic crossovers. Eight communication bridges have been eliminated and 4 direct communications have been established. Thus, the total number of communication links is reduced from eight to four by using our software component.

Table 1. Specifications of the six servers

Architecture	CPU MHz	Main Memory (KB)	Cache (KB)
Linux 2.6.9-1.667smp	3200	1038412	1024
Linux 2.4.21-27.0.2.Elsmp	1099	513960	1024
Linux 2.6.11-1.1369_FC4	800	524288	1024
Linux 2.6.11-1.1369_FC4	800	524288	1024
Linux 2.6.11-1.1369_FC4	800	524288	1024
Linux 2.6.11-1.1369_FC4	800	524288	1024


Simulated Annealing (Server)
Genetic Crossover (Client)
Waiting time
Communication time
Computation time
Synchronous & Asynchronous Model

(a)


CLIENT
Genetic crossovers
Server 1 (Simulated Annealing)
Server 2 (Simulated Annealing)
Genetic crossovers
Server 4 (Simulated Annealing)
Server 3 (Simulated Annealing)

(b)

Figure 1. (a) Asynchronous and synchronous models for PSA/GAc. (b) Enabling direct communications between NetSolve servers while performing Genetic crossovers.

Table 2. B – Bridge communication time (in seconds); D – Direct communication time (in seconds).

Protein Size (kb)	Trail 1		Trail 2		Trail 3		Average		Speedup
	B	D	B	D	B	D	B	D	
1000	50	30	51	30	53	31	52	30	45%
2000	106	62	108	63	108	62	107	62	42%
3000	175	98	170	100	178	105	174	101	42%

Table 2 shows experimental results of three trails with different sizes of protein. It gives communications times of the original NetSolve application using bridge communications and the modified application emplyong direct communications. The average communication speedup due to elimination of bridge communications is around 43%.

3.2 Image processing using sequential algorithms

So far, image and video processing software has been predominantly written for conventional (sequential) desktop computers and embedded digital signal processors (DSPs), which implement a wide range of operations [9] such as smoothing, sharpening, noise reduction, etc. These applications usually have a tremendous potential for parallelism but unfortunately, existing techniques are not adequate for compiling sequential multimedia programs to such parallel architectures. Therefore, some researchers focus on extracting the essential computations and data dependency to ensure that each computation has the data it requires [10, 11]. Our research aims to optimize communications of data transaction for sequential multimedia operations. The method is to enable direct communications for sequential image processing by using our supplementary software component. For experiments, we chose an example, which is Simple Linear Combination Filtering [12]. Linear combination filtering functions are taken from Image Processing Library 98 [13].

For image enhancement, linear combination filtering can blur smooth parts of an image while sharpening areas that contain detail. The reason for this combination is that blurring reduces noise, but degrades edges and image detail while sharpening enhances edges and detail but makes noise more visible. Figure 2 displays the example pictures of simple linear combination filtering. Figure 3 depicts how bridge communications between the client and NetSolve servers are replaced by direct communications between NetSolve servers while performing linear combination filtering functions in this case.

To eliminate un-necessary communications between the client and the servers while performing linear combination filtering, we select two servers to perform linear combination filtering functions in parallel:

Server 1:
- Laplacian of image (a);
- Spatially invariant high-pass filtering, sum of image (a) and image (b);

Server 2:
- Mask image, Sobel gradient of image (a) smoothed by a 5x5 box filter;
- Product of image (b) and image (d);
- Space-variant enhancement, sum of image (a) and image (e);

Direct communications are enabled between the servers by transferring image (b) from Server 1 to Server 2 directly.

