ParkBench 2.0 Release Notes

parkbench-comments@cs.utk.edu

April 16, 1996

Abstract

This report consists of the README files, the Run Rules and some miscellaneous information for the 2.0 release of the ParkBench suite of benchmarks.

Introduction

Welcome to the ParkBench suite of benchmarks. The ParkBench distribution consists of four distinct packages, all included here. All but 5 of the codes are parallel, and require that either PVM or MPI be installed *and configured properly.* MPI and PVM are software systems that present a uniform message passing API to different tasks in a parallel or distributed application. If neither of these are available, or your not even sure how to find out if they are, fire up Netscape, browse the Web and do some reading.

- ParkBench at `http://www.netlib.org/parkbench/html/index.html`
- PVM at `http://www.epm.ornl.gov/pvm/pvm_home.html`
- MPI at `http://www.erc.msstate.edu/mpi/`
- MPICH at `http://www.mcs.anl.gov/home/lusk/mpich/index.html`

Getting started

What to build?

Ok, first you need to decide which parts of the package you want to build. Compiling the entire distribution takes lots of space, especially if you have to build the libraries or are compiling for multiple architectures. That being said, if you have more than 100 Megs of disk and lots of cycles to spare, go for it. For those of you who are less fortunate, decide which of the four packages you'd like to build. If you only want certain executables, keep reading. If all else fails, go to the appropriate directory and look at the Makefile.

Here is a breakdown of the ParkBench distribution's directory structure.

Directory	Description
`Low_Level/`	5 Serial and 5 Parallel benchmarks to measure the performance of some hardware, system and communication operations.
`Kernels/`	5 Parallel Linear Algebra "Kernels" to measure the performance of some standard matrix operations.
`Comp_Apps/`	4 Sample Message Passing Application Codes.
`NPB2.1/`	The Nasa Ames Parallel Benchmarking Suite.
`lib/`	A stripped source tree of various parallel linear algebra library distributions. Chances are, you have optimized versions of these routines, which you can specify in step 2. The Makefile's in the subdirectories don't always compile the required libraries when building a single benchmark. You might have to go here and type `make` if the link fails.
`conf/`	Variable and rule definitions for different architectures that are included by the makefiles. These shouldn't need to be changed.
`bin/$PVM_ARCH`	This is where the executables end up.
`include/`	Header files included by some of the codes.

Configuring the build

Read and edit the file `ParkBench/make.local.def` to reflect your current setup. This file contains pointers and flags necessary to the build process. It's very short and self documenting. If you need to change *anything* not specified in this file, you should edit the file `ParkBench/conf/make.def.$PVM_ARCH`. Anytime either of these files change, `ParkBench/conf/make.def` will be *overwitten with a new version.*

The NPB configuration code `ParkBench/NPB2.1/sys/setparams` requires an ANSI C compiler. `/bin/cc` on SunOS 4.x will not work, a standardized compiler like `gcc` is always a good thing.

Selecting a target

Typing `make` or `make help` displays the possible targets, described below.

Target	Description
make conf	This makes the file ParkBench/conf/make.def which is included by all Makefiles in the distribution. It is a hacked version of ParkBench/make.local.def and ParkBench/conf/make.def.$PVM_ARCH"
make all	Make all executables for both PVM and MPI.
make all.pvm	Make all executables for PVM.
make all.mpi	Make all executables for MPI.
make Low_Level	Make the Low Level executables for PVM and MPI including...
make Low_Level.seq	The sequential codes.
make Low_Level.mpi	
make Low_Level.pvm	
make Kernels	Make all the Linear Algebra and the NPB Kernels.
make Kernels.mpi	
make Kernels.pvm	
make Comp_Apps	Make all the Compact and NPB Applications.
make Comp_Apps.mpi	
make Comp_Apps.pvm	
make NPB	Make all of the Nasa-Ames Parallel Benchmark Codes.
make NPB.mpi	Please read ParkBench/NPB2.1/2.1README for all
make NPB.pvm	of these.
make "program"	Make both the MPI and PVM executables for a single benchmark. (poly1 poly2 rinf1 tick1 tick2 comms1 comms2 comms3 poly3 synch1 LU_solver MATMUL QR TRANS TRD FT MG BT LU SP PSTSWM)
make clean	Removes any * and *.o files throughout the distribution.
make clobber	Removes all created files except the directories ParkBench/lib/$PVM_ARCH and ParkBench/bin/$PVM_ARCH.

