A Parallel Programming Environment on Grid

Weiqin Tong, Jingbo Ding, and Lizhi Cai

School of Computer Engineering and Science,
Shanghai University, Shanghai 200072, China
wqtong@mail.shu.edu.cn

Abstract. Computational grids are environment that provides the ability to exploit diverse, geographically distributed resources. Bulk Synchronous Parallel (BSP) model is a widely used parallel programming model. With rapid development of grid technologies, users need a new environment that can run parallel programs on computational grids. We present here a new implementation of BSP, which is called BSP-G. It constructs a parallel programming environment on computational grids. In our BSP-G environment, users can develop parallel programs with using BSP model and run them on grid. Our BSP-G library uses services provided by the Globus Toolkit for authentication, authorization, resource allocation, executable staging, and I/O, as well as for process creation, monitoring, and control.

1 Introduction

There are many different parallel programming environment available today for a variety of parallel architectures and models. Examples include PVM [22], MPI [23] and BSP [1, 3, 4]. They are widely used in scientific computational field for their respective characteristics. The BSP, Bulk Synchronous Parallel, has advantages relative to others such as prediction of performance and avoidance of deadlock.

Computational Grid [10], which focuses on large-scale resources sharing, provides protocols and tools to construct an integrated virtual supercomputer that is geographically distributed on different sites.

Our BSP-G is an implementation of BSPlib, which is based on the well-known grid middleware Globus Toolkit [11] that is developed by Globus project. It provides a programming library for portability of parallel programs.

In this paper, we propose a strategy for designing the BSP-G and some details of implementation. The rest of the paper is organized as following. In the next two sections, we briefly review the grid computing technologies and BSP model. In the subsequent sections, we first outline the BSP-G model. Then describe the some details about the design and implementation of BSP-G. Results of performance are presented in section 7. We conclude with a discussion of some future directions in Section 8.

2 Grid Computing

The term "Grid" denotes a proposed distributed computing infrastructure for advanced science and engineering. The goal of the Grid is to construct a Virtual Organization (VO) that can share various resources. A layered Grid architecture was proposed to identify fundamental system components, specifying the purpose and functions of these components [9].

Globus Toolkit is a collection of software components designed to implement the protocols of Grid architecture. The detailed description of these components can be seen in [12, 25]. A lot of Grid services are used to construct the BSP-G architecture. Open Grid Service Architecture (OGSA) [25] will be a new core infrastructure of grid. It provides a grid service in order to make grid environment transparent to grid application developers.

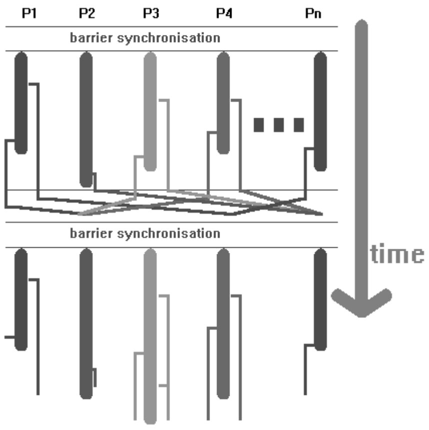


Fig. 1. Superstep of BSP program

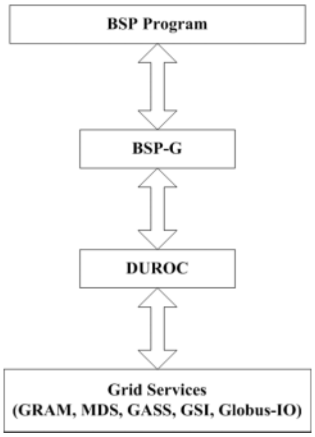


Fig. 2. BSP-G Architecture

3 BSP Model

The Bulk Synchronous Parallel (BSP) model is a generalization of the widely researched PRAM model and was initially proposed by G. Valiant as a Bridging Model for Parallel Computation [1, 2]. Much work on BSP algorithms, architectures and languages has demonstrated convincingly that BSP provides a robust model for parallel computation, which offers the prospect of both scalable parallel performance and architecture independent parallel software [24].

