Using the UML to Describe the BSP Model of Parallel Computation

M. Scherger, J. Baker, and J. Potter
Department of Computer Science
Kent State University
Kent, Ohio, U.S.A. 44242

Abstract – *A Unified Modeling Language (UML) description of the BSP model of parallel computation is presented. This UML description identifies BSP classes and objects and specifies various object and inter-object relationships, dependencies, and behaviors. This was achieved by describing various views of the BSP model using many of the UML structural and behavioral diagrams. The use to the UML to describe the BSP model has been highly effective for further parallel modeling techniques, comparisons to other parallel models, BSP parallel system software research, and BSP algorithm development.*

Keywords: *parallel models, BSP, object oriented, parallel architectures.*

1 Introduction

The Unified Modeling Language (UML) developed by Booch et al. [1] has become the standard modeling tool for specifying, visualizing, constructing, and documenting software intensive systems. The UML is a modeling language capable of providing several different views of a system for various target software and hardware disciplines. This is achieved by presenting different diagrams detailing various structural, object behavioral, activity, and usage interactions of the components in the model; each intended to detail a particular aspect of component interaction.

The Bulk Synchronous Parallelism model of parallel computation is highly regarded as a computational model of parallel computing [3]. BSP is a model of parallel computation that abstracts from low-level program structures in favor of supersteps. A superstep is defined as a set of independent local computations followed by a global communication phase and a barrier synchronization step. The model consists of:

- A set of processor-memory pairs.
- A communication network that delivers messages in a point-to-point manner.
- A mechanism for the efficient barrier synchronization for all or a subset of the processes.
- There are no special combining, replicating, or broadcasting facilities.

Some of BSP's fundamental properties are that programs are simple to write, the model is independent of target architectures, and the performance of a program on a given architecture is predictable [4]. This is achieved by considering computation and communication at the level of the entire program and executing computer [4].

The original description in [3, 4, 5, 6, 7] provides a conceptual view of the principle components and basic component interactions of the BSP model. Using the UML to describe BSP transforms this conceptual description into an object oriented model for further BSP research in algorithm development and predictability, system software design and implementation, hardware simulation and modeling. This object-oriented description of BSP details the principle objects and inter-object behaviors by using the UML structural and behavioral diagrams.

The remainder of this paper will first give a conceptual description of the BSP model of parallel computation and also present the BSP predictability parameters used in algorithm predictability analysis. Next the basic object of the BSP model are presented and organized into classes for basic structural modeling. The structure UML descriptions are presented including a class hierarchy of the various objects in BSP. Once the structural elements are defined, the object responsibilities are outlined ad the basic use-case diagrams of BSP are presented. Other behavioral UML diagrams are presented that describe the sequence diagrams for the BSP predictability parameters.

2 The BSP Model

The following is a conceptual description of the BSP model of parallel computation. As stated in the introduction, the model consists of a set of processor-memory pairs, a communication network for point-to-point messages, and a mechanism for efficient barrier synchronization for a set (all) of the processes.

The BSP model has a vertical and horizontal structure illustrated in figure 1. The vertical structure consists of a series of supersteps. Each superstep is composed of a set of virtual processes performing local computation. This is followed by a process communication phase in which the processes communicate using point-to-point communication for the movement of data between the local memories of the processes. Finally, a barrier synchronization phase is performed to wait for all communication actions to complete. The horizontal structure of the BSP model arises by utilizing concurrency among a fixed number of virtual processes. Processes do not have a particular order and can be mapped to a fixed number of processors in any manner. The parameter p is used to denote the number of virtual parallelism in a program, i.e. the number of virtual processes.

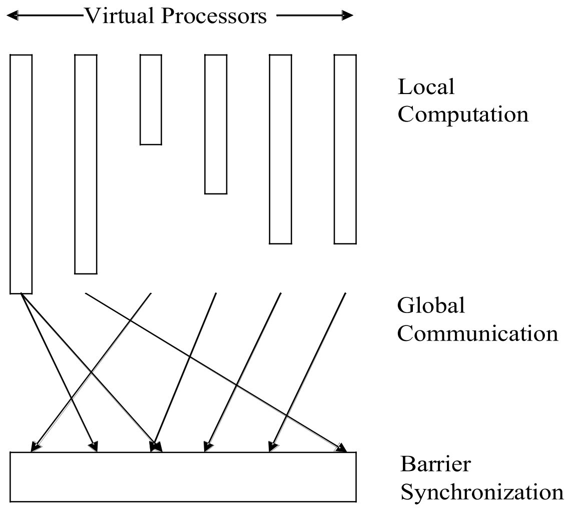


Figure 1: Basic phases of BSP programming.

