Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

SIMULATION INTERFACE FOR THE BENEVOLENT BANDIT LABORATORY

Rusti Baker

January 1989
CSD-890001

Table of Contents

List of Figures

page

Simulation Interface for the Benevolent Bandit Laboratory
User Manual

Rev. 1.0
Software Version 4.0

1 Introduction

This manual is designed to facilitate the use of the Simulation Interface for the Benevolent Bandit Laboratory (SIBBL) which runs in the Benevolent Bandit Laboratory (BBL) environment [SCHO88], [SCHO88a]. SIBBL is a shell that runs on microcomputers in a PC network and permits the parallel execution of a simulation on multiple microcomputers. This system is available on IBM PC's running DOS. The system allows the user to transparently run many points of a simulation in parallel.

2 Installation

SIBBL runs in the BBL environment. The BBL software requires the the following hardware:

1) At least 3 IBM PC-AT or compatible systems
2) 3Com EtherLink Board #3C501 for each PC
3) Ethernet interconnection network

and software:

1) DOS version 3.1 including virtual disk installation (e.g. D: disk).

SIBBL requires the following three executable software modules:

1) Resource Manager (RM) - also used in BBL
2) Node Manager (NM) - also used in BBL
3) Simulation Manager (SM)

and two files (bbluser.h, bbluser.lib) used in compiling code to run on the system.

The BBL system provides an alternate interface for the execution of distributed programs. The User Interface/Process Manager (UI/PM) is used in place of the SM for distributed program execution. The BBL system requires the following modules:

1) Resource Manager (RM)
2) Node Manager (NM)
3) User Interface/Process Manager (UI/PM)

The BBL and SIBBL user share the same pool of nodes executing the Node Manager code and use only one Resource Manager. The User Interface/ Process Manager or Simulation Manager can be chosen to provide the appropriate interface to NM and RM. This manual includes a brief description of the RM and NM and information regarding the use of these modules by the SM manager. For more information on the

RM and NM, see [SCHO88] and [SCHO88a].

The Resource Manager must be installed and running on only ONE (1) machine on the network. The other two modules can be installed and running on any number of machines on the network. Ususally, the NM should be installed on all systems and DOS should be configured to automatically start the NM shell each time the system is brought up. The SM module needs to be available on systems that SIBBL users will run simulations from. Instructions for operation and termination of each module are outlined in the following sections.

Each piece of software should carry the same version number, otherwise compatibility problems may result. To find the version number of a module, consult the **Operation** subsection of the appropriate module's section.

3 Resource Manager

3.1 Description

The Resource Manager is a dedicated machine responsible for keeping track of the available PCs in the network. When a PC is not used for the number of seconds that has been defined as the timeout period, the Node Manager declares that the machine is idle and an "I_AM_UP" message is sent to the Resource Manager. Thus, an idle NM is added to the pool of free nodes owned by the RM. In this manner, an unused machine is made available to do work for BBL. When the user returns to the machine, the current owner of the machine (RM or SM) immediately returns control to the owner.

When the Resource Manager receives a request for free nodes from the Simulation Manager, it returns a list of physical Ethernet addresses. The SM uses these addresses when allocating new work to a node that is **idle** or **done**, or when reassigning work from a NM that has been taken away by its owner. The RM is able to support an arbitrary number of users of the BBL system, provided that there are sufficient idle PCs to fulfill all the requests.

3.2 Operation

The RM is invoked by running the executable file 'RM.exe'. Execute the RM by typing:

```
RM
```

When the RM begins execution, the following message appears on the PC screen:

```
BBL Resource Manager Version X.XX
```

where "X.XX" is the version number of the RM. This version number should match those on both the SM and all the NMs. This manual is intended to describe the operation of software version 4.0.

After printing its version number, the RM prints system status information. The RM lists the current NMs or registered users by printing the number of users on the system, the number of available NMs and the current time on the screen. As these numbers change, the current values will be updated on the screen. A typical example of the message printed by the RM at initialization follows:

```
BBL Resource Manager Version  4.00

Sending reset channel message
Delaying
Sending I_AM_THE_RM message
Delaying
Users = 0    Nodes = 1    16:33:11.74
WHERES_THE_RM
Users = 0    Nodes = 1    16:33:35.69
Users = 0    Nodes = 1    16:33:42.28
Users = 1    Nodes = 0    16:33:46.95
```

Users = 1 Nodes = 0 16:33:56.12

After the simulation has been initiated, the RM will let the SM know if new nodes become available. Depending on the number of maximum nodes requested for the simulation and the number of nodes currently owned by the SM, the RM may be asked to allocate new nodes to the SM. The Resource Manager notifies the Simulation Manager each time an "I_AM_UP" message is received. The Simulation Manager will request ownership of the new NM if it has not exceeded the number of maximum nodes specified by the user.