BSP programs have both a horizontal structure and a vertical structure. The horizontal structure arises from concurrency, and consists of a fixed number of virtual processes. These processes are not regarded as having a particular linear order, and may be mapped to processors in any way.

As Fig. 1 shows, the vertical structure arises from the progress of a computation through time. For BSP programs, this is a sequential composition of global superstep, which conceptually occupies the full width of the executing architecture. Each superstep consists of three phases as following:

Computation in each processor, using only locally held values;

Global message transmission from each processor to any set of the others;

Barrier synchronization.

Each end of barrier synchronization is the start of next superstep. It iterates during the execution of BSP program.

A heterogeneous Bulk Synchronous Parallel (HBSP)[19] model is a generalization of the BSP model [2] of parallel computation. HBSP provides parameters that allow the user to tailor the model to the required system. As a result, HBSP can guide the development of applications of traditional parallel systems, heterogeneous clusters, the Internet, and the computational Grid [17].

4 Previous and Related Works

4.1 Other Implementations of BSP

Oxford Parallel is building a BSP Programming Environment which includes implementations of the internationally agreed standards for BSP primitive functions [4] for a wide range of parallel machines together with parallel performance profilers & analyzers, support for debugging parallel programs and benchmarking tools.

The Oxford BSP library, which is developed by Miller, was the first BSP library [3, 4]. It contains basic functions, supporting both BSMP (Bulk Synchronous Message Passing) operations and DRMA (Direct Remote Memory Access) operations. It also supports a lot of architectures and communications devices. In the new version, implementations of BSPlib enable a homogeneous cluster of workstations to be used as a parallel machine. But it cannot adapt to the Grid computing environment.

PUB (the Paderborn University BSP) library [14] is a comprehensive and high performance one. It presents a BSP object concept. It supports the use of threads and provides rich thread functions, but one single BSP object and its subgroup must be used in a same thread, thus may not be easy for a user to program in such a way. Though it provides the process migration mechanism, the migration capability cannot be make full use because of too many limitations [20].

SHUBSP [15] is a BSP library that is designed to improve the computing performance of SMP cluster by Shanghai University BSP research group. It can automatically create the suitable threads rather than processes in the same SMP node when running a BSP program. Because a thread possesses fewer resources than a process, and the communication between threads is more effective than communication between processes through sockets, SHUBSP can achieve high performance in SMP cluster.

xBSP [5], is a implementation of BSP programming library for VIA (Virtual Interface Architecture)[21]. xBSP demonstrates that BSPlib is more appropriate than MPI to exploit the features of VIA. This library can also achieved similar application performance to the native performance from VIPL (the Virtual Interface Provider Layer, an interface provided by VIA), by reducing the overheads associated with multithreading, memory registration, and flow-control.

4.2 MPICH-G

MPICH-G2 [6] is a second-generation version of earlier MPICH-G, which is a Grid-enabled implementation of MPI. It allows users to run MPI programs across grid nodes, which is at the same or different sites; use the same commands as that used on a parallel computer. It extends MPICH-G by incorporating faster communications and quality of service.

5 Motivation of BSP-G Model

All the BSP implementations forenamed in Section 3.1 focus on the performance of the BSP for specified device or platform. The heterogeneous and distributed computing capability is not addressed. But the development of requirement for computing resources in the science and engineering field are seeking this capability. The wide applications of MPICH-G2 have proved this opinion. On the other hand, the advantages of BSP model and portability of existing BSP programs lead us to develop the BSP-G.

Globus toolkit provides lots of core interfaces and services, which intend to construct higher-level policy components. Even though the Globus toolkit provides tremendous APIs or SDKs for programmer to develop applications directly, it is still necessary to build parallel programming libraries on grid. BSP has, compared with various other parallel programming environments such as PVM and MPI, two major advantages:

Most message-passing libraries, such as PVM and MPI, are based on pairwise send-receive operations, which are likely to cause deadlocks. Deadlocks do not occur in a BSP program, which is partitioned into phases or supersteps, because explicit send and receive operations are no longer necessary.

