The Deterministic Parallel Java Installation Manual Version 1.0

University of Illinois at Urbana-Champaign

Revised June 2010

This document explains how to install Deterministic Parallel Java (DPJ), so you can use it to compile and run programs. It also explains how to build DPJ from source code, in case you wish to study and/or modify the compiler internals. A working knowledge of Java is required. Please also consult *The Deterministic Parallel Java Tutorial* and *The Deterministic Parallel Java Language Reference Manual* for further details about the language.

The instructions in this document assume that you are working in a UNIX-like system (including Linux, Mac OS X, etc.). Everything should work in Windows, but we have not tested DPJ on Windows systems. To follow the instructions as stated here, you will have to install Cygwin on your Windows system to get a UNIX-like interface and tool set.

Contents

1 Before You Begin

Thank you for you interest in DPJ. This section covers the information you need to know before installing DPJ on your system.

1.1 Requirements

DPJ is based on the Java language and `javac` compiler from Sun Microsystems. This manual assumes that the reader is familiar with Java, including compiling and running Java programs. You will need at least a Java Runtime Environment (JRE) to run DPJ codes. If you wish to work on the compiler sources you will need a Java Development Kit (JDK) to build the compiler from source. We recommend the latest Sun JDK from `java.sun.com`.

1.2 Developer or User?

Instructions and requirements are slightly different depending on whether you want to use DPJ *(user install)* or develop DPJ *(developer install)*. A user install of DPJ includes the bytecode versions of the compiler and runtime, documentation, and the DPJ source code for the benchmarks. A developer install of DPJ includes everything from the user install, and adds the compiler and runtime sources and build environment.

Performing a user install is easier, but there are two reasons to do a developer install:

1. You are interested in reading or modifying the source code for the DPJ compiler and/or runtime; or
2. You want access to the latest updates to the DPJ compiler, without waiting for the next bytecode release.

1.2.1 Requirements for User Install

For a user install of DPJ, do the following:

1. Make sure you have a JRE, such as Java 2 Standard Edition (J2SE), installed. You must be using at least Java 6.
2. Follow the instructions in § 2 to do the installation and start using DPJ.

1.2.2 Requirements for Developer Install

For a developer install of DPJ, do the following:

1. Make sure you have a JRE (at least Java 6) and JDK installed.
2. Make sure that you have a working installation of Apache `ant`, the Java build tool.
3. Make sure you have a working installation of `git`, the version control system.

   `http://git.com`.
4. Make sure you have a working `make`.
5. The Eclipse IDE is invaluable for studying and modifying large Java programs, so we recommend installing it. See

   `http://www.eclipse.org`
6. Follow the instructions in § 3 to do the installation. Then read §§ 2.2 and 2.3 to learn how to use DPJ.

1.3 What's in the Release

The release directory structure contains the following directories:

- `Documentation` : Manuals and instructions for using and/or building DPJ.
- `Implementation`: The DPJ compiler and runtime.
- `Benchmarks`: Example code kernels and applications written for DPJ.

1.4 DPJ Resources

You should keep the following resources in mind as you work with DPJ:

1. The DPJ home page: `dpj.cs.illinois.edu`.
2. The DPJ public code repository: `http://github.com/dpj/DPJ.git`.
3. The DPJ development mailing list: `dpjdev@cs.uiuc.edu`. Joining the list allows you to follow major announcements, news, and technical discussions regarding DPJ. To subscribe to the list, please visit

   `http://lists.cs.uiuc.edu/mailman/listinfo/dpjdev`.
4. DPJ documentation is located in the `Documentation` directory of the DPJ release and is also available on the DPJ web site at `http://dpj.cs.illinois.edu/DPJ/Download.html`.

The DPJ development team appreciates your feedback and questions. Please check the email list archives before submitting a question or bug report. We prefer using the list for submissions so that the entire DPJ community can benefit from the growing knowledge base of questions and answers.

1.5 DPJ License

The DPJ software is subject to the following licenses:

- The DPJ compiler is based on Sun's `javac` compiler and is covered by the GNU General Public License version 2.
- The programs in the `Benchmarks/Applications` directory are based on codes written by various third parties, and are subject to their licenses.
- The rest of the code is by the University of Illinois and is released under the University of Illinois/NCSA Open Source License.

See the file `LICENSE.TXT` in the top-level directory of the DPJ software for further license information.