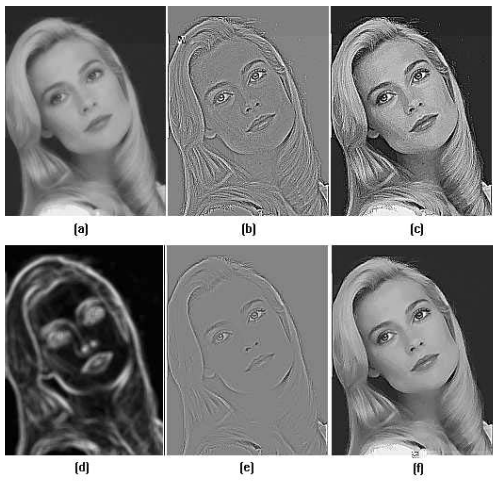


Fig. 2

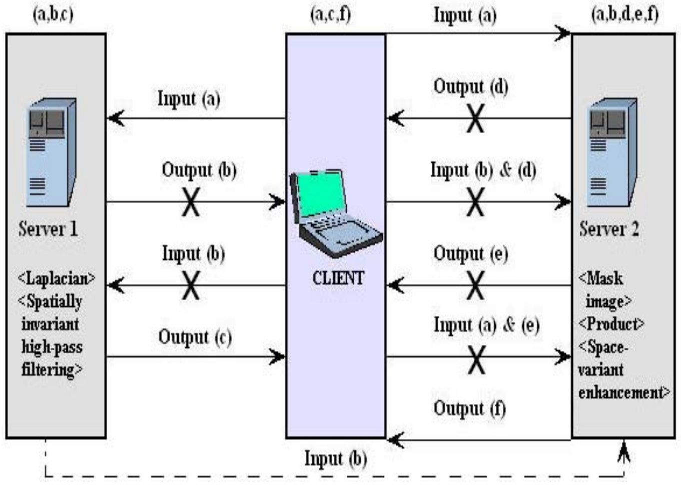


Fig.3

Figure 2. (a) Input image; (b) Laplacian of (a); (c) Spatially invariant high-pass filtering [sum of (a) and (b)]; (d) Mask image [Sobel gradient of (a) smoothed by a 5x5 box filter]; (e) Product of (b) and (d); (f) Space-variant enhancement [sum of (a) and (e)].

Figure 3. Enabling direct communications between NetSolve servers while performing linear combination filtering.

Table 3. B – Bridge communication time (in seconds); D – Direct communication time (in seconds)

Picture Size (kb)	Trail 1		Trail 2		Trail 3		Average		Speedup
	B	D	B	D	B	D	B	D	
1000	60	29	60	29	61	29	60	29	51%
2000	125	61	122	62	125	63	124	62	50%
3000	195	97	209	98	203	98	200	98	51%

Those images, which will be used as input for other image processing functions, will NOT be returned to client. This reduces bridge communications between client and servers. We can see that in Figure 3 communication links are reduced from ten to five by using our software component. Only necessary images (a), (c) and (f) will be on the client side.

We experimented with the linear combination filtering application for different size pictures. Table 3 shows experimental results of three trails with different sizes of picture. The results show that the average communication speedup is around 50%. This is due to the fact that six communication bridges were eliminated and one direct communication was established between two servers.

3.3 Matrix chain product problem in general scientific computations

Given N matrices $A_1, A_2, \ldots, A_n$ of size $N \times N$, the matrix chain product problem is to compute $A_1 \times A_2 \times \ldots \times A_n$. The matrix chain product is an important computational kernel that is used in computing the characteristic polynomial, determinant, rank, and inverse of a matrix, in solving graph theory problems, and in general scientific computations [14, 15].

Figure 4 shows how the product of $A_1, A_2, \ldots, A_8$ can be obtained by using the standard binary tree method. The leaves are input matrices $A_1, A_2, \ldots, A_8$, and the root task of the tree computes the final result $A_{12345678}$. Figure 5 depicts how bridge communications between the client and NetSolve servers are replaced by direct communications between NetSolve servers for matrix chain product computation, where six communication bridges were eliminated among the total fourteen.

In the experiments, we selected four servers to perform matrix chain product computation in parallel:

Server 1: - to perform $A_1 \times A_2$;
Server 2: - to perform $A_3 \times A_4$, $A_{12} \times A_{34}$;
Server 3: - to perform $A_5 \times A_6$;
Server 4: - to perform $A_7 \times A_8$, $A_{56} \times A_{78}$, $A_{1234} \times A_{5678}$;

Direct communications are enabled between these four servers by directly transferring output A_{12} from Server 1 to Server 2, output A_{56} from Server 3 to Server 4, output A_{1234} from Server 2 to Server 4. These output matrices will NOT be returned to the client. This reduces bridge