BSP program's correctness and time complexity of the program can be predicted while others cannot.

The sharing requirement for grid computational resources and the advantages of BSP model motivates use to develop grid-enabled BSP library. Our new implementation of BSPlib for grid is called grid-enabled BSP (BSP-G). To the best of our knowledge, BSP-G is the first implementation of BSPlib for grid.

Our BSP-G utilizes Globus Toolkit services to support efficient and transparent execution in heterogeneous grid environments. As Fig. 2 shows, BSP-G first uses DUROC (Dynamically-Updated Request Online Coallocator) [7], which is a component of grid services, to specify where to create processes of a BSP program. The following steps, include authentication, resources allocation and startup, are based on many grid services such as GRAM (Grid Resource Allocation Manager) [13], MDS (Metacomputing Directory Service) [26, 27], GASS (Global Access to Secondary Storage) [28], GSI (Grid Security Infrastructure) [29], Globus-IO [30].

6 Implementation of BSP-G

6.1 User Interface

It's necessary for users to create the suitable Grid computational environment before startup of a BSP program. It includes client, server, SDK bundles of the three Globus Toolkit 2.0 components. User should obtain a user certificate through *grid-cert-request* command, and run *grid-proxy-init* command to authenticate the user to each remote site.

Before using *bsprun* to run a BSP program, BSP-G needs a host file, which includes name of computational nodes. Unlike other implementation such as Oxford BSPlib, the nodes listed in the host file do not mean the exact execution nodes, but just specify all nodes, which can be exploited.

When a user wants to execute a program via *bsprun*, the number of computational nodes should be specified through *–p* parameter. In nonpure SPMD mode, the *bsprun* shell script will enquire the load of all the computational nodes via MDS (Monitor and Discovery Service). The computational resource whose workload is lightest will be written in the Resource Specification Language (RSL) file [16]. The environment variable MASTERPROCESS will also be written in the RSL file. Then *bsprun* will execute the *globusrun* to submit the master process. When the master process calls *bsp_begin()*; the number of processes will be specified. The computation nodes, which have the lightest workload, will be selected. The remaining process will be started via a co-allocation library distributed with the Globus Toolkit, the DUROC [7] control library.

The DUROC library itself uses GRAM [13] API and protocol to start and subsequently manage a set of subcomputations, one for each site. For each subcomputation, DUROC generates a GRAM request to a remote GRAM server who authenticates the user, performs local authorization, and then interacts with local scheduler to initiate the computation.

When user specifies the GRAM RSL parameters executable with GASS URL variable $(GLOBUSRUN_GASS_URL), and run *globusrun* with *–s* parameter, GRAM will use GASS to stage executables from remote locations (indicated by URLs). In the same way, when user specifies the parameters directory, stderr, stdout with a GASS URL. GASS is also used, once an application has started, to direct standard output and error (stdout and stderr) stream to the user's terminal, and to provide access to file regardless of location. This masks essentially all aspects of geographical distribution except those associated with performance.

6.2 Creation of Communication Channels

In this section, we will discuss the communication between processes. Every process obtains a port using function call *globus_io_tcp_create_listener()*. The port and hostname will be exchanged in all the processes. After having got the port and hostname, every process will create *nprocs-1* communication handles or to control the status of all subjobs. Every process listens and accepts any process whose pid is smaller; connects any process whose pid is larger than itself. When a process listens the handle as server, it does not know from which client the connection comes. So when the connection is created, the client first sends its pid to server. Then server judges the client from the first message it received. The following shows the pseudo code of the creation of communication channels.

```
for(pid=0;pid<nprocs;pid++){
  if(pid<mypid){
    globus_io_tcp_listen(&listener_handle);
    globus_io_tcp_accept(&listener_handle, &attr,
      &conn_handle);
    globus_io_read(&conn_handle, (globus_byte_t
      *)&recvbuf, nbytes, nbytes, &nbytes_read);
    position=recvbuf;
  }
  else if(pid>mypid){
    globus_io_tcp_connect(host_port_array[pid].hostname,
      host_port_array[pid].port, &attr, &conn_handle);
    globus_io_write(&conn_handle, (globus_byte_t *)&mypid,
      nbytes, &nbytes_written);
    position=pid;
  }
  handle_array[position].bsp_pid=position;
  handle_array[position].handle=conn_handle;
}
```

