A Three Tire Architecture Model for A Computational Grid

J. Karthik Prashanth[1], Pallav Kumar Baruah[2]

Abstract

The past few years have been witness to a growing interest in grid computing as a means of realizing the need for scientific computing capabilities. The grid environment has one severe setback i.e. the requirement of a very steep learning curve. In this work we propose a model, the SMCG (Simple Model for Computational Grid), which is much simpler in comparison to the conventional grid model, for a robust environment that caters to the needs of scientific computation. The major goal of this model is the ease of using this environment for creating applications as well as for programming. This work was motivated by the latest trend in integrating grid computing and cluster computing.

Keywords: SMCG, MPISAI, PARALIB, MPIIMGEN, NOW

1. Introduction

Over the years the scientific community have come to place greater demands on hardware and software infrastructure that provide dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. In response to these demands, there have been many levels of innovation on the supercomputing front.

The past few years have been witness to a growing interest in grid computing as a means of realizing this need for scientific computing capabilities. The grid community defines the grid as a distributed computing environment that operates as a uniform service, which looks after resource management and security management independent of the individual technology choice. Typically the grid infrastructure must include knowledge management resources, an integrator such as OGSA, and the appropriate network to accommodate the various interactions.

The major implementation questions that any new technology, especially one such as the grid, which is proposed to largely replace established and tested technologies, must answer are:

- Will this new technology require a massive learning curve just to employ it?
- Will the user loose significant time from present efforts, implementing this technology?
- Will the transition to this new technology be smooth and painless?

2. The Current Scenario

However, in spite of this burgeoning wide spread acceptance, the grid environment has one severe setback i.e. the requirement of a very steep learning curve. To begin with, the grid administrator is faced with the task of choosing the appropriate grid middleware [Globus Toolkit [1], Sun Grid, Alchemi, and many more] depending on a proper assessment of the future needs of the computing environment. This is often followed by the daunting procedure of configuring the chosen middleware as per the drawn up requirements. Often it

[12]Department of Mathematics and Computer Science Sri Sathya Sai Institute of Higher Learning, Prashanthi Nilayam.

becomes a herculean task to get familiarized with the programming model that is employed. Whole new protocols [3,4,5] like the GSI, GRAM, MDS, GridFTP and others have to be adopted. Adding on to these not-so-favorable conditions, the application programmer must consciously program so as to make effective use of the grid infrastructure. It is quite obvious from these facts that the grid technology, as it stands today, requires a steep learning curve and further demands a greater investment of time and effort. Another very intimidating scenario that we have to come in terms with is the large repository of already existing sequential code, which almost turn unacceptably archaic in the grid environment. For it is widely acknowledged that these sequential codes will have to be rewritten to enable them to make full utilization of the various facilities provided by the grid.

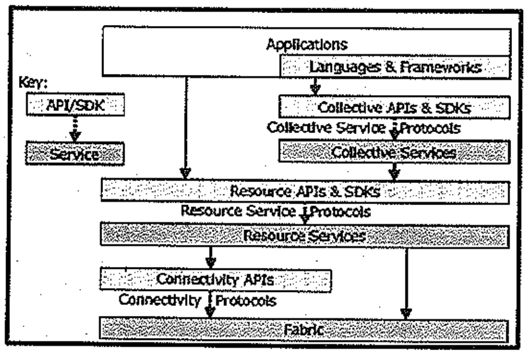

Figure 1. Regular Grid Model

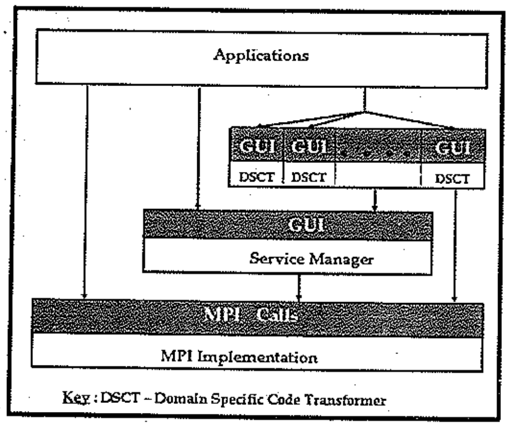

Figure 2. SMCG Model

3. Simple Model for Computational Grid

In this work of ours we propose a much simpler model, in comparison to the grid model, for a robust environment that caters to the needs of scientific computation. This work is inspired by the latest trend in the integration of the grid and cluster computing. The major goal of this model is the ease of using this environment for creating applications as well as for programming. During the course of this paper it will be made explicit how this proposed technology presents itself as a promising solution to the above discussed problems. We begin by making a structural comparison of the two models i.e. the grid and the one we propose, the SMCG model (Simple Model for Computational Grid).