A good strategy for those unfamiliar with this package would be the following sequence:

- edit ParkBench/make.local.def

- `make Low_Level`
- Run the `comms*` codes to test the setup.

Running the Benchmarks

All executables and their I/O files will be placed in the directory `ParkBench/bin/$PVM_ARCH`. Yes, even the MPI ones. Specific instructions as to how to set things up for use with PVM and MPI is contained in `ParkBench/README.pvm` and `ParkBench/README.mpi` and is duplicated here.

In case of the NPB benchmarks, the executables are compiled for the `Small` size problems only. To compile and run the codes for different problem sizes, read the file `ParkBench/NPB2.1/2.1README`. This package uses a different configuration file format than the ParkBench distribution. If your planning on building any of these on a regular basis, it would be a set up a `NPB2.1/config/suite.def` file that defines a suite of NPB benchmarks to build every time with set problem sizes. Read the `ParkBench/NPB2.1/2.1README` file for further information.

The Shallow Water Compact Application code, PSTSWM also defaults to the `Small` problem size and *only one task*. To run this code in parallel for different problem sizes, copy the files `ParkBench/Comp_Apps/PSTSWM/input/problem` and `ParkBench/Comp_Apps/PSTSWM/input_[p` to the directory `ParkBench/bin/$PVM_ARCH` that contains all the binaries. If you want to execute the default (really small) size problem that is compiled in, just remove the `problem` file from this directory.

PVM Instructions - `ParkBench/README.pvm`

To use this distribution with PVM, you need to do the following:

- Compile and install PVM. Make sure the `PVM_ROOT` environment variable is set and that `$PVM_ROOT/lib` is in your executable path. If you can't type `pvm` at your prompt *after login* and get the pvm console, then you haven't installed it properly. Note that PVM is very sensitive to what's in the user's `.login .cshrc` and `.profile` files.

- Build the ParkBench codes you wish to run as above.
- Modify the `ParkBench/pvmhostfile` to include a path to the ParkBench executables. See `ParkBench/pvmhostfile.example`.
- Possibly modify the `ParkBench/bin/$PVM_ARCH/*.dat` files to specify the number of PVM tasks. The defaults are different for every benchmark, so take a look.
- Start PVM with the `ParkBench/pvmhostfile` as an argument.

```
straw:ParkBench> pvm pvmhostfile
pvm> quit
pvmd still running.
```

- Add the hosts to your virtual machine. If they are of a different architecture, you had better make sure the ParkBench codes are built for them. Two hosts really is a minimum, you may or may not see deadlock, coredumps or floating point errors with less.

```
straw:ParkBench> pvm
pvmd already running.
pvm> add menkar
1 successful
                    HOST     DTID
                  menkar    80000
pvm> add dancer hydra1a
2 successful
                    HOST     DTID
                  dancer    c0000
                hydra1a   100000
```

- Run the codes.

```
straw:ParkBench> bin/SUN4/comm1
```

Native PVM - CM5 Example

If the front end needs to spawn the PVM process on the nodes, do the following:

-

- Build the ParkBench codes you wish to run as above. The benchmarks will be named "exename" for the node processes and "exename_host" for the front end.
- Start PVM and type "exename_host". The front end process will take care of the spawning.

PVM Dependencies

All of the PVM routines used within ParkBench are in file `ParkBench/Info.PVM.dependencies`. If your using a customized version of PVM, you had better make sure all the routines are in place, else risk undefined references at link time.

MPI Instructions - `ParkBench/README.mpi`

MPI on a Network of Workstations - MPICH

- Compile and install MPI, make sure that the `mpirun` that starts an MPI job is in your path on all machines. Modify `ParkBench/make.local.def` to point to the MPI libraries and includes.
- Build the ParkBench codes you wish to run as above.
- Create an MPI "machinefile" listing the hostnames of machines on which the MPI codes will be run, one per line. If your running the MPI codes heterogeneously, then consult the `mpirun` man page and go from there. Some installations of MPICH require at least five hosts be in the machine file, however duplication of hostnames is allowed.
- Modify the `*.dat` files in `ParkBench/bin/$PVM_ARCH` files to specify the number of MPI tasks to run. The default is different for most benchmarks, so take a look. None of the benchmarks spawn more than four tasks, unless you edit these files.
- Run the codes

```
straw:ParkBench> mpirun -np 2 -machinefile machines.sun4 ParkBench/bin/SUN4/comms1_mp
```

1 Native PVM/MPI on an SP2

There are two choices for the SP2. The first is to use PVM written on top of the optimized IBM MPI library, the second is to use the socket based PVM. Set your environment variable `PVM_ARCH` to `SP2MPI` for the MPI based implementation of PVM or `PVM_ARCH` to `RS6K` for the socket based one. If you have decided to use the `RS6K` architecture, you should change the definitions for MPI in the file `ParkBench/conf/make.def.RS6K` to those in the file `ParkBench/conf/make.def.SP2MPI`. Also you should put `master=0` and `spawn=0` in the corresponding `*.dat` files. Also, you have to spawn all your executables from the PVM console as the current MPI specification does not provide for this.

```
pvm> spawn -(number of processes) <executable\_name>
```

Contacts

- The UTK ParkBench Team at `parkbench-comments@cs.utk.edu` consists of:
  Michael Kolatis at `http://www.cs.utk.edu/k̃olatis`
  Philip J. Mucci at `http://www.cs.utk.edu/m̃ucci`
  Erich Strohmaier at `http://www.cs.utk.edu/ẽrich`
  Dr. Jack Dongarra `http://www.netlib.org/utk/people/JackDongarra.html`

Thanks to Pat Worley for his beta test of this release.

ParkBench Run and Report Rules

Here is a new version of the Run Rules for this release of ParkBench. What are the Run-Rules? These are guidelines that you should follow if you'd *ever* like to *make use* of your results. If you plan on doing anything related to information obtained through the use of a ParkBench executable, you should read this very carefully. This information can also be found in `ParkBench/runrules.draft6`.

Introduction

All results publicly disclosed must adhere to these, the *ParkBench Run and Report Rules*. The following are expected:

- Adherence to the criteria for use of compiler directives, source modifications, and compiling system flag selection.
- Availability of a full disclosure report.

Benchmark Compilation and Measurement

Baseline Runs

Optimizations as described below are allowed.

- Compile and load options

  Compiler or loader flags which are supported and documented by the supplier are allowed. These include porting, optimization, and preprocessor invocation. The flags should apply to a class of programs larger than the single benchmark.
- Libraries

  The BLACS, BLAS, LAPACK, MPI, PVM and ScaLAPACK libraries may be replaced with optimized versions subject to the following rules:
  - All libraries used shall be disclosed with the results submission. Each library shall be identified by library name, revision, and source (supplier). Libraries which are not generally available are not permitted unless they are made available by the reporting organization within 6 months.
  - Calls to library subroutines should have equivalent functionality to that in the released benchmark code. Code modifications to accommodate various library call formats are not allowed.

Optimized Runs

- Compiler Directives

Source code directives which are supported and documented by the supplier of the compiler may be used in the ParkBench benchmark codes. Language extensions instead of directives (such as `PARALLEL DO` instead of `C$DIR PARALLEL`) may also be used.

  - Loop level vectorization directives for systems with vector processors
  - Loop parallelism directives for multi-processor systems
  - Data distribution directives for distributed memory systems
    Compiler directives must not be uniquely applicable to the execution of the ParkBench benchmark codes.

- Code Modifications
  - Removal of parts of the code which are never executed.
  - Avoiding unnecessary executions of parts of the program.
  - Loop unrolling.
  - Loop blocking.
  - Changes of the data layout.
  - Calls to library subroutines.
- Limitations of Optimization
  - Optimized assembly modules.
    Substituting any part of the code by optimized assembly modules or modification of compiler generated assembly code or executables should be disclosed in reporting the results.
  - Code with limited calculation accuracy.
    The calculation should be carried out in full precision. However the substitution of algorithms is allowed (see Exchange of the used mathematical algorithm).
  - Exchange of the used mathematical algorithm.
    Any change of algorithms must be fully disclosed and is subject to review by the ParkBench Committee. Passing the verification test is a necessary condition for such an approval. The substituted algorithm must be as robust as the baseline algorithm.
  - Using knowledge of the solution.
    Any modification of the code or input data sets, which uses the knowledge of the solution or of the verification test, is not permitted.
  - Code to circumvent the actual computation
    Any modification of the code to circumvent the actual computation is not permitted.

- Special Requirements

  Additional restrictions for single benchmarks might be specified separately in the Readme files (eg. PSTSWM).

Disclosing and Publishing ParkBench Results

All ParkBench benchmark results should conform to the rules defined here in this this document. Full disclosure of all information defined in this section is necessary for acceptance and possible verification of any results.

Configuration Disclosure

A full disclosure consists of:

- System Identification
  - System model name
  - ParkBench version number
  - Test sponsor (name, location)
  - Test date (month, year)
- Configuration

  A description of a necessary and sufficient hardware and software configuration should be supplied. For example, this could include information about the CPU type, memory organization, operating system name and level, and names and versions of compilers and loaders.
- System Tuning Options

  Description of special system tuning, including any special OS parameters set, or changes to standard daemons, etc.
- Benchmark Code Tuning Options
  - Portability compiler/load flags used by any benchmark.
  - Source code directives used in any benchmark code.
  - Performance libraries used in creating any benchmark.

ParkBench Problem Sizes

Benchmark results are reported for each application and problem size. Up to five problem sizes are provided for each of the benchmarks.

The user may report a *table* of results for a particular problem size varying only the number of tasks in the benchmark *only if all of hardware and software used is identical. Otherwise, a full report must be generated for each run.*

Test Results Disclosure

The actual test results as produced by the benchmarks programs should be copied to a file. They consist of the elapsed times as measured by the program and verification information. See the specific instructions in the distribution of ParkBench, as to which timers are permitted.

Submitting Results to ParkBench Committee

Prior to public dissemination, all ParkBench results and configuration disclosures should be sent to the ParkBench Team via email at parkbench-comments@cs.utk.edu for certification. The ParkBench Team intends to publish verified gathered results periodically.

Changes since the last release

```
./doc/
./doc/README
./doc/README.mpi
./doc/README.pvm
./doc/Info.PVM.dependencies
./doc/Makefile
./doc/README.tex
./doc/runrules.tex
./doc/runrules6.tex
./doc/Changes
./doc/runrules.draft6
./Changes
./Comp_Apps/Makefile
./Comp_Apps/PSTSWM/makefile.mpi
./Comp_Apps/PSTSWM/makefile.pvm
./Kernels/LU_solver/LU_solver.dat
./Kernels/LU_solver/src/pdluinfo.f
./Kernels/MATMUL/MATMUL.dat
./Kernels/MATMUL/src/pdsummainfo.f
./Kernels/Makefile
./Kernels/QR/QR.dat
./Kernels/QR/src/pdqrinfo.f
./Kernels/TRANS/TRANS.dat
./Kernels/TRANS/src/pdtransinfo.f
./Kernels/TRD/TRD.dat
./Kernels/TRD/src/pdtrdinfo.f
./Low_Level/Makefile
./Low_Level/comms1/src_mpi/comms1.inc
./Low_Level/comms1/src_pvm/comms1.inc
./Low_Level/comms2/src_mpi/comms2.inc
./Low_Level/comms2/src_pvm/comms2.inc
./Low_Level/comms3/src_mpi/COMMS3.f
./Makefile
./NPB2.1/Makefile
./NPB2.1/sys/make.common
./README
```

```
./README.mpi
./README.pvm
./conf/Makefile
./conf/make.def.ALPHA
./conf/make.def.CM5
./conf/make.def.Generic
./conf/make.def.HPPA
./conf/make.def.RS6K
./conf/make.def.SGI5
./conf/make.def.SP2MPI
./conf/make.def.SUN4
./make.local.def
./pvmhostfile
./pvmhostfile.example
./runrules.draft6
```