6.3 Barrier Synchronization of a Superstep

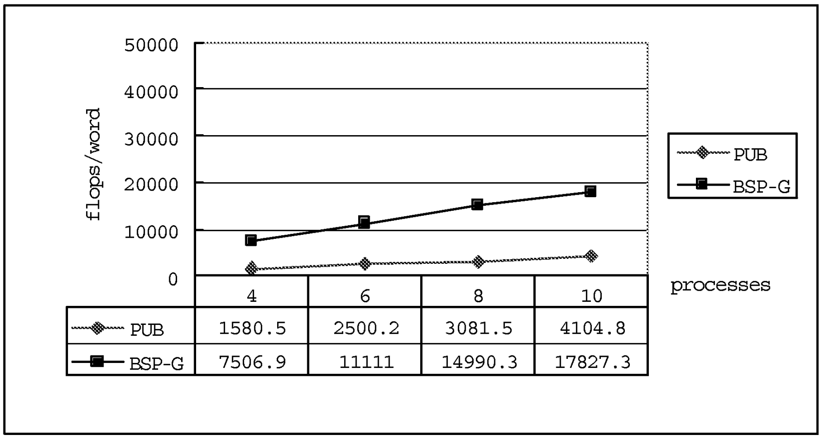
In a Direct Remote Memory Access (DRMA) mode, the memory block should be registered via *bsp_push_reg* function before operation. Users should create a map relation for the same variable in different processes. BSP-G uses a register table to record the relation. The same variable in the different processes have the same ID. In an operation of DRMA, the ID will be integrated in the message and will be sent to the corresponding process by the *bsp_syc()*. The peer of DRMA can find the actual local address according to the ID.

Processes are partitioned into several independent subsets – subgroups. The advantages of introducing subgroup can be seen in [14]. Users submit a job through DUROC, DUROC will allocate the subjobs to the different GRAMs. Generally speaking, the processes within a subjob are in the same LAN. The communication within a subjob is faster than that between different subjobs. A subjob can be considered a natural subgroup.

In BSP model, all communications, including DRMA and BSMP, do not take place until *bsp_sync()*. When a program executes a DRMA or BSMP operation, it only copies the data into send queue except high performance operations. The *bsp_sync()* function should be called by all the processes in the job. All the processes first get the destination pid from the schedule matrix (will be discussed in Section 6.4) which also determines the process should send or receive messages first in a communication. A *bsp_get()* operation reaches into the local memory of another process and copies previously registered remote data held there into a data structure in the local memory of the process that initiated it. So to complete a *bps_get()* operations needs two communications. In the first communication, the initiator sends a message to tell the peer that it wants to get data from the registered memory. The size of the message is very small; it can be piggybacked in the other message. In the second communication, the peer sends the data back. A process which first sends messages should experiences three steps: send-receive-send, but a process which first receives messages should experiences three steps: receive-send-receive, in a complete communication. If the communications take place within a subgroup, we should select the sub synchronization rather than global synchronization to improve the performance.

7 Performance Results

To quantify the performance of the BSP-G library, we have designed some small experiments on a pairs of PC. Each PC has dual Pentium III 550MHz processors with 128Mbyte RAM connected via 100Mbps Ethernet, running Redhat Linux 7.2 SMP version. The BSP-G was built using a nonthreaded, no-debug Globus toolkit 2.0. The BSP library we use for reference is Pub 7.0.