In an earnest effort to simplify and enhance the performance of the computational environment, the SMCG model strive hard to incorporate in it the bare necessities that prove sufficient, from the various popular grid technologies for computational sciences. Just managing to encompass components that aid in making this model effortlessly pervasive, the SMCG model is fundamentally designed for speed and efficiency of the computations involved.

4. The Grid Model Vs SMCG

In figure 1 we have the regular grid model as in [2,4]. Though it offers a great level of flexibility, most of it is not of much use for scientific computing. A very small subset of it would suffice for achieving what is needed. The complexity of the model, though it brings about the flexibility, proves to be a hindrance for a novice. On the other hand SMCG model presents a simpler yet effective solution, calling for no additional knowledge of new protocols. This model is to employ GUIs for the various components, with the intent to be self explanatory and user friendly. They would abstract the underlying architecture providing the end user an easy to use front end. Architecture wise too, SMCG model places very less demand, and hence can be made pervasive with little effort.

Here we present the SMCG model, figure 2, for scientific computing. The lower most layer is any reliable implementation of the MPI standard. This layer will allow the proper utilization of the computing resources. It must be noted that we have not explicitly presented the distributed communication layer on which the MPI would be built, for we believe, given this model the end user would seldom need to access this layer. Moreover most MPI implementations are built over existing distributed technologies like Sockets, CORBA, DCOM, Java, etc.

On top of the MPI implementation layer sits the service manager. It is the role of the service manager to offer the ease of building distributed services and to play the part of an UDDI in web services, to provide the user's application the information on the location of the various services on the network. However this information will be logically abstracted from the end user once the applications are built using the service manager. For example, the application programmer can build a distributed library of the various services offered at the different computing resources, using the service manager. And the end user of this library can access the services in the library without having to know on which node of the network the service is hosted. The service manager also sets up the distributed computing environment i.e. the MPI environment, for parallel execution for any application that is built using the service manager. Thus the end user can write normal sequential programs in which, calls to the various services included in the distributed library can be made, like simple procedural calls. The programmer need not set up the MPI environment or even be acquainted with the knowledge of MPI programming.

In addition to these two layers which by themselves provide good support for the majority of computations, we have introduced the third layer, which is composed of a set of domain specific code transformers(DSCT) with the capability to parallelize domain specific sequential code. It is a well studied fact that constructing a generic parallelizing compiler for any programming language is a humongous task. Since most scientific computations pertain to a specific domain, we can instead build a domain specific code generator. For instance, if we consider a DSCT for the linear algebra domain, then this tool will help us to parallelize any sequential code, that has in it any calls to matrix functions. This layer will throw open a large repository of already existing sequential code to be parallelized. Depending on the domains of the applications handled, the corresponding DSCTs can be employed. Then the sequential codes from this domain can be effortlessly parallelized. Not only can these sequential codes be parallelized, they can be used as stand alone applications or even better, can form a part of a distributed parallel library, which can be created using the service manager.

In any computation oriented environment load balancing and fault tolerance, are two indispensable features. They must be either included in the MPI implementation layer or the service manager layer. Ideally, it would be preferable to include it in the MPI implementation layer. This will offer these features for a wider variety of users.

MPISAI – SMCG

As an instance of the proposed model, we have the MPISAI - SMCG. This tool kit comprises of three main components : MPISAI – a cross platform implementation of MPI [6], PARALIB – a parallel library generation tool and MPIIMGEN – a code transformer for image processing domain[10]. Each one of these three components has been designed to be stand alone parts. The service they offer can be made use of, individually in the absence of the other components. This is a very essential attribute of the component for a modular structuring.