3.3 Termination

The Resource Manager can be terminated by simply typing any key on the keyboard. The system monitors the keyboard continuously, and terminates the execution of the RM whenever a key is hit. Once a key is hit the following messages should appear:

```
GOT KEY HIT.... TERMINATING
Pointer is NULL
First Reset got (80)
Second Reset got (89)
RESTORING INTERRUPT VECTOR
```

followed by the DOS prompt. The RM can be restarted by simply typing "RM". Stopping the RM will not usually terminate the simulations being run on the SM and NM nodes. However if the SM makes a request for free nodes from the RM before it is restarted, it may detect the missing RM and start error processing that will slow down the execution of the simulations. Ususally, the RM should not be stopped and restarted while simulations are running on SIBBL.

4 Node Manager

4.1 Description

The Node Manager's purpose is to steal a machine from its owner after the machine has been in the idle state for a given number of seconds. A PC is said to be idle when it is displaying a DOS prompt, waiting for the owner to type a command. The Node Manager is designed as a shell on top of DOS and emulates its operation. It is normally configured to automatically execute when the system boots. The NM shell waits for the owner to type commands, decrementing a timer as it waits. When the owner completes a command by hitting the return key, the timeout counter is reset and the NM passes the command on to DOS for execution. The timeout value is a parameter passed when the shell is started.

In addition to allowing the user to execute DOS commands, the NM's task is to send a message to the Resource Manager indicating it is free for use. The NM then waits to be assigned to a UI/PM that wants to perform a distributed computation or the SM node that wants to utilize it in a distributed simulation. When the NM is assigned to a SM node, it waits for messages from the SM which contain code to be executed and the arguments to be passed on the command line to the simulation. Once this code is received, the NM executes the code and returns the results to the SM which logs the output from the simulation in a file in the user's directory.

If a key is hit at any time while the NM has control of the PC, the NM notifies the Resource Manager (if it is still in the RM's pool of available nodes), or the Simulation Manager (if it has been assigned to a user) that it is going down and immediately returns to processing commands from its owner. The owner does not notice any delay, since the overhead for processing the context switch is negligible.

When the Node Manager receives the simulation code to be run it places the code into a virtual disk in the memory of the PC. This downloaded code is simply an executable file. When it receives the parameters from the SM the NM appends them to the command line that will passed to DOS to invoke the simulation. The SM notifies the NM that it is to start running the code, at which time the NM gives DOS the command line. Output from the program, which contains results of the simulation, is copied from standard output to a temporary file during execution. At the end of the simulation, these results are written back to the SM, and the next point to be run is given to the NM.

4.2 Operation

The NM runs as a process on top of DOS. It is invoked by running the executable file called 'shell.exe'. The Node Manager will begin execution by simply typing "SHELL n" ("n" replaced by an integer). The parameter passed to the shell is the number of seconds that the machine must be idle before the Node Manager takes over. The only indication that the NM is running is by the prompt printed on the screen. It is generally of the form "BBL [C:BBL]". The prompt begins with "BBL" and then contains the name of the current directory inside brackets. The above example shows that the default directory is "BBL" on the C disk (hard disk). Operation at this point will appear to the user to be the same as if DOS were running. Any DOS commands simply pass through the program and are executed by DOS. There are some exceptions to this. The current version of the NM (4.0) will not work in con-

junction with PC Interface (PCI), the program to allow file access to a mainframe running a PCI server. This is because both the NM and PCI use the 3Com EtherLink board and we have yet to resolve the contention. Therefore, **do not run the NM and PCI simultaneously!** They are actively hostile to each other and the machine. You will be forced to reboot the machine if these two programs are running concurrently.

The command **quit** causes the NM to terminate and return direct control to DOS. If the owner opts not to include her PC in the pool of BBL NMs, the NM shell can be terminated by typing "EXIT" or "QUIT" to it.

4.3 Termination

As mentioned above, to terminate the NM shell, simply type "QUIT" or "EXIT" at the BBL prompt.

5 Simulation Interface

5.1 Description

The SM interface is invoked by running the executable file 'sim.exe'. Execute the SM by typing:

SIM

When the SM begins execution, the following message appears on the PC screen:

> The Resource Manager has *x* nodes available
>

The prompt for the SM is a ">". The value of *x* depends on however many nodes the RM has available at the time the SM is initialized. If there is no message from the RM, it implies no connection has been established yet between the SM and RM. In this case, the **rm** command should be entered followed by **free** to find out how many nodes exist at the RM.

The SM interface enables the user to execute the simulation on a set of "transient" processors without explicitly initiating this execution or tracking the output from each execution. The SM also permits the user to view the current work being performed by NMs (**view** command), the data returned from the completed simulations (**results** command), and also allows the user to modify parameters during execution so that additional points can be run, or unnecessary ones deleted (**change_points** or **data_file** command).

One SM node exists for each user running simulations on the system. The Simulation Manager schedules the points that the user gives in the configuration file or in the input data file. To accomplish this, the SM must coordinate the allocation of NMs and reschedule points that do not complete because the NM is taken away. The SM also keeps track of the results from each simulation in a log file named by the user.

The SM stores the status of each of its nodes. A node allocated to the SM can be in one of five states; **free** to be used by a simulation, **loaded** with the simulation code, **busy** running the code, **done** running the code, or **dead**. The SM relies on this information when a node is taken back by its owner. The status of the node is used in rescheduling the work being performed by the node in the event that the user reclaims the node. The status flags are also used to determine the availability of a node in the event that points are added to the simulation during execution.