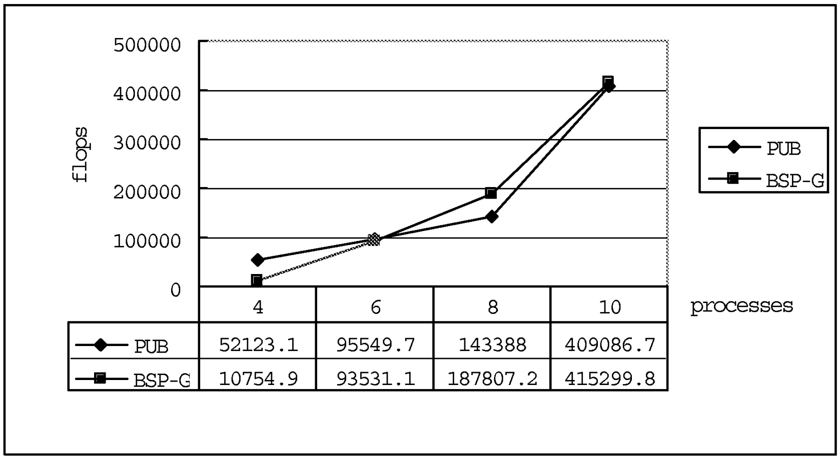
	4	6	8	10
PUB	1580.5	2500.2	3081.5	4104.8
BSP-G	7506.9	11111	14990.3	17827.3

Fig. 3. BSP-G Performance with parameter g

The BSP model simplifies a parallel machine by three components, a set of processors, an interconnection network, and a barrier synchronizer, which are parameterized as $\{p, s, g, l\}$. Parameter p represents the number of processors; s is the rate at which computation can be performed by each processor. The parameter g is the permeability of the communications system to continuous traffic between uniformly random destinations. l is the time required for the barrier synchronization. The time required for a superstep is given by:

$Time\ for\ superstep_i = max_i(s*w_i)+maxi(h_i*g)+l$

Where h_i is the size of the h-relation realized in this superstep by process i.



	4	6	8	10
PUB	52123.1	95549.7	143388	409086.7
BSP-G	10754.9	93531.1	187807.2	415299.8

Fig. 4. BSP-G Performance with parameter l

These parameters are obtained with the BSP benchmark program in the package of BSPEDUpack provided by Rob Bisseling [18], a library of numerical algorithms written in C language according to the BSP model, using the BSPlib standard library. The benchmark program tries to expose the worst behavior of a system. The performance is expected to be better in real applications.

Fig. 3 and Fig. 4 shows the performance of BSP programs. As seen from these figures, performance of BSP-G library is close to that of PUB library.

8 Conclusions and Future Research

In this paper, we have presented an implementation of Grid-enabled BSP called BSP-G. BSP-G exploits the services provided by Globus toolkit 2.0 to enable and hide the heterogeneity of different computation resource. Users can run a BSP program on the Grid but do not need to care abort the detail the resource. The benchmark shows the performance of our implementation is very close to that of PUB library.

Our BSP-G will be applied in a project - complex material molecular simulating grid, which is taken cooperatively by Shanghai University and East China University of Science and Technology.

In the future, we expect to further our research on the following directions:

Build our BSP-G on future Globus Toolkit 3.0, which will be based on a new core infrastructure compliant with the Open Grid Service Architecture (OGSA) [25].

Implement BSP-G based on MPI-G [6] so that MPI programs and BSP programs can be developed and run in a unified environment.