2 User Install

This section explains how do a binary installation of DPJ including the bytecode version of the DPJ compiler and runtime, and the source code for the DPJ benchmarks. If you install this way, then you can compile and run DPJ programs. However, you will not have access to the source code for the DPJ compiler. (The runtime source code is included in the user install, because you need that to compile against the runtime. See § 2.2.) If you want to study or modify the compiler and runtime source code, then you should install from the `git` source base, as described in § 3.

2.1 Installation and setup

To install the bytecode version of the compiler and runtime, do the following:

1. Get the DPJ binary install tarball from `http://dpj.cs.uiuc.edu`.
2. Unpack the tarball:

   ```
   tar -xvf dpjbin.tar
   ```
3. Set `DPJ_ROOT` and your `PATH`:

   ```
   setenv DPJ_ROOT ${HOME}/dpjbin
   setenv PATH ${PATH}:${DPJ_ROOT}/Implementation/bin
   ```

   This assumes that `dpjbin` is in your home directory; if not, make the necessary changes.

4. To check that you have a good installation, build the programs in the directory `Benchmarks/Kernels`:

   ```
   cd dpjbin/Benchmarks/Kernels
   make
   ```

5. Test the kernels:

   ```
   make test-all
   ```

You are now ready to compile and run DPJ programs.

2.2 Invoking the DPJ Compiler

You invoke the DPJ compiler by saying

```
dpjc
```

on the command line, followed by some DPJ files to compile, e.g.,

```
dpjc Foo.java Bar.java
```

The compiler translates source files written in DPJ to source files written in plain Java, that can be compiled to bytecode by an ordinary Java compiler (such as `javac`) and then run by a Java Virtual Machine. By default, the compiler translates DPJ's parallel constructs to Java parallel code using the `ForkJoinTask` library to express the parallelism. A command-line option (see below) can be used to generate sequential code.

Since `dpjc` is based on `javac`, you can use all the command-line options that `javac` supports. In addition, `dpjc` supports the following options:

- `-seq`: If this flag is present, then the compiler translates the DPJ sources into sequential Java code. By default, the compiler generates parallel code.
- `-instrument`: This option makes sense only with `-seq`. With this option turned on, the compiler adds instrumentation to the program, so that when run its performance characteristics can be analyzed. The instrumentation consists of method calls into the API defined by the `Instrument` class in the package `DPJRuntime`, located in `Implementation/Runtime`. See the DPJ runtime API documentation for further details.
- `-count`: When compiling the program, count the various kinds of DPJ annotations, and report the counts.

One important limitation of the DPJ compiler, as of DPJ v1.0, is that it has very limited support for separate compilation. For example, in ordinary Java, if `Foo.java` defines a class `Foo`, and `Bar.java` defines a class `Bar` that uses `Foo`, you can compile `Foo.java` to `Foo.class` separately, and later compile `Bar.java` to `Bar.class`, as long as `Foo.class` is in the class path specified on the compiler command line.

In DPJ, you can still do this if `Foo.java` is an ordinary Java file (i.e., it doesn't have any DPJ annotations). However, *any source files containing DPJ annotations must be compiled together with code that depends on them.* That is because DPJ's region and effect annotations are currently not represented in the bytecode (i.e., DPJ uses ordinary Java bytecode). Therefore, the compiler needs all the source files containing the DPJ annotations. In particular, in the example given above, if class `Foo` is defined with a region parameter, and you attempt to compile class `Bar` that uses `Foo` by linking against `Foo.class`, then the compiler will generate an error, saying that `Foo` doesn't take parameters. That's because the region parameter information is *erased* in the `Foo.class` bytecode. Instead, you need to compile `Foo.java` and `Bar.java` together, so the compiler can see the parameter. The same limitation applies to the classes in the DPJ runtime; see § 7 of *The Deterministic Parallel Java Language Reference Manual* for more details.

2.3 Running DPJ Programs

You run DPJ programs by saying

```
dpj
```

on the command line, followed by a Java class to execute, e.g.,

```
dpj Foo
```

The class, and any classes it depends on, must be in the class path. The options for setting the class path and the other runtime options are the same as for `java`. In fact, `dpj` just invokes the ordinary `java` with a few options, so you can use `java` to run DPJ programs if you want.