For algorithm analysis, there are four predictability parameters defined in the BSP model.

1. $\boldsymbol{w_i}$ is the time for local computation in process i.
2. $\boldsymbol{h}$ is the maximum number of incoming or outgoing messages per processor.
3. $\boldsymbol{g}$ is the permeability of the network to continuous traffic addressed to uniform destinations.
4. $\boldsymbol{l}$ is the cost of performing a barrier synchronization.

There are other predictability parameters often defined in the BSP model such as $\boldsymbol{s}$, the number of flops/s when implemented on a particular machine and $\boldsymbol{m}$ the length of a message. However, BSP makes no distinction between a message of length 1 (one) or of length m.

Thus the cost of a superstep is defined by the following equation:

$$\underset{processes}{MAX}(w_i) + \underset{processes}{MAX}(h_i g) + l$$

or more commonly expressed as:

$$w + hg + l$$

The standard BSP cost model is the sum of the cost of each superstep. A further discussion of the predictability parameters and other BSP cost models can be referenced in [4, 6, 7, 8].

There have been several implementations of the BSP model, including implementations on the Cray T3E, IBM SP2, SGI Origin 2000, and various clusters of workstations. One such implementation is the BSPlib define in [5]. The BSPlib defines the operations of processes, communication, and barrier synchronization using an API of (approximately) 20 functions. Although a complete description and example of these functions is outside the scope of this paper, they do further illustrate the properties and requirements of the BSP model.

Initialization Functions
bsp_init() – Simulate dynamic processes
bsp_begin – Start of SPMD code
bsp_end() – End of SPMD code

Inquiry Functions
bsp_pid() – Find my process id
bsp_nprocs() – Number of processes
bsp_time() – Local time

Synchronization Functions
bsp_sync() – Barrier synchronization

DRMA Functions
bsp_pushregister() – Make region global visible
bsp_popregister() – Remove global visibility
bsp_put() – Push to remote memory
bsp_get() – Pull from remote memory

BSMP Functions
bsp_set_tag_size() – Choose tag size
bsp_send() – Send to remote queue
bsp_get_tag() – Match tag with message
bsp_move() – Fetch from queue

Halt Functions
bsp_abort() – One process halts all

High Performance Functions
bsp_hpput()
bsp_hpget()
bsp_hpmove()

3 BSP Structural Diagrams

The structural diagrams of the UML provide the model with a foundation of classes, objects, aggregations, and inheritance. Figure 1 presents an example class for the BSP process.

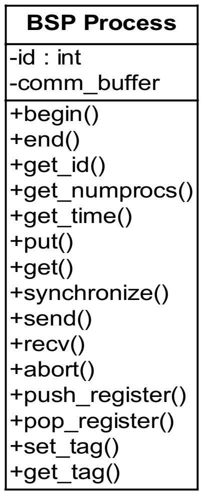


Figure 2: BSP Process Class Structure

The BSP process class is modeled after the BSPlib functions. Each BSP process has an process identification number and a communication buffer as class attributes and remain as private data members. The BSP process public methods are primarily implementations of the BSPlib API functions. Other classes for the BSP model include Interconnection Network, Processor, and Local Memory can be similarly described. However these classes are primarily abstract and ultimately depend on the implementation of the BSP model on a parallel computer and interconnection network.

The classes in the BSP model can be combined and organized into a UML aggregation diagram as shown in the figure 2.

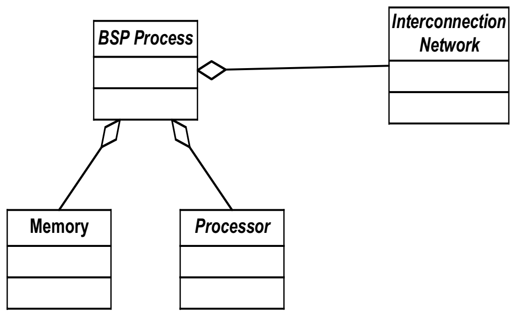


Figure 3: The BSP process using the UML aggregation diagram.

Assuming virtual BSP processes, top BSP process class has a one-to-one relationship with the process memory and processor classes. Also there is a one-to one relationship with the BSP process and Interconnection Network class.