References

1. D. Skillicorn, J. M. D. Hill, W. F. McColl: Questions and Answers about BSP. Scientific Programming, vol. 6(3) (1997) 249–274
2. L. G. Valiant: A bridging Model for Parallel Computation. Communications of the ACM, vol. 33(8) (1990) 103–111
3. Richard Miller: A Library for Bulk Synchronous Parallel Programming. Processing of the BCS Parallel Processing Specialist Group workshop on General Purpose Parallel Computing (1993) 100–108
4. Jonathan M. D. Hill, Bill McColl, Dan C. Stefanescu, Mark W. Goudreau, Kevin Lang, Satish B. Rao, Torsten Suel, Thanasis Tsantilas, Rob Bisseling: Standard: BSPlib: The BSP Programming Library. Parallel Computing, vol. 24 (1998) 1947–1980
5. Y. Kee, S. Ha: An Efficient Implementation of the BSP Programming Library for VIA. Parallel Processing Letters, vol.12, No.1 (2002) 65–77
6. http://www-fp.mcs.anl.gov/division/publications/abstracts/abstracts02.htm
7. K. Czajkowski, I. Foster, C. Kesselman: Co-allocation Service for Computational Grids. Proc 8th IEEE Symp, On High Performance Distributed Computing, IEEE Computer Society Press (1999)
8. Stephen R. Donaldson, Jonathan M. D. Hill, David B. Skillicorn: BSP Cluster: High Performance, Reliable and Very Low Cost. Parallel computing, vol. 26 (2000) 199-2424
9. I. Foster, C. Kesselman, S. Tuecke: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, vol. 15(3) (2001) 200–222
10. I. Foster, C. Kesselman: The Grid: Blue Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers. (1999)
11. I. Foster, C. Kesselman: Globus: A Metacomputing Infrastructure Toolkit. International Journal of Supercomputer Application (1997)
12. http://www.globus.org/gt2/admin/guide-overview.html
13. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke: A Resource Management Architecture for Metacomputing Systems. Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing (1998) 62–82
14. O. Bonorden, B. Juurlink, I. von Otte, I. Rieping: The Paderborn University BSP (PUB) Library - Design, Implementation and Performance. Proc. of 13th International Parallel Processing Symposium & 10th Symposium on Parallel and Distributed Processing (IPPS/SPDP), San Juan, Puerto Rico, April 12 – April 16, 1999.
15. Tong Weiqin, Dong Jingyi, Meng Rui: Targeting BSP Library for SMP Cluster. J. of Shanghai University. Vol. 4, Suppl. Dec (2000)
16. http://www.globus.org/gram/rsl_spec1.html.
17. I. Foster, C. Kesselman: The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann (1998)
18. http://www.math.uu.nl/people/bisselin/software.html
19. T. L. Williams, R.J. Parsons: The Heterogeneous Bulk Synchronous Parallel Model. Parallel and Distributed Processing. Lecture Note in Computer Science, vol. 1800, Springer-Verlag, Cancun, Mexico, (2000) 102–108
20. http://www.upb.de/~pub/docu/pub8.pdf
21. D. Dunning, G. Regnier, G. McAppine, D. Cameron, B. Shubert, F. Berry, A. Marie Merritt, E. Gronke, C. Dodd: The Virtual Interface Architecture. IEEE Micro, vol. 18(2) (1998) 66–76
22. V.S. Sunderam: PVM: A Framework for Parallel Distributed Computing. Concurrency: Practice and Experience, vol. 2(4) (1990) 315–339
23. Message Passing Interface Forum: MPI: A Message Passing Interface Standard. Tch. Report Version1.1, Univ. of Tennessee, Knoxville, Tenn (1995)

24. Jonathan M. D. Hill, Stephen R. Donaldson, David Skillicorn: Stability of Communication Performance in Practice: From the Cray T3E to Networks of Workstations. Technical Report PRG-TR-33-97. Oxford University Computing Laboratory (1997)
25. I. Foster, C. Kesselman, J. Nick, S. Tuecke: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Globus Project (2002)
26. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman: Grid Information Services for Distributed Resources Sharing. Proc. 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC-10). IEEE Press (2001)
27. S. Fitzgerald, I. Foster, C. Kesselman, G. V. Laszewski, W. Smith, S. Tuecke: A Directory Service for Configuring High-Performance Distributed Computations. Proc. 6th IEEE Symp. on High Performance Distributed Computing
28. J. Bester, I. Foster, C. Kesselman, J. Tedesco, S. Tuecke: GASS: A Data Movement and Access Service for Wide Area Computing Systems. http://www.globus.org
29. I. Foster, C. Kesselman, G. Tsudik, S. Tuecke: A Security Architecture for Computational Grids. Proc. 5th ACM Conference on Computer and Communications Security Conference (1998) 83–92
30. Foster, D. Kohr, R. Krishnaiyer, J. Mogill: Remote I/O: Fast Access to Distant Storage. Proc. Workshop on I/O in Parallel and Distributed Systems (IOPADS)
31. http://www.globus.org/research/papers/ogsa.pdf