MPISAI

MPISAI [6] is an implementation of MPI – 1.1 standard. Cross platform execution and fault tolerance are key aspects added to this implementation, which is enabled due to its design. The primary goal of MPISAI is to use a heterogeneous cluster. The tool chosen for this, in this regard is DCOM (Distributed Component Object Model).

Components of MPISAI

- MPIRUN : A graphical user interface for setting up the execution environment for any MPI program to run.
- DAEMON : The DCOM object which form the core of the implementation and is responsible for interprocess communication.
- INTERMEDEATE : A library, linked statically to the user process, which acts as an interface between the user process and the daemon

PARALIB : The Service Manager

Developing a library of sequential codes is not a very difficult task. But Building even a modest parallel library does not present itself to be an easy undertaking. Going one step ahead, to develop a library of parallel codes written in MPI and to incorporate the facility that the library routines may be called from an ordinary sequential C/C++ code, calls for a tremendous amount of work. To make one such library the MPI programmer not only has to develop the library but also must dwell deep into the underlying implementation details. Understanding the code of any MPI implementation, if it is available, is a very tedious job for any parallel library developer. To overcome this difficulty of a parallel library creator, "PARALIB" has been developed over MPISAI.

PARALIB is a GUI based tool that helps the library creator in his/her endeavor, keeping the implementation details transparent. It is a resourceful generation tool that assists any MPI programmer to build his/her own parallel library over MPISAI. It provides a user friendly interface to help in building the library. The motivation for developing such a tool is as follows:

- A conventional sequential C/C++ programmer must have native access to a library of parallel routines, for use in his/her code i.e. the programmer must be able to make use of these parallel routines, that are part of the parallel library, as simple function call in his/her programs.
- A large onus lies on the library creator to understand the implementation of the underlying MPI protocol. The library creator must take care of setting up of the parallel execution environment, thus providing an abstraction of a sequential environment for the end user of the library.
- As mentioned earlier, an effortless and efficient means of creating a library of parallel routines is needed. However, a library on distributed environment to be effectively made use of, must itself be distributed across the nodes of the environment. If the parallel library generator could further include this feature of building a distributed library, it would be much appreciated.

Often in a NOW, the routines for creating such a library would already exist, on the various nodes that constitute the network. It would save great effort on the part of the library creator if the library generation tool could provide for including these routines in the new distributed library it creates.

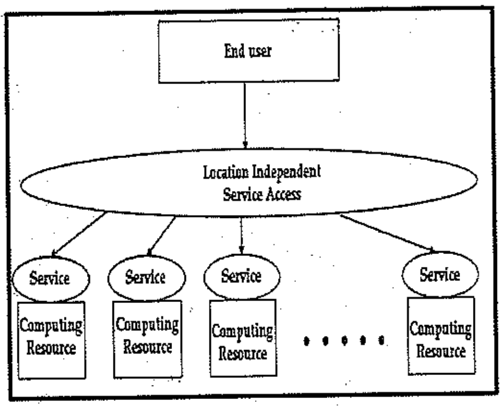

Figure 3. MPISAI-SMCG Environment.

The Parallel Library Auto Generation Tool - "PARALIB" includes all the above desired features and in addition provides a user friendly GUI for the library creator. These features make PARALIB an efficient service manager in the computational grid MPISAI-SMCG.

MPIIMGEN : The DSCT

Most of the image processing operations are highly computation intensive. Not much effort has been directed in adopting a parallel approach, for these applications often involve complex sequential algorithms whose parallelization is found intimidating. However, they have a tremendous potential for parallelism. As an answer to this problem, we have described a code transformer, that is built using the pattern driven approach [7,8] to substitute parts of the sequential code that are identified to be bottle necks for faster computations, with calls to their parallel counter part that are themselves part of a parallel library.

MPIIMGEN[10] is a tool that can automatically replace these sequential image processing programs by equivalent, efficient parallel programs, from a library, that are capable of running on a cluster of workstations. This tool uses a pattern driven approach to parallelize the sequential codes.

Any sequential image processing code can be converted into its corresponding parallel version using MPIIMGEN tool and can be added to the parallel library using the PARALIB tool. Thus, a distributed parallel library of image processing routines can be built. This can be used by the service provider and made available to be invoked by the application developers.

