Pyjama (PJ) help - v1

March 6, 2017

This help documentation is used for Pyjama version from v1.2.0 to v1.5.4.

1 Using the Pyjama compiler

Pyjama v1.x.x compiler can be used to compile a .pj file into a paralleled .java file. Then users can use standard javac compiler to compile this .java file into runnable .class file.

You should write your Java code inside a file with extension *.pj, and eventually the corresponding *.java file will be produced (which you can compile using the standard Java compiler). Therefore, when you try and run your code, make sure you are executing the *.java file as opposed to the *.pj file you are editing. You should never modify the produced *.java files, since your changes will be lost when you compile the original *.pj files using the OpenMP compiler:

```
> java -jar pyjama.jar:. package/MyPJClass.pj
```

The above will produce a MyPJClass.java file in the same directory. You then compile this file using the standard Java compiler:

```
> javac -cp pyjama.jar:. package/MyPJClass.java
```

And run it:

```
> java -cp pyjama.jar:. package.MyPJClass
```

- PJPlugin

Also, you may download Pyjama Eclipse Plugin to help you automatically convert your .pj file into .java file.

PJPlugin can be download from Update Site URL:

>http://www.ece.auckland.ac.nz/~parallel/plugins/PyjamaEclipsePlugin/

Notice that Pyjama and PJPlugin have different vesion control: current PJPlugin release version is 1.4.0.201603022036, integrated lib file is pyjama v1.5.3.

If you want to use latest Pyjama version along with PJPlugin, download latest Pyjama file from

>http://homepages.engineering.auckland.ac.nz/~parