NJIT
New Jersey Institute of Technology

A. V. Gerbessiotis | CIS 750
March 28, 2006 | Spring 2006
**BSPlib installation** | Handout 3

# 1 Introduction

Before you start reading this document consult and reread Handout 2.

Make sure you have changed the passwords both on `pcc15, pcc16`, though this is not critical, to your own secure password from the default ones.

Make sure that by now you have copied `/home/u20/cshrc` to your local directory as `$user/cshrc`. You will have to edit this file later on. Every time you edit this file, remote copy it to the `.cshrc` version in both `pcc15` and `pcc16`, and then logout and login again or type in the command-line

```
% source ~/.cshrc
```

You can copy this file with the set of commands.

```
% cd
% hostname
pcc16.njit.edu
% rcp cshrc pcc15:/home/$user/.cshrc
% rcp cshrc pcc16:/home/$user/.cshrc
```

All relevant BSP-related files are located in directory `/home/u20/bsp-files`. You can copy them to your local directory and use them as needed. In addition some files can be retrieved from the protected area of the web-page section C3, link 19.

The uniprocessor version of BSPlib needs only to be installed on `pcc16.njit.edu`. The multiprocessor version can be installed first on the same machine and all relevant files (the directory structure of `BSP`) be copied then to other machines of the cluster. However make sure that you update correctly the `.cshrc` files between the uniprocessor and multiprocessor installation.

## 2 Uniprocessor installation of BSPlib

**Step 0.1** We assume that you have a Red Hat linux system (RH 7.0 or later); the test platform that was used to compile these instructions was a RH 9.0 machine. Make sure your shell is tcsh. All `uXY` accounts have `tcsh` by default, unless you have deliberately changed that. You can check what your default shell is by typing the line below, and checking the last few characters after the colon : in the response that should read `/bin/tcsh`.

```
% cat /etc/passwd | egrep $user
u20:x:517:517::/home/u20:/bin/tcsh
```

If however `tcsh` is not your default shell, then you need to change it. Here is how this can be accomplished.

```
% chsh
```

and after insterting your password change shell appropriately

```
Changing shell for XXXXX.
Password:
New shell [/bin/bash]: /bin/tcsh
```

**Step 0.2** Logout and login again.

**Step 0.3** Edit the `.cshrc` or `cshrc` file and comment out the line

```
setenv BSP_DEVICE MPASS_UDPIP
```

so that it looks like

```
#setenv BSP_DEVICE MPASS_UDPIP
```

Then, make sure you copy the `cshrc` file to the relevant cluster machines as `.cshrc`. This is explained in Section 1 (Introduction). Then continue with the following steps that install the uniprocessor version of BSPlib.

**Step 1** Grab the source code v1.4a_bsplib_toolset.tar.gz and ungzip it, if necessary. The file is located in `/home/u20/bsp-files`.

```
% gzip -d v1.4a_bsplib_toolset.tar.gz
```

**Step 2** Tar xvf it as

```
% tar xvf v1.4a_bsplib_toolset.tar
```

**Step 3** A directory named BSP was created. Get into it

```
%  cd BSP
```

**Step 4** Unzip the documentation instructions if you like

```
%  gzip -d  v1.4_bsplib.README.ps.gz
```

**Step 5** Run configure.

```
%  ./configure
```

Question 1 is architecture (LINUX) : Accept it by pressing return and then confirming it by typing y for yes.
Question 2 is inter-processor comm : Choose SHMEM_SYSV for uniprocessor simulation.
Question 3 is number of procs : Choose 8.
Let it work out the configuration.

**Step 6** Run install by typing

```
%  make
```

**Step 7** Run make install by typing

```
%  make install
```

**Step 8** You can check if everything is ok by typing

```
%  bspcc
```

If you get bspcc: no input files specified
bspcc usage: for basic information, try the '-help' option
Then everything worked fine.
You can type

```
%  man bspcc
```

to get the manual pages

Type then

```
%  bsprun
```

If you get

```
bsprun: Command not found.
```

type

```
 % rehash
```

**Step 9** Run the program in directory `/home/$user/BSP/contrib/programs/bsp_probe` by going there and then typing in

```
 % make all
```

Make will output something like

```
bsp_probe.c: In function 'main':
bsp_probe.c:78: warning: return type of 'main' is not 'int'
bspcc  -flibrary-level 2 -O3  -o bspprobe bsp_probe.o
```

Disregard it.