MPISAI-SMCG ENVIRONMENT

The features that are predominantly looked for in any environment that provides for scientific computations are dependability, consistency, pervasive and inexpensive access to high-end computational capabilities. The MPISAI tool kit provides this and much more on any LAN, WAN or even across the Internet. The ease of programming, when using our tool kit is an added incentive for the end user. Unlike the conventional grid, there are no new protocols to come in terms with. Since, most of the scientific environment is over a private scalable network of well identified workstations, we can often do with minimum security setups. What on the other hand is of greater importance is a fault tolerant and load balanced setup for computation intensive operations that need to run for long periods of time.

In the MPISAI environment, the Daemon running on the individual nodes of the cluster, provide the distributed scientific setup. The huge wealth of already existing sequential code can be easily and efficiently converted into their parallel counter parts. These can then be used to construct a distributed parallel library making use of PARALIB, the parallel library generator. Once this is achieved, a user who has little or no expertise in parallel programming can write applications that can execute parallely, making dexterous use of the cluster employed. Thus conventional programmers can, without compromising their never

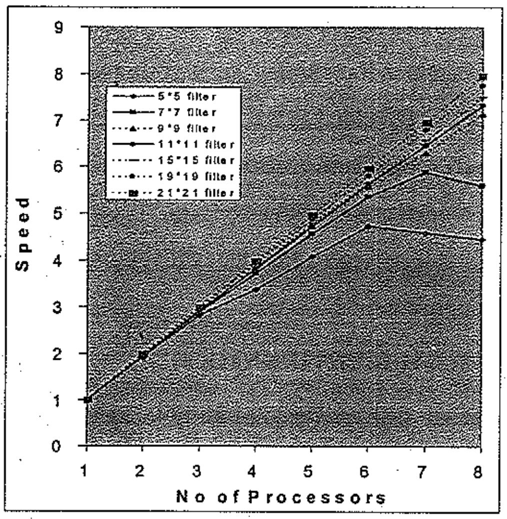

Figure 4. Graph for Median Operation.

programming environment, make adroit use of the nodes of the network. This indeed presents itself as an attractive alternative for many of the recent cluster solutions, from which the end user has to make his/her choice.

5. Results

We created a parallel image processing library using the PARALIB tool. It has the following four components:

- Sequential Image Processing Operations : This component contains a large set of sequential operations typically used in image processing.
- Parallel Extension: This component consists of routines that introduce parallelism into the library and are implemented using the MPI 1.1 standard. These routines can be classified into two classes namely, routines for image distribution and redistribution and routines for overlapping communications.
- Parallel Image Processing Operations: To reduce the code redundancy as much as possible, much of the code for the sequential generic algorithms is reused in the implementation of their respective parallel counterparts. To that end, for each generic algorithm a parallelizable pattern is defined. Each such pattern represents the maximum amount of work in a generic algorithm that can be performed both sequentially and in parallel. In the latter case without having to communicate to obtain non-local data. All the parallel image processing operations follow the master-slave paradigm. This paradigm is implemented using the SPMD approach provided by the MPI standard.
- Single Uniform API: The parallel library is provided with an application programming interface that is similar to that of a sequential image processing library. The only parallel feature that the user needs to specify in this API is the number of processes to run the operation on.

Each of these library routines can be invoked as a service by the application programmer using the grid resources.

Performance Results Of Paralib

This section gives a comparison of parallel vs sequential algorithms implemented in PARALIB. The machine configurations of the nodes in the cluster of workstations, on which the parallel programs were tested are as follows:

- Each node in the cluster is a 2.4ghz intel processor with 512MB RAM.
- The LAN speed was 100mbps with a 10/100 Ethernet switch .
- Median Operation- The timing analysis was done for a 1024*1024 image with different filter sizes. All the algorithms assume that the images were read from files. The Speedup obtained is also shown in Figure 4.

Performance Results Of Mpiimgen

The MPIIMGEN tool has been tested for various programs and it gives good performance results. For eg consider the program with the following operations

- Histogram equalization on a 256*256 image.
- Sum of two 256*256 images.
- A vertical sobel filter of size 3*3 on a 256*256 image.