5.2 Operation

The SM accepts commands from the user and takes actions based on the prompt. Menus and help messages assist the user in finding the appropriate action to request. This interface closely parallels that of the UI/PM.

The SM module manages the allocation of nodes (based on information provided by the user), the downloading of code to the NMs owned by the SM, and the scheduling of work to be assigned to each NM. The SM defines two classes of commands; those considered legal during the selection and initialization of a simulation (the setup phase) and those intended for use during and after the execution of the simulation (the runtime phase). The SM presents a different set of commands depending on whether the system is in the setup or the runtime phase. A user selects a command by either typing out the name in full or typing however many characters are needed for the SM to recognize the name as unique. A description of the SM command language follows: ~

SM Command Language

Command	Phase†	Description
bye	B	Exit from the BBL environment
change_points	R	Allow the input parameters to be modified if these parameters were give in the configuration file for the simulation
data_file	B	Change the default data input file if input parameters were not given in the simulation configuration file
dos	B	Invoke a DOS environment
exit_dos	B	Return from DOS environment
flow_control_diagram	B	Diagram of the flow of control of the SM command language
free_nodes	B	Ask the RM how many NMs are available
heartbeat	B	Ask the RM to broadcast a heartbeat message to pick up any stray NMs
help	B	Present the current SM command language options
output_file	B	Select a new file for simulation results to be written to
quit	B	Exit the BBL environment (same as the bye command)
reset	B	Reset the simulation to its initial state
results	R	Display the current list of results received from completed simulations
rm_addr	B	Broadcast to find the address of the RM (when the RM is started after the SM)
runtime	S	Enable the runtime phase routines
setup	S	Enable the setup phase routines
simulation	S	Choose a simulation from the 'info.sim' simulation library file

† The phase during which the command is enabled; S=Setup, R=Runtime, B=Both.

SM Command Language (continued)		
Command	**Phase†**	**Description**
start	R	Start the execution of the user's distributed simulation
time	R	Retrieve execution times of NMs and entire simulation
view_points	R	Allow the user to view the system state (i.e. The data point that is being run by each active node).
"?"	B	Present the current command language options (same as the help command)

The user is prompted in the setup phase to select a simulation from the list contained in the 'info.sim' file. The simulation description files that are listed in this file contain information about the name of the executable file, the maximum number of nodes to be used, and optionally, the parameter values to be run in the simulation. In the configuration file, a negative integer indicates an unbounded limit on the maximum number of processors. The optional information in this file follows the simulation executable name and the maximum number of processors. The following describes the organization of the entries in the simulation configuration file:

```
<Comments and header information>

test.exe                        /* Name of the executable */
-1                              /* Maximum number of NMs */

<1st parameter beginning value>
<1st parameter ending value>
<1st parameter delta value>

<2nd parameter beginning value>
<2nd parameter ending value>
<2nd parameter delta value>
        .
        .
        .
<last parameter beginning value>
<last parameter ending value>
<last parameter delta value>
```

The items in brackets are optional. The space between the header line and the executable file name is required, all others are optional and are included to improve readability. If the user does not wish to include the input parameters in the configuration file, or if the values are not easily represented as a range with discrete increments over the interval, the user may exclude this field from his simulation description file.

† The phase during which the command is enabled; S=Setup, R=Runtime, B=Both.

If no input parameters are given in the configuration file, the SM will prompt the user for the input data file. This file is chosen from a list of files in 'data.lst'. The user must edit the 'data.lst' file to include the name of the data file prior to the setup phase. There is not an option to enter the parameter list interactively at this time. Parameters can either be entered in the configuration file, or in a data input file named in the data file library.

During execution the user monitors the progress of the simulation by using the runtime phase commands. The **view** command allows the user to display the parameters being run by each NM owned by the SM manager. The **results** command allows her to check the output from simulations that have completed. If the user wishes to modify the points that are being run, she has two options. If the points are parameters included in the configuration file, she can modify the parameters with the **change_points** command. If the parameters are being read from an input data file, the user may edit this file during execution to either add or delete points; this is accomplished by using the **dos** command to go the DOS environment in order to edit the file, and returning to SM by the **exit** command. The user can determine which NMs have completed by issuing a **time** command which will list the vids which are still **busy** and the completion time for those that are **done**. For an example of the interaction with the user to retrieve setup information see **Appendix A**.

5.3 Termination

The commands **bye** or **quit** are used to exit the SM.

6 Writing Simulations for SIBBL

The BBL system provides a set of library routines to allow the user program to execute on the NM. For the SIBBL environment, the only library routine required is the function **Init_Vars** which must be the first line in the program. This function allows the NM to set up various data structures and links into the user's code specifically, an area for packet buffers and associated control structures and the address of an exit routine to be called when the NM is forced to terminate the user's code.