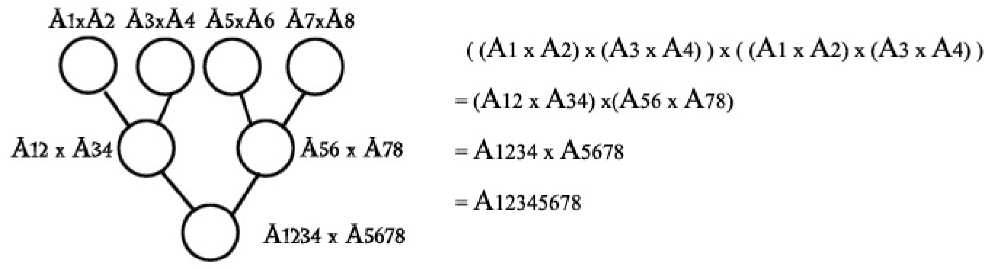


Figure 4. Standard binary tree method used for matrix chain product problem.

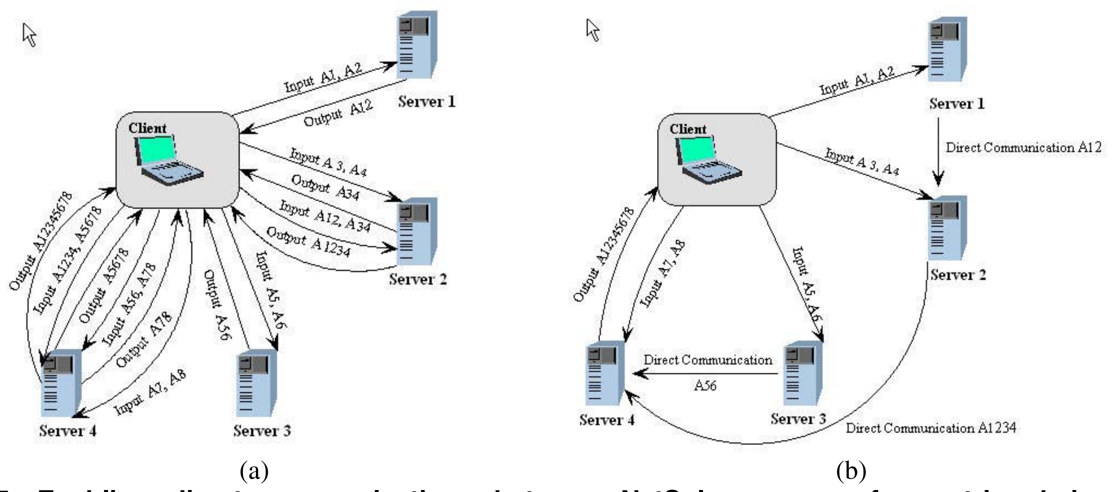


Figure 5. Enabling direct communications between NetSolve servers for matrix chain product computation. (a) bridge communication (b) direct communication.

Table 4. B – Bridge communication time (in seconds); D – Direct communication time (in seconds)

Matrix Size	Trail 1		Trail 2		Trail 3		Average		Speedup
	B	D	B	D	B	D	B	D	
1000	102	66	101	67	103	67	102	67	38%
2000	210	132	220	136	212	138	214	135	36%
3000	335	220	315	226	310	216	320	221	31%

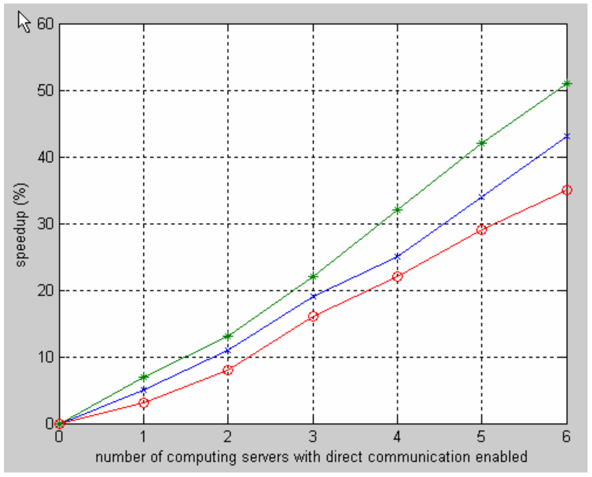
communications between the client and the servers. Communication links are reduced from fourteen to eight. Only the result matrix A12345678 is returned to the client.

We experimented with different matrix sizes. Table 4 shows the experimental results. The average communication speedup is around 35%.

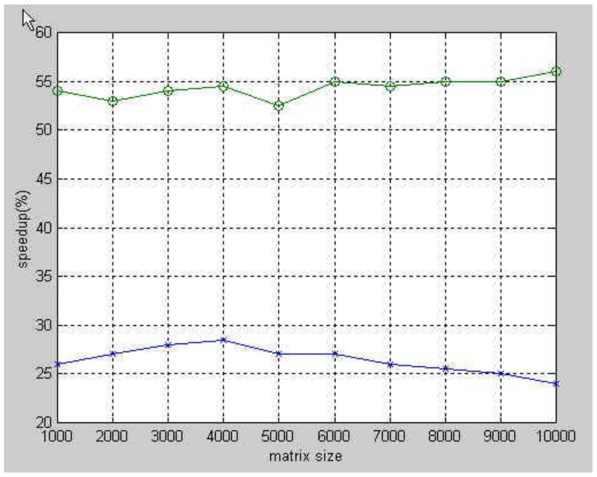
3.4 Other Experiments

One feature of our approach is increment. It means that the supplementary software component does not have to be installed on all computers to enable applications with direct communications. In this case, direct communications can only happen between those computing nodes, where our supplementary software component is installed. Non-enabled computing nodes can only communicate with the client. The speedup of a NetSolve application due to the use of our software component depends on how large is the fraction of computing nodes with enabled direct communications in the overall set of computing nodes used by the application.

In our next experiment, we use six computing servers for computation. We manually changed the number of computing nodes enabled with direct communications. Figure 6(a) shows that the average communication speedup for our three applications is growing linearly while the number of computing servers with direct communication enabled increases from 0 to 6.



(a)



(b)

Fig. 6. (a) Speedup for the three applications increases linearly with the increase of the number of computing servers with direct communication enabled from 0 to six ('o' – the matrix chain product; '.' – Genetic crossover; '*' – Image processing using sequential algorithms). (b) Speedups for the matrix chain product. ('*' – homogeneous network; 'o' – heterogeneous network).

If communication links connecting remote computers are much faster than communication links connecting the remote computers and the client computer, the speedup due to elimination of bridge communications will be much higher. In our next experiment, we manually made all bridge communications to be performed at the speed of 10 Mbit per second. For direct communications between remote servers, we still used 100 Mbit Ethernet interconnecting network. Figure 6(b) presents the average communication speedup of performing matrix chain computations with direct communications on both homogeneous and heterogeneous networks. The experimental speedup for the heterogeneous network is around 54% when the ratio of eliminated bridge communications is 2/9. Thus, much higher speedup can be achieved in heterogeneous communication networks, which are typical for real-life Grid environments, than in artificially designed homogeneous ones.

4 Related Works

To enable direct communications, NetSolve introduces an original mechanism called Request Sequencing [16]. The mechanism imposes a number of restrictions on the sequence of remotely called tasks, the most restrictive of which is that all the tasks have to be performed on the same computing node. Another effort to reduce the overhead of bridge communications in NetSolve is the Logistical Computing and Internetworking (LoCI) [17]. LoCI provides facility to schedule the data storage at a place 'close' to the receiver. The mechanism is mainly aimed at replicating data in order to keep them even in the case of crash of some of the computers. Although it is sufficient for enabling direct communications, the goal of building a complete network storage system makes LoCI over-heavy for enabling just this particular feature.

The REDGRID project [18] is closest to our approach sharing the similar idea behind its design. The main difference is that REDGRID is built into NetSolve and difficult to be migrated to other GridRPC-based systems. The REDGRID project uses an intrusive and non-incremental approach and requires re-compilation and re-installation of the modified NetSolve on all involved computing nodes to enable direct communication. A certain amount of work is needed to port REDGRID to other GridRPC-based systems. Another related project is SmartNetSolve [19], an extension of NetSolve aimed at higher performance of Grid applications, which also enables direct communications but in an intrusive way.

5 Conclusion

In this paper, we have presented the implementation of a software component enabling direct communications in NetSolve in a non-intrusive and incremental way. We have also presented the results of experiments with three typical real-world applications having different communication structures. The experimental results have shown that the performance of NetSolve applications can be significantly and easily improved by using our software component. The software component, the user's guide and the sample applications are freely available at http://hcl.ucd.ie. Future work has been planned to test this software in a larger Grid environment such as Grid Ireland [20].

6 Acknowledgements

This work was supported by the Science Foundation Ireland.

References

[1] http://icl.cs.utkedu/netsolve/

[2] H.Casanova, J.Dongarra. “NetSolve: A Network Server for Solving Computational Science Problems”, The International Journal of Supercomputer Applications and High Performance Computing, 11(3):212-223, 1997.

[3] D. Arnold, H. Casanova, J. Dongarra. “Innovation of the NetSolve Grid Computing System”, Concurrency: Practice and Experience, 14(13-15):1457-1479, 2002.

[4] Tanaka, Y., Nakada, H., Sekiguchi, S., Suzumura, T., Matsuoka, S. “Ninf-G: A reference implementation of RPC-based programming middleware for Grid computing”. Journal of Grid Computing, 1(1): 41-51, 2003.

[5] A. Lastovetsky, X. Zuo, P. Zhao. “A Non-intrusive and Incremental Approach to Enabling Direct Communications in RPC-Based Grid Programming Systems”. Proc, of the 2006 Int’l Conference on Computational Science (ICCS 2006), pp 1008-1011, IEEE Computer Society, 2006.

[6] Seymour, K., Nakada, H., Matsuoka, S., Dongarra, J., Lee, C., Casanova, H. “Overview of GridRPC: A Remote Procedure Call API for Grid Computing”, Proc. of the Third Int’l Workshop on Grid Computing, pp.274–278, 2002.

[7] Hiroyasu T., Miki M., Ogura M, "Parallel Simulated Annealing using Genetic Crossover", Proc. of the IASTED Int’l Conference on Parallel and Distributed Computing Systems, pp.145-150, 2000.

[8] Y. Tanimura, K. Aoi, T. Hiroyasu, M. Miki, Y. Okamamoto and J. Dongarra. “Implementation of Protein Tertiary Structure Prediction System with NetSolve,” Proc, of the 7th Int’l Conference on High Performance Computing and Grid in Asia Pacific Region, pp. 320–327, 2004.

[9] J. Worley, T. Robey, K. Shuldberg. “Image Processing Operations”, Khoral Research, Inc., April 28, 1998.

[10] L. Baumstark Jr., L. Wills. “Exposing data-level parallelism in sequential image processing algorithms”, Proc. Of the 9th Working Conference on Reverse Engineering, pp. 245- 54, 2002.

[11] L. Cordella, A. d'Acierno, C. De Stefano, M. Vento. “Mapping schemes for sequential image processing algorithms”, Proc. of CAMP'95, pp. 184 – 189, 1995.

[12] http://www.adires.com/05/Project/LinCom.shtml

[13] http://www.mip.sdu.dk/ipl98/

[14] K. Li, "Fast and Scalable Parallel Algorithms for Matrix Chain Product and Matrix Powers on Reconfigurable Pipelined Optical Buses", Journal of Information Science and Engineering 18, 713-727, 2002.

[15] K. Li, "Fast and Scalable Parallel Algorithms for Matrix Chain Product and Matrix Powers on Distributed Memory Systems," Procs. of the 15th International Parallel and Distributed Processing Symposium (IPDPS'01), 2001.

[16] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Seymour, K. Sagi, Z. Shi, S. Vadhiyar. “Users’ Guide to NetSolve V1.4.1”. Innovative Computing Dept. Technical Report ICL-UT-02-05, University of Tennessee, Knoxville, 2002.

[17] M. Beck, D. Arnold, A. Bassi, F. Berman, H. Casanova, J. Dongarra, T. Moore, G. Obertelli, J. Plank, M. Swany, S. Vadhiyar, and R. Wolski. “Middleware for the use of storage in communication”, Parallel Computing 28(12), December 2002.

[18] F. Desprez, E. Jeannot. “Improving the gridrpc model with data persistence and redistribution”, Proc of ISPDC 2004/ HeteroPar’04, pp. 193–200, IEEE Computer Society, 2004.

[19] T. Brady, E. Konstantinov, A. Lastovetsky. “SmartNetSolve: High Level Programming System for High Performance Grid Computing”, Proc. of the 20th International Parallel and Distributed Symposium (IPDPS 2006), IEEE Computer Society, 2006

[20] http://www.grid.ie/