**Step 10** You are ready to run the first "parallel" program. Type

```
% bsprun -npes 4 ./bspprobe
```

The `./bspprobe` IS REQUIRED BY LINUX to indicate that the executable is local. You will get an output of the form

```
**WARNING{bsp_begin}**
        Guessing the BSP parameters for a 4 processor LINUX machine
        using SHMEM_SYSV for communication. Please update the
        file "/home/XXXXX/BSP/include//bsp_parameters.ascii" with
        the information produced by running bspprobe, or from
        the following web site:

          http://www.bsp-worldwide.org/implmnts/oxtool/params_frame.html

Probe started on process 0 of 4 [pcc40.njit.edu]
Probe started on process 1 of 4 [pcc40.njit.edu]
Probe started on process 2 of 4 [pcc40.njit.edu]
Probe started on process 3 of 4 [pcc40.njit.edu]
For p=4, BSPlib's default values for the BSP parameters are:
        S=     10.0 Mflops
        L=   3000.0 flops/word ( 300.000 usec)
        g=     12.0 flops/word (   1.200 usec/word)

The calculated values are:
```

and the system will get stuck for a while... before it print something like the output below. This might take plenty of time: the uniprocessor library is simulating a 4-processor parallel machine. You might wait 10-20 minutes before you get the following output. Alternatively edit `bsp_prob.lc` and change some of the defaults into the following lower values.

```
#define MIN_SAMPLE          8      /*    10 */
#define S_DOT_OVERSAMPLE    2      /*    20 */
#define S_MAT_OVERSAMPLE    2      /*    10 */
#define L_OVERSAMPLE        500    /*160000 */
#define G_OVERSAMPLE        3      /*    30 */
```

```
S (average)    = 449.306 Mflop/s
L (low)        = 89861152 (-741.501818 usec) <<<Don't worry about this
L (high)       = 134791818 (3667.504605 usec)

(local shift) g=11149.48  block size =8192    32bit words
(local shift) g=11094.02  block size =4096    32bit words
(local shift) g=11094.02  block size =2048    32bit words
(local shift) g=33281.96  block size =1024    32bit words
(local shift) g=77657.86  block size =512     32bit words
(local shift) g=166520.69  block size =256     32bit words
(local shift) g=344024.35  block size =128     32bit words
(local shift) g=290551.62  block size =64      32bit words
(local shift) g=588258.68  block size =32      32bit words
(local shift) g=1267210.31  block size =16      32bit words
(local shift) g=2854259.60  block size =8       32bit words
(local shift) g=6645461.31  block size =4       32bit words
G_infty = 11094.02 (24.69 usec) Nhalf = 2392.050665 32bit words

(all-to-all)  g=11094.02  block size=8192    32bit words
(all-to-all)  g=11094.06  block size=4096    32bit words
(all-to-all)  g=11094.02  block size=2048    32bit words
(all-to-all)  g=33337.47  block size=1024    32bit words
(all-to-all)  g=79710.30  block size=512     32bit words
(all-to-all)  g=166465.20  block size=256     32bit words
(all-to-all)  g=344079.83  block size=128     32bit words
(all-to-all)  g=288554.72  block size=64      32bit words
(all-to-all)  g=588203.06  block size=32      32bit words
(all-to-all)  g=1267210.33  block size=16      32bit words
(all-to-all)  g=2856311.91  block size=8       32bit words
(all-to-all)  g=6645128.53  block size=4       32bit words
G_infty = 11094.02 (24.69 usec) Nhalf = 2391.930680 32bit words
```

Note: Because the parallel system is ”simulated” on a uniprocessor machine, the time it takes to complete this program is much longer than p times the “anticipated” uniprocessor version.

# 3 Cluster version of BSPlib

The BSPlib installation steps are similar to the uniprocessor installation. Some additional steps are required for the cluster version of the library to run parallel programs.

**We assume that your account is uXY in the remainder of this discussion**.

0. Remove the uniprocessor version of BSPlib from pcc16.njit.edu by doing a

```
% cd
% hostname
pcc16.njit.edu
% rm -rf /home/uXY/BSP  # This is a comment line  # is equivalent to // in C++
```

1. Edit the .cshrc file to uncomment the BSP_DEVICE line.

```
#setenv BSP_DEVICE MPASS_UDPIP
```

so that it becomes

```
setenv BSP_DEVICE MPASS_UDPIP
```

The last line is important if you plan to run the cluster version. The former line is probably already there from the uniprocessor installation. Interprocessor communication under BSPlib will utilize UDP/IP. You may also wish to verify that the following lines appear in .cshrc.

```
alias 13 'rlogin pcc13 -l $user'
alias 14 'rlogin pcc14 -l $user'
alias 15 'rlogin pcc15 -l $user'
alias 16 'rlogin pcc16 -l $user'
```

However the default cshrc file in u20 already includes this information. Copy it locally to your account and then copy it to the .cshrc files of both machines by following the Section 1 (Introduction) instructions.

```
% cp /home/u20/cshrc /home/uXY/cshrc    # You can grab the cshrc template from u20
% # Do as in Section 1.
```

2. Grab the tar file v1.4a_bsplib_toolset.tar and tar xvf it as before. Note the a after 1.4! This file is in /home/u20/bsp-files.

```
% cp /home/u20/bsp-files/v1.4a_bsplib_toolset.tar /home/uXY/
% cd /home/uXY
% tar xvf v1.4a_bsplib_toolset.tar
% cd BSP
% ./configure    # and read step 3 below on how to answer the questions
% make           # be patient; it takes time
% make install   # do not forget this step to complete installation
```

The library takes time to compile, because it compiles itself about 9 times,

3. Follow the steps of the uniprocessor installation when you execute the ./configure above except

```
  a. if you are asked about architecture   use LINUX
  b. communication medium                  use MPASS_UDPIP
  c  number of processors                  use 8 or larger
  d. switch                                use    Ethernet 100Mbit
  e. full duplex or half duplex            use full duplex switch
  f. Roundtrip time                        use 200 microseconds
  g. Send latency                          use 100 microseconds
```

The last two figures are not very important; you can always change the values during linking time within your own program. So don't worry if you type wrong.

4. After the installation completes, test it with a bspcc or which bspcc.

```
% which bspcc
```

If an error is returned, try first the sequence of commands below and then execute the line above.

```
% rehash
% hashstat
```

5. Copy the `ccp` script to `BSP/bin`. Also the `udpip` script.

```
% cp /home/u20/bsp-files/ccp   /home/uXY/BSP/bin
% cp /home/u20/bsp-files/udpip /home/uXY/BSP/bin
% chmod 755  /home/uXY/BSP/bin/ccp
% chmod 755  /home/uXY/BSP/bin/udpip
% rehash
% hashstat
```

Copy also the `bsptcphosts` file.

```
% cd /home/uXY/
% cp /home/u20/bsp-files/bsptcphosts   .bsptcphosts
```

6. From now on you can use the `rcp, rsh` commands to send information from one machine to the other. Information is available by typing for example `man rcp`. Create a tar file of the BSP installation on `pcc16` by doing at `/home/uXY` a `tar cvf BSP8.tar BSP` Do the following on `pcc16` to effect this.

```
% cd /home/uXY
% mkdir run                      # this is going to be needed later
% tar cvf BSP8.tar BSP
% udpip
```

The `ccp` commands copy the `.cshrc` and `BSP8.tar` to all the machines of the cluster. The `udpip` script does these and in addition it creates a directory `run`, just as you did above on `pcc16`, untars `BSP8.tar` and installs BSPlib from that tar-file on the remaining machines.

The command `ccp`, cluster copy, takes as a first argument a single file and as a second argument a directory. It copies the file into the directory on all machines of the cluster.

7. You can run some sample parallel programs written in BSPlib by creating a directory say `phello` somewhere in your filespace and copying there the phello tar file available in `bsp-files`. The most recent version is version 3 with tar file `phello2004v3.tar`. Tar xvf this tar file in `phello` and the issue a `make all` command to compile and link the various executable files. This is also depicted in step 10 below. In order to run an executable file say `a.out`, it must first be copied to the `run` directory of all machines. You can do that as follows.

```
% ccp a.out /home/uXY/run
```

Perhaps the first file to run in the created set of executable files is `hello` or `nprocs`. `ccp` it to the nodes of the cluster first using the command described above. You then need to setup the communication subsystem through step 8 below.

8. You are ready to start running programs. In order to allow interprocessor communication between any two of the four machines, run

```
% bsplibd -all
% bspload -all -start
```

Some communication programs are started for the cluster defined in `/home/uXY/.bsptcphosts`

After you are done with your coding, testing, and program execution, do a

```
% bspshutd
% bspload -all -end
```

to shutdown the programs you had started before. Note that after a shutdown you need to wait for a while (about a minute) before you restart with a `bsplibd`. Then you can run say `hello` with the following set of instructions (the first line is the `ccp` of the executable).

```
% ccp hello /home/uXY/run
% cd /home/uXY/run
% bsprun -noload -local -npes 2 ./hello
```

9. For files that do not have a makefile it is straightforward to compile, link, cluster copy nad run, a BSPlib program by typing.

```
% bspcc -O3 -flibrary-level 2 file1.c -o file1   # -O3 is optimization level 3
% ccp file1 /home/uXY/run
% cd /home/uXY/run
% bsprun -noload -local -npes 2 ./file1
```

**BSPlib will complain about** : `the use of ‘tempnam’ is dangerous, better use ‘mkstemp’`. Disregard this message. It is just a warning. It does not affect compilation. You can compile things under BSPlib under library-level 0, 1 or 2. The most efficient is level 2; the most debug-friendly but also slowest is level 0. For a uniprocessor version, it is better if you run it at level 1 than 2 for example. Do a `man bspcc` for more details on the differences.

10. We now show how to setup `phello` to install the sample programs and run the `nprocs.c` file.

```
% cd /home/uXY
% mkdir phello
% cp /home/u20/phello2004v3.tar phello/
% cd phello3
% tar xvf phello2004v3.tar
% make all
% ccp nprocs /home/uXY/run
% cd /home/uXY/run
% bsprun -noload -local -npes 2 ./nprocs
```

It should print something like

```
Hello World from process pcc16.njit.edu with id=0 of total 2
Hello World from process pcc15.njit.edu with id=1 of total 2
Number of processes allocated: 2
```

How does BSPlib correspond processor id’s to IP addresses? It uses `.bsptcphosts`. The last address in that file MUST BE the local machine. It is assigned a processor id of zero. The other address assignments are top to bottom starting from one. Note that `.bsptcphosts` MUST reside at `/home/uXY`.

Now modify the `.bsptcphosts` file to include

```
host(pcc15.njit.edu);
host(pcc15.njit.edu);
host(pcc16.njit.edu);
host(pcc16.njit.edu);
```

and run again the `nprocs` or `hello` program but now use 4 processor. Note that both cluster machines are dual-processor SMPs. You can use both CPUs for program execution.

```
% cd /home/uXY/run
% bsprun -noload -local -npes 4 ./hello
```

The output would look like

```
Hello World from process pcc16.njit.edu with id=0 of total 4
Hello World from process pcc15.njit.edu with id=1 of total 4
Hello World from process pcc15.njit.edu with id=2 of total 4
Hello World from process pcc16.njit.edu with id=3 of total 4
Number of processes allocated: 4
```

### 3.1 Checklist

- Have you updated `.cshrc` ?
- Have you copied `cshrc` from `u20` into `.cshrc` in `uXY`?
- Did you update/reinstall BSPlib on `pcc16`? Did you use the `1.4a` copy?
- Did you tar cvf the installation, copy it to the remaining cluster machines, and install it there using `udpip`? If in doubt, `rsh pcc15 ls -l BSP`.
- Did you create `run`?
- Before you run a BSP program did you `bsplibd -all , bspload` ?
- Did you by any chance reuse an old executable (compiled under the uniprocessor version) to run it under the cluster? It won't work!
- When you run a sequential program say, `iseq` located in the current working directory are you doing a `./iseq` or are you trying in vain an `irun`? Linux, for security reasons, does not have `.` (i.e. the local directory defined in the `$path` variable. Local files cannot be thus executed, unless one requests the execution explicitly, i.e. do an `./irun` on the command line.
- When you run a BSPlib program say, `ipar` located in the current working directory are you doing a `bsprun -noload -local -npes n ./ipar` or are you trying in vain a `bsprun -noload -local -npes irun` or even `bsprun -noload -local -npes n irun`? $n$ is a number that indicates the number of processors requested.
- Are yoy trying `bsprun -noload -local -npes 1 ./irun`? $p = 1$ programs won't run under the cluster version!

You might want to read the Postscript file `v1.4_bsplib.README.ps` that is available at any `BSP/` directory after you decompress it with `gzip -d v1.4_bsplib.README.ps.gz`. Just to make sure things are ok,

```
% setenv
.... other lines printed
BSP_DEVICE=MPASS_UDPIP
```

run `setenv` and make sure you see a line like the line above. Compile your executable file in some arbitrary directory and then run `ccp file /home/uXY/run`, where `file` is the name of the BSPlib created executable. What `ccp` does is something as simple as

```
rcp ./$1 pcc15.njit.edu:$2/$1
rcp ./$1 pcc16.njit.edu:$2/$1
```

If you are not familiar with `rcp`, then do a `man rcp` to see how it works.

You are ready to run your `executable`. Go to the `run` directory and type in

```
bsprun -noload -local -npes 4 file command-line-arguments-if-any
```

Note that BSPlib manual pages are available on line

```
% man bspcc
% man bsp_put
% man bsp_time
```