The **Init_Vars** function must be called at the beginning of the user code. This is because the NM must have an address of an exit routine to call when a key gets hit, or when instructed to kill the node. The address of the exit routine is passed as a parameter to **Init_Vars** as "user_exit". This exit routine must free any allocated space. This routine **must** end with a call to the function "exit". The other parameters passed are information for the BBL system and are not used by SIBBL. Generally the call will look like this:

```
Init_Vars(input_buffer, input_size, input_free, U_QUEUE_SIZE, user_exit);
```

In order to issue the function call to the NM, the user must include the 'bbluser.h' file in their C source and must include the 'bbluser.lib' file when linking their code. The include file, 'bbluser.h' defines the parameter structures used in the **Init_Vars** call. The **Init_Vars** routine is in 'bbluser.lib'. This library file is linked with the user's source.

In addition to making a call to the BBL library routine **Init_Vars**, the simulation must be written to accept its inputs as arguments from standard input and write its results to standard output. The input line cannot exceed 80 characters when converted to an ascii string, and the output should be no more than 1100 bytes.

In summary, the restrictions upon simulation programs are:

1) Code must be written in C
   (to link with the BBL library)
2) Each simulation in the library
   must be compiled into a '.exe' file
3) No interrupt driven routines
4) First executable statement MUST be a call to the **Init_Vars** routine
5) Input passed as an argument to the program and no
   more than 80 characters
6) Output written to standard output and less than 1100 bytes
7) Must be compiled with the large model in Turbo C
8) Must provide an exit routine that releases user allocated memory in
   the event of abnormal termination.

7 Simulation Execution

In this section, a sample simulation (STAGE.sim) is provided to illustrate the use of SIBBL. The purpose of this simulation is to analyze the correctness of an approximation made in an analytical evaluation of a task graph model. The parameters to 'STAGE.exe' program are the number of stages and the number of processors used in the distributed computation of a task. The product of these two numbers is the number of iterations in the calculation of mean execution time. The creation of all necessary files for the execution of this simulation will be described in this section.

7.1 The Simulation Configuration File (STAGE.sim)

After writing and compiling the simulation code into the file called 'STAGE.exe' the simulation description file must be created. The simulation configuration file 'STAGE.sim' is created and added to the list of other configuration file names in the 'info.sim' library file, so that it is accessible from the SIBBL environment.

The 'STAGE.sim' file contains the name of the executable and additional information about the execution of the simulation. An upper bound on the number of processors to be used to run points can be set in this file and the ranges of values for parameters can be given for the simulation. If the user has not given the parameters in the configuration file, they are prompted for the name of the data file which contains the points to be run. Each line of input in the input data file should be separated by a carriage return. After this information is input, the system is ready to invoke the runtime phase commands.

7.1.1 Required Data

The beginning entries in a configuration file are required, while the latter are optional. The first required lines of the file are comments meant to describe the simulation. Therefore, 'STAGE.sim' might begin something like:

```
The stages simulation is allowed to utilize as many processors as
it is able to allocate. The length of execution time is proportional
to the product of the two input parameters. Inputs are integers
ranging from 1..10.

STAGE.EXE           /* executable name */
-1                  /* maximum number of nodes */

10                  /* first parameter to the simulation */
1                   /* ranges from 10..1 */
-1                  /* decremented by one at each step */

10                  /* the second parameter is always 10 */
10
0
```

A blank line terminates the comment section. This is followed by the name of the '.exe' file, and the maximum number of processors on which this code should run.

For 'STAGE.sim' maximum is set to -1, indicating an unlimited upper bound. The lines following this are optional; they describe the parameters to the simulation.

7.1.2 Optional Data - Input Parameters

The optional entries in the configuration file should be arranged in the order which they are to be passed to the simulation. Each parameter requires three lines which describe the beginning, end, and delta value for the parameter. These values can be real or integer, positive or negative. If the parameters to the simulation are non-numeric values, the parameters must be retrieved from an input data file, the parameter input cannot be specified in the configuration file. For the "STAGE.sim" file there are two parameters to the simulation, one ranges from 10..0 decremented by one after each execution, and the second parameter is always 10. The parameters for the simulation should be given in order of greatest to least anticipated execution time. This is necessary to insure that scheduling of the points is highly efficient. The points are run in the order given in the configuration file. If the input data is being read from a file it is run in the order given in the file, so the same guidelines apply. The simulation will terminate as soon as one of the parameters has reached the end value specified unless the delta value for this parameter was zero.

7.2 The Simulation Input Data Library

Input data file names must be appended to the system 'data.lst' file in order to be listed and selected from the SM. Data in the files listed in this library should be the argument lines to be passed to each invocation of the program written by the user. Each line must be terminated by a carriage return as in the input data file for the stages simulation, 'STAGE.dat':

```
10 10
9 10
8 10
7 10
6 10
5 10
4 10
3 10
2 10
1 10
```

If the 'STAGE.sim' configuration file had not specified the input parameters, the user would be prompted to select an input data file. The example data file given above would cause the same points to be executed as in the configuration file 'STAGE.sim' example. After each execution of the simulation, the results from the preceding executions can be displayed. Then modifications can be made to the data input file for subsequent runs of the simulation. It is not necessary to reset the simulation. If a data file exists that contains the next group of points to be run this file can be selected with the **data_file** command. Alternatively, the user could edit the current data file by exiting to dos then returning to runtime after modifying the file. It is not necessary to go through allocation and download again; the simulation can be run with the new points via the **start** command. Each time the simulation is restarted, SIBBL prompts the user for the name of a new output file. This is to prevent the

accidental overwriting of data accumulated in a previous run.

7.3 Simulation Results File

The output from each completed simulation is logged to a file named by the user at either setup or run time. Each line of the output file has the following format:

<input parameters> <output received from the NM> <carriage return>

The input is followed by the results for the inputs given and a carriage return terminates the entry in the file. The contents of the results file can be viewed during or following execution by using the **results** command. The results command displays one screen of data at a time. The user can inspect the completed points without exiting the runtime mode or resetting the simulation. The display of data is terminated by issuing a **quit** command when the SM prompts the user for continuing the display of data. If a **quit** is not issued, the user pages through the data until the end of the results file is reached.

The output from the user simulation should not have new lines in the data to be returned. The Simulation Manager interprets extraneous carriage returns as termination of data and will not write any additional data following a carriage return character to the log.

8 References

[SCHO88] Schooler, E.M., Felderman, R.E., Kleinrock, L., "The Benevolent Bandit Laboratory: User Manual for Software Version 3.0" Technical Report #880017, University of California, Los Angeles, CA, (March 1988).

[SCHO88a] Schooler, E.M., Felderman, R.E., Kleinrock, L., "A Testbed for Distributed Algorithms using PCs on Ethernet" Technical Report #880016, University of California, Los Angeles, CA, (March 1988).

9 Appendix A: Demonstration of the SIBBL System

The following demo of the SIBBL environment makes use of the example simulation discussed in section 7.1, **The Simulation Configuration File**. The configuration file for the stages simulation looks as follows;

```
The stages simulation is allowed to utilize as many processors as
it is able to allocate.  Inputs are integers
ranging from 1..10.

STAGE.EXE               /* executable name */
5                       /* maximum number of nodes */

10                      /* first parameter to the simulation */
0                       /* ranges from 10..0 */
-1                      /* decremented by one at each step */

10                      /* the second parameter is always 10 */
10
0
```

The entries in the configuration file are identical to those given in the earlier example with the exception of the maximum number of processors. In this configuration, the simulation is not to acquire more than five NMs. To begin the demonstration of the SIBBL environment, the user types "SIM" to the DOS prompt.

```
SM User Interface Version  4.00

> The Resource Manager has 4 free nodes
> help

        Command Language :
        ==================
        <debug_control>
        <dos>
        <flow_control_diagram>
        <free_nodes>
        <heartbeat>
        <help | ?>
        <quit | bye>
        <reset>
        <setup>

> set
```

```
SETUP routines have been enabled

Choose one of the following simulation choices: (default is 'STAGE.sim' )
        (0) STAGE.sim
        (1) BROWN.sim
        (2) STRATEGY.sim
        (3) OFF.sim
        (4) TEST.sim
0
Nodes requested = 5, nodes to use = 4
Downloading . . . .
DOWNLOAD completed: All nodes successfully downloaded

Enter the name of the output file for results to be written to:
out1
```

The SM downloads the code, gets the input parameters for the first four executions of the simulation and sends this information to its assigned NMs. The user has given a maximum number of nodes equal to five. The simulation will only be allocated four initially. If a fifth NM becomes free during the execution of the simulation, the SM will acquire the NM. If any uncompleted work remains, the SM will download the simulation executable to the newly allocated NM and will assign it the next point to run.

```
> help

        Command Language :
        ==================
        <data_file>                 <quit | bye>
        <debug_control>             <output_file>
        <dos>                       <reset>
        <flow_control_diagram>      <runtime>
        <free_nodes>                <rm_addr>
        <heartbeat>                 <simulation>
        <help | ?>
```

At this point the RUNTIME phase routines can be enabled, and the simulation started.

```
> run
RUNTIME routines have been enabled

        Command Language :
        ==================
        <change_points>             <quit | bye>
        <debug_control>             <output_file>
        <dos>                       <reset>
        <flow_control_diagram>      <results>
        <free_nodes>                <rm_addr>
        <heartbeat>                 <start>
        <help | ?>                  <time>
        <data_file>

> start
> help

        Command Language :
        ==================
        <change_points>             <output_file>
        <debug_control>             <reset>
        <dos>                       <results>
        <flow_control_diagram>      <rm_addr>
        <free_nodes>                <start>
        <heartbeat>                 <time>
        <help | ?>                  <view_points>
        <quit | bye>

> results

        Results From Completed Nodes:
        =============================

 10 10 15.3842  15.1035  1.8240 29.2897 29.2905 -0.0027
  9 10 14.1314  13.8514  1.9815 26.3607 26.3614 -0.0027
  8 10 12.8639  12.5855  2.1636 23.4317 23.4324 -0.0027
  7 10 11.5785  11.3039  2.3709 20.5028 20.5033 -0.0027
  5 10  8.9356   8.6843  2.8123 14.6448 14.6452 -0.0027
  6 10 10.2710  10.0043  2.5966 17.5738 17.5743 -0.0027
  4 10  7.5627   7.3417  2.9215 11.7159 11.7162 -0.0027

        << End of Data >>
```