- A median filter of size 27*27 on a 1024*1024 image.
- A template matching operation of an image of size 128*128 in an image of size 1024*1024.
- A morphological operation dilate on a 256*256 image.
- A translate operation on a 256*256 image.

The time taken by MPIIMGEN for converting this program into a parallel program is 4.12 seconds. The timing analysis for the generated parallel program on a cluster of workstations is shown in Table 1.

Image Size	No of Nodes	5*5 filter (in secs)	Speed up 5*5
1024*1024	1	7.015	1
1024*1024	2	3.64	1.92
1024*1024	3	2.484	2.82
1024*1024	4	2.078	3.37
1024*1024	5	1.723	4.07
1024*1024	6	1.486	4.72
1024*1024	7	1.53	4.57
1024*1024	8	1.579	4.44

Table 1. Results of MPIIMGEN

Conclusion

We have explained our proposed SMCG (Simple Model for Computational Grids). As an implementation of this model, we have described the MPISAI – SMCG that comprises of the three components, MPISAI, PARALIB and MPIIMGEN. In an effort to overcome the difficulty in constructing an efficient parallelizing compiler, we have introduced the idea of domain specific code transformers. MPIIMGEN is an instance of this for image processing applications. We would also like to make a mention of our on going efforts. We are extending MPISAI, our implementation of the MPI standard, to include MPI – 2. We have also found much encouragement in an endeavor to build a generic parallelizing compiler that can in due course of time enhance the DSCT layer. We believe, that SMCG offers an efficient and user friendly environment for the end user.

Acknowledgement: Authors dedicate this work to the Chancellor of Sri Sathys Sai Institute of Higher Learning, Bhagawan Sri Sathya Sai Baba.

6. References

[1] Foster, I. and Kesselman, C., *"Globus: A Toolkit-Based Grid Architecture"*. In *The Grid: Blueprint for a New Computing Infrastructure*, (Foster, I. and Kesselman, C. eds.) Morgan Kaufmann, 1999, 259-278.

[2] Foster, I., Kesselman, C. and Tuecke, S., *"The Anatomy of the Grid: Enabling Scalable Virtual Organizations". International Journal of High Performance Computing Applications*, *15* (3). 200-222. 2001.

[3] Foster, I. and Kesselman, C. (eds.). *"The Grid: Blueprint for a New Computing Infrastructure."* Morgan Kaufmann, 1999.

[4] I. Foster, C. Kesselman, J. Nick, S. Tuecke, *"The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Open Grid Service Infrastructure WG"*, Global Grid Forum, June 22, 2002.

[5] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Meder, S. Tuecke, GridFTP Protocol Specification, *GGF GridFTP Working Group Document,* September 2002.

[6] Bhaskaran .V, Vijay Krishna .P, Sai Swaminathan. G and Baruah P.K. , *"Design and implementation of MPISAI."*, Web Proceedings, HiPC 2003, Hyderabad, 2003.

[7] Christoph W Kessler, *"Pattern-Driven Automatic program transformation and Paralleization"*, **IEEE** 3rd Euromicro Workshop on Parallel and Distributed Processing.

[8] B. Di Martino and G Ianello, *"PAP Recognizer: A Tool for Automatic Recognition of Parallelizable patterns"*, IEEE 4th International Workshop on Program Comprehension (WPC '96), 1996, p.164.

[9] S. Bhansali, J.R. Hagemeister, *"A Pattern-matching Approach for Reusing Software Libraries in Parallel Systems"*.

[10] Vinod Verma and Pallav Kumar Baruah, *"MPIIMGEN-A code transformer that parallelizes Image processing codes to run on a cluster of workstations"* Proceedings of IEEE CLUSTER 2004,San Diago California USA.

Author's Biography

Karthik Prashanth J completed his Bachelors and Masters in Mathematics from Sri Sathya Sai Institute of Higher learning, Prasanthinilayam, INDIA. He then went on to do his Masters in Technology in Computer Science from the same University, in the year 2005. His current areas of interest are parallel and grid computing.

Dr.P.K.Baruah completed his Ph.D. in Mathematics from Sri Sathya Sai Institute of Higher learning, Prasanthinilayam, INDIA, in the year 1994 and is serving in the department of Mathematics and Computer Science since then. He has published number of articles in international journals. Current areas of research interest includes High Performance Computing.