4 BSP Behavioral Diagrams

The UML behavioral diagrams define class responsibilities by using the UML use cases. Behavioral diagrams also can define the object state transitions by using the UML state diagrams and event constraints by using the UML sequence diagrams.

Two such BSP actors are the BSP process class and the Interconnection Network class. The responsibilities and behaviors of the BSP process and Interconnection Network class are described in figure 4.

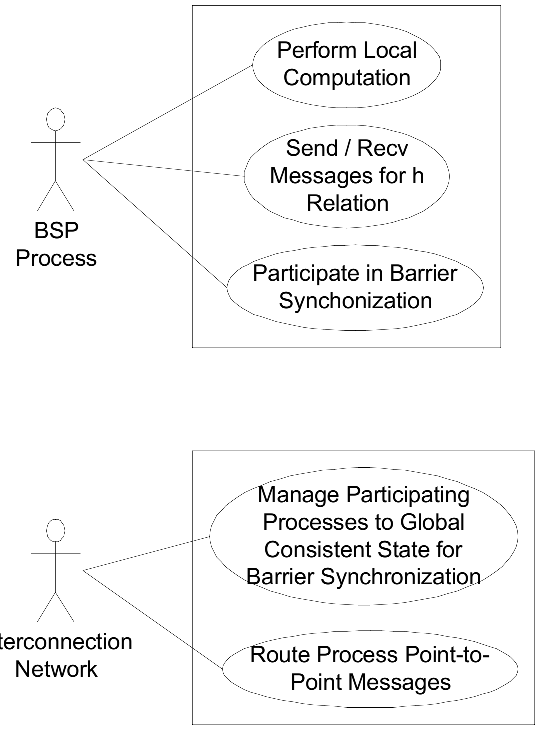
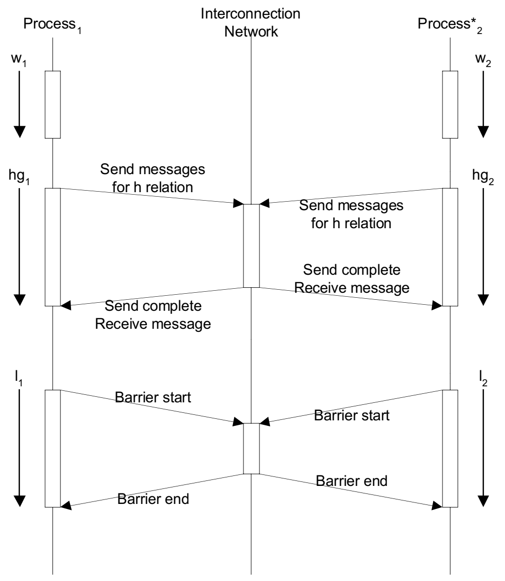

Perform Local Computation
Send / Recv Messages for h Relation
BSP Process
Participate in Barrier Synchonization
Manage Participating Processes to Global Consistent State for Barrier Synchronization
Interconnection Network
Route Process Point-to-Point Messages

Figure 4: Example BSP actors and use case diagrams.

The superstep and BSP predictability parameters can also be defined using the UML sequence diagram; one is presented in this paper illustrating the basic superstep and BSP parameters is shown in figure 5.

Each superstep begins with each process (two shown in this example) performing computation on their respective local memories. Note that "Process *" is used to denote that multiple BSP Processes are involved during the progression of a superstep as defined by the h-relation. This is characterized by the parameter w_1 and w_2 respectively. This is followed by a series of point-to-point communications as defined by the h-relation for each BSP process. Each process must initiate communication with the interconnection network, transfer data to the other process, (possibly) receive data from another process, and then finalize communication. This is characterized by the predictability parameters *h* and *g*.



Process1
Interconnection Network
Process*2
w1
w2
hg1
hg2
Send messages for h relation
Send messages for h relation
Send complete Receive message
Send complete Receive message
l1
l2
Barrier start
Barrier start
Barrier end
Barrier end

Figure 5: The BSP superstep and predictability parameters w_i, h, g, and l using the UML sequence diagram.

Finally, synchronization is performed using a barrier method. This is characterized by the parameter *l*.