The first two entries on each line are the input parameters sent to the simulation. The user determines the format of the output and the amount of data output by the simulation.

```
> time
```

```
   Completion Time:
   ================
Vid 0 00:02:44.15
Vid 1 not done
Vid 2 not done
Vid 3 not done

   NM Execution Time:
   ==================
Vid 0 00:02:44.15
Vid 1 BUSY
Vid 2 BUSY
Vid 3 BUSY

Simulation not complete
```

After all points have completed, the user can again view the results from the simulation execution. If it is desirable to run additional points, the parameters can be modified using the SM **change** command and the simulation can be rerun using the new values as parameters.

```
>

Completion: All nodes have completed

>change

Which parameter would you like to change? (default is none)

        0) BEG: 10  END: 0   DELTA: -1
        1) BEG: 10  END: 10  DELTA: 0
0

Which field would you like to change? (default is none)
        1: Beginning value
        2: Ending value
        3: Delta value
1

Input new value:
6

Which field would you like to change? (default is none)
        1: Beginning value
        2: Ending value
        3: Delta value
3

Input new value:
-2
```

Which field would you like to change? (default is none)
1: Beginning value
2: Ending value
3: Delta value

Which parameter would you like to change? (default is none)

0) BEG: 6 END: 0 DELTA: -2
1) BEG: 10 END: 10 DELTA: 0

>start

Enter the name of the output file for results to be written to:
out2

>view

Current work for each node:
==========

vid 0: 6 10
vid 1: 4 10
vid 2: 2 10
vid 3:

>
Completion: All nodes have completed

>reset
reset_sys: send FREE_NM
reset_sys: free up allocated space free simulation related space
reset_sys: send SUICIDE
reset_sys: reset channels
reset_sys: free up debug info
reset_sys: re-initialize debug info
reset_sys: re-initialize structures

The user must use the **reset** command if she wishes to choose another simulation. The **reset** command returns the environment to the setup mode where a new simulation is chosen and new input parameters are retrieved and new output results files are opened.

> set
SETUP routines have been enabled

Choose one of the following simulation choices: (default is 'STAGE.sim')
(0) STAGE.sim
(1) BROWN.sim
(2) STRATEGY.sim
(3) OFF.sim

```
        (4) TEST.sim
0
Nodes requested = -1, nodes to use = 4
DOWNLOAD completed: All nodes successfully downloaded


        Choose one of the following data_files: (default STAGE.1)

        (1) STAGE.1
        (2) BROWN.1
        (3) STRATEGY.1
        (4) TEST.1
1

Enter the name of the output file for results to be written to:
out3

> run
RUNTIME routines have been enabled


        Command Language :
        ==================
        <change_points>             <quit | bye>
        <debug_control>             <output_file>
        <dos>                       <reset>
        <flow_control_diagram>      <results>
        <free_nodes>                <rm_addr>
        <heartbeat>                 <start>
        <help | ?>                  <time>
        <data_file>

> start
Completion: All nodes have completed
```

When all simulations have been completed, the user can exit by issuing the **quit** command.

```
> quit
pm_quit: send SUICIDE
pm_quit: send FREE_NM
pm_quit: free up allocated space
free algorithm related space
pm_quit: free up debug info
First reset got (89)
Second reset got (89)
RESTORING INTERRUPT VECTOR
```

10 Appendix B: Optimizing maximum number of processors

In many simulations the program execution times will depend upon parameters passed to the program, in others the parameters will have no affect upon simulation execution time. For those simulations where execution time depends upon the program parameters, the sequence in which points are run can greatly affect the total execution time. In general, the greater the time required for the simulation to execute a point, the sooner that point should be scheduled. Anticipated running time can be used to determine the execution priority of each point. The SM attempts to schedule points based on their execution time. To this end, the SM assumes that points are ordered from greatest to least execution time. If the input parameters are given in the configuration file, they should have a beginning value of the largest point and an end value of the least (delta values can be negative). If the parameters are retrieved from a data file, the inputs should be in order of greatest to least execution time. Points are scheduled in the order in which they are entered in the configuration or input data file.

The 'STAGE.sim' file specifies ten points to be run in the simulation. In this simulation, the length of each simulation point depends upon the size of the parameters passed to it. The parameters to the program control the number of iterations of the calculation performed in the program. Thus, in the execution of the points in 'STAGE.sim', the longest simulation is the point <10, 10>. The speedup of the simulation is bounded by the length of the longest single computation. Since the simulation cannot be executed more quickly than the single longest point, a lower bound on the simulation execution time is the time for the point <10, 10>. Timing tests demonstrate (see Figure 1) that the simulation execution time is not improved when more than six NMs are allocated for the ten points run in this example.