The DPJ runtime has several configurable parameters. These are set in one of two ways. First, they can be passed as command-line arguments to the program. The special DPJ command-line arguments must come first; they are processed by the runtime and then stripped from the argument list, which is passed to the main program. For example, the following invocation of the class `Foo` sets the DPJ foreach cutoff (explained below) to 100, then passes `42` and `bar` as the command-line arguments to the program:

```
dpj Foo --dpj-foreach-cutoff 100 42 bar
```

The following command-line options are processed specially by the DPJ runtime as stated above. Each of them must be followed by a numeric argument; if not, an error is reported. If the options are not present, then the default is used as stated below.

- `--dpj-foreach-split` n: Set the branching factor used to split a `foreach` loop to n. The loop is recursively split into this many branches in each iteration, until the cutoff is reached (see below). The default is 2.
- `--dpj-foreach-cutoff` n: Set the minimum number of `foreach` iterations allocated to a single task to n. Beyond this point, no more parallel splitting of a foreach loop occurs. The default is 128.
- `--dpj-num-threads` n: Set the number of worker threads used to run the program to n. The default is the number of available processors given to the java virtual machine.

The second way to set the runtime parameters is to assign to them from within the program. This is useful if you want different `foreach` loops in your program to use different parameters. See the runtime API documentation (available on the DPJ web site) for information about how to do this.

3 Developer Install

3.1 Installation and setup

To install the DPJ compiler and runtime from source, do the following.

1. Check that your system meets the requirements in § 1.1.
2. Get the source code distribution:

   ```
   cd ~
   git clone git://github.com/dpj/DPJ.git DPJ
   ```

   This assumes you are calling the working directory `DPJ` and putting it in your home directory. If not, make the appropriate adjustments.
3. Set the `DPJ_ROOT` environment variable:

   ```
   setenv DPJ_ROOT ~/DPJ
   ```

4. Tell `ant` to build the compiler, using your JDK:

   ```
   cd DPJ/Implementation/Compiler/make
   ant -Dboot.java.home=/usr/lib/jvm/java
   ```

   If your JDK is not located at `/usr/bin/jvm/java`, then substitute the appropriate path.

5. A successful compiler build puts `jar` files for the compiler and related tools (such as `javadoc`) in

   `Implementation/Compiler/build`.

   The executable files `dpjc` and `dpj` in `Implementation/bin` invoke these `jar` files.

6. Add the DPJ compiler bin to your `PATH`:

   ```
   setenv PATH ${PATH}:${DPJ_ROOT}/Implementation/bin/
   ```

7. Build the runtime classes located at `${DPJ_ROOT}/Implementation/Runtime`. You can expect some warnings.

   ```
   cd ../../Runtime/
   make
   ```

8. You can now build and run the kernels:

   ```
   cd ${DPJ_ROOT}/Benchmarks/Kernels
   make
   ```

9. Then run all the kernel tests:

   ```
   make test-all
   ```

3.2 Testing the Installation

1. JUnit 4 tests are in `${DPJ_ROOT}/Implementation/Compiler/test/dpj-junit-tests`. The corresponding DPJ source files are in `test/dpj-programs`.

2. The JUnit tests are designed to be run from within Eclipse, by running the launcher file

   `test/dpj-junit-tests/dpj-junit-tests.launch`.

   If the JUnit tests fail with a `MethodNotFound` error, it is probably because the classpath is wrong. Many of the javac classes are included in the system library (`rt.jar`), and so the DPJ versions must precede the system library in the classpath. If you use the launcher, this error should not happen.

3.3 Browsing and Modifying the Compiler Code

After an initial `ant` build, DPJ javac will build automatically in Eclipse (albeit to a `bin/` directory instead of `build/`). However, if any of the resource bundles are changed (be sure to change the `.properties` files in

`src/share/classes/com/sun/tools/javac/resources`

and not the generated `.java` files!), the compiler must be re-built using the `ant` script.

The `ant` script bootstraps the compiler. So if there is an internal compiler error, it is due to a fault in the DPJ compiler. Note also that the compiler may compile successfully in Eclipse but fail when it is bootstrapped in the `ant` script due to differences between pure Java and DPJ (or an error in the DPJ compiler). In particular, note that the following keywords are reserved in DPJ and therefore may not be used as variable names (in any DPJ program, including the DPJ compiler):

- `under`
- `region`
- `reads`
- `writes`