5 Example Parallel Algorithm

As an example of using the UML sequence diagram, consider the BSP algorithm for allsums (parallel prefix sum). In this algorithm, the partial sum of p integers is stored on p processors (BSP processes). The algorithm uses the logarithmic technique that performs $\lceil \lg p \rceil$ supersteps such that during the kth superstep, the processes in the range $2^{k-1} \leq i \leq p$ each combine their local partial sums with process $i-2^{k-1}$. The following algorithm further outlines this method.

```
int bsp_allsums1( int x )
{
  int ii, left, right;
  bsp_pushregister( &left,
                    sizeof(int));
  bsp_sync();
  right = x;
  for(ii=1; ii<bsp_nprocs(); ii*=2)
  {
    if( bsp_pid()+I < bsp_nprocs())
      bsp_put( bsp_pid()+ii,
               &right,
               &left,
               0,
               sizeof( int ));
    bsp_sync();
    if( bsp_pid() >= ii )
      right = left + right;
  }
  bsp_popregister( &left );
  return( right );
}
```

Modeling this algorithm using 4 BSP processes, it is shown that there are lg(4) = 2 supersteps. To begin, each BSP process has a local number that participates in the prefix sum. During the global communication phase, this number is sent to it neighbor during the first superstep. The each BSP process participates in a barrier synchronization which concludes the first superstep. The process repeats for the second superstep, etc. At the end of the algorithm, each BSP process contains the prefix sum.

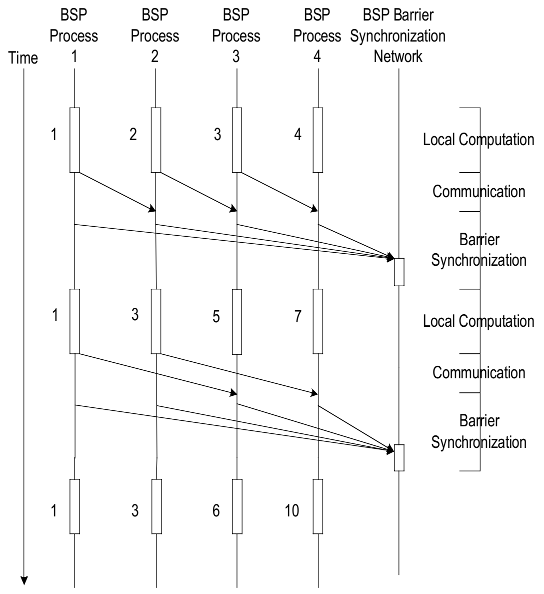


Figure 6: UML sequence diagram for the parallel prefix algorithm.

6 Conclusions

The Unified Modeling Language has provided a structured and organized communication vehicle to describe and develop and object oriented description of the BSP model of parallel computation.

By using the various UML structural and behavioral diagrams and views, the same model can be used for various disciplines of parallel computing research such as parallel architecture, parallel algorithm development, and object oriented implementations of the BSP model on parallel computing machinery. The structural UML diagrams of the BSP model provided the foundation classes and class aggregations. The behavioral UML diagrams of the BSP model provided the responsibilities and requirements of the classes in the model.

7 References

[1] Grady Booch, James Rumbaugh, and Ivar Jacobson, The Unified Modeling Language User Guide, Addison Wesley, Reading, MA, 1999.

[2] Bruce Powel Douglass, Real-Time UML: Developing Efficient Objects for Embedded Systems, Addison Wesley, Reading, MA, 1998.

[3] Todd Heywood and Claudia Leopold, "Models of Parallelism", Abstract Machine Models for Highly Parallel Computers, John R. Davy and Peter M. Dew, Eds., Oxford Science Publications, Oxford, England, 1995, pp. 1-16.

[4] J. M. D. Hill and W. F. McColl, “Questions and Answers About BSP”, http://www.comlab.ox.ac.uk/oucl/users/bill.mccoll/oparl.html

[5] J. M. D. Hill, et al. “BSPlib: The BSP Programming Library”, http://www.comlab.ox.ac.uk/oucl/users/bill.mccoll/oparl.html

[6] W. F. McColl, “Bulk Synchronous Parallel Computing”, Abstract Machine Models for Highly Parallel Computers, John R. Davy and Peter M. Dew, Eds., Oxford Science Publications, Oxford, England, 1995, pp. 41-63.

[7] W. F. McColl, Scalable Computing, http://www.comlab.ox.ac.uk/oucl/users/bill.mccoll/oparl.html

[8] L.G. Valiant, A bridging model for parallel computation, “Communications of the ACM”, 33(8): 103-111, August 1990.