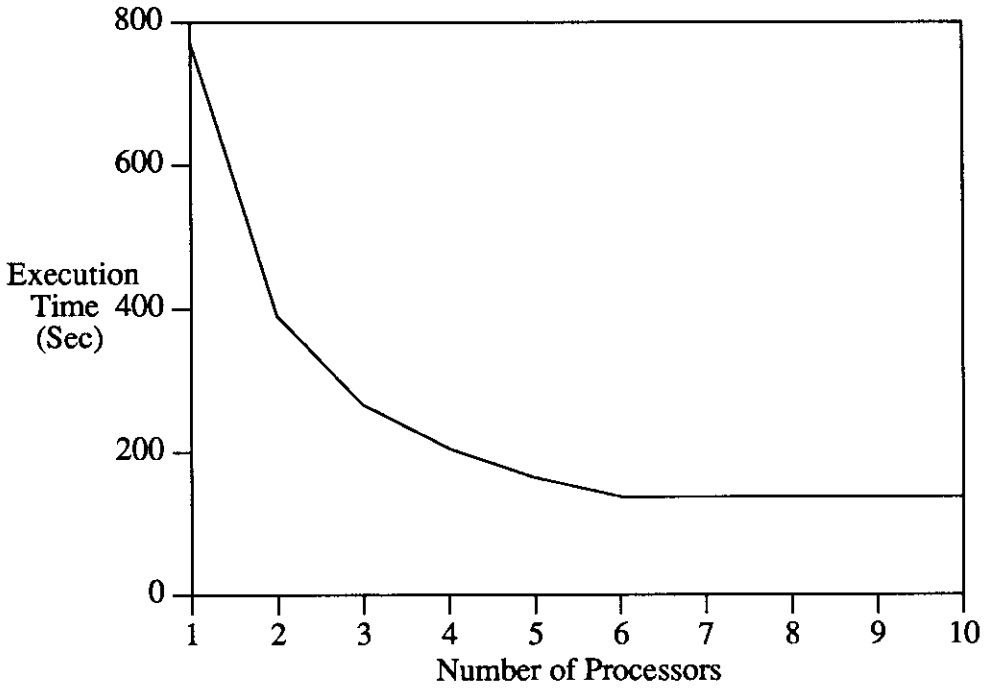

Figure 1. Execution Time versus Number of Processors (STAGE.sim).

This behavior should be taken into consideration when trying to optimize the maximum number of processors in the '.sim' configuration file. The speedup for the execution of all points is graphed in Figure 2. The speedup peaks at six processors.

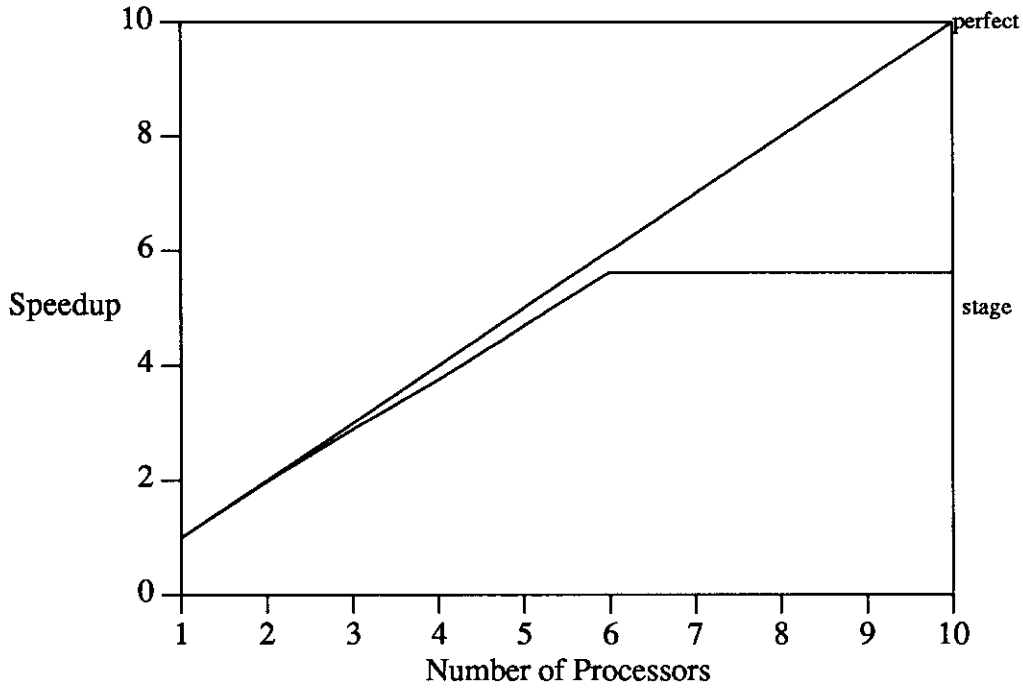

Figure 2. Speedup versus Number of Processors (STAGE.sim).

Allocating more than six nodes does not increase the speedup of the execution time for all simulation points being run in 'STAGE.sim'. In addition, it will reduce the number of NMs available to other users of BBL. The nodes which have completed remain idle until the last node completes and the simulation is restarted (with new work), or the Simulation Manager is returned to setup mode. Since SM retains ownership of all nodes as long as it is in runtime mode, the completion of a node does not mean it is returned to the pool of free nodes managed by the RM. Allocating more than six nodes to run this set of points wastes resources.

Not all simulations will exhibit this kind of behavior. For example, the Brownian motion simulation program has an execution time which is dependent only upon the number of trials executed. If all points perform the same number of trials, the execution times are very homogeneous. As a result, execution continues to decrease with additional processors.

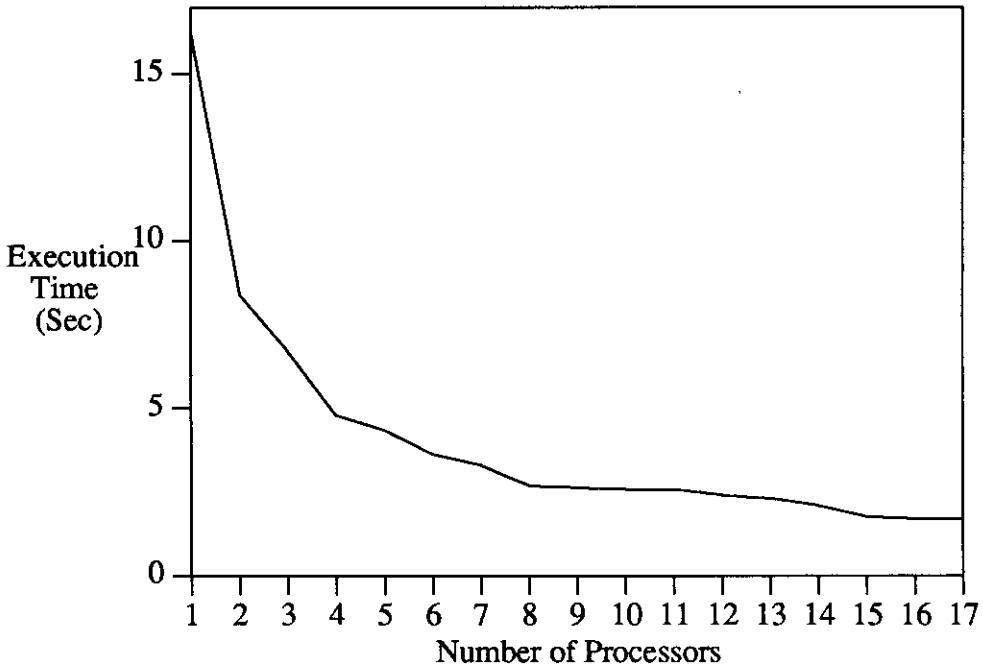

Figure 3. Execution Time versus Number of Processors (BROWN.sim).

The execution time for the Brownian motion simulation continues to decrease with addition of NMs because one point does not dominate the others in execution time as it did with the stages simulation. The speedup for this simulation is graphed against perfect speedup in Figure 4.

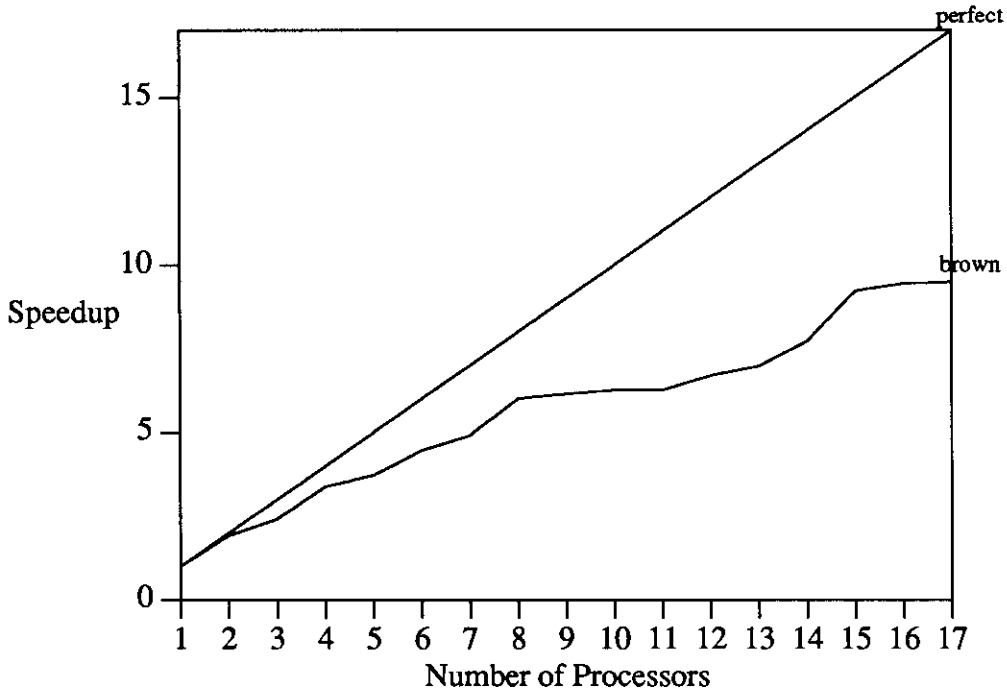

Figure 4. Speedup versus Number of Processors (BROWN.sim).

Speedup continues to improve until as many processors are being used as there are points to be run. Each processor is running only one point when there are at least sixteen processors. For this simulation, the speedup continues to increase with the addition of nodes. In general, it is desirable to allocate as many NMs as possible for a simulation running points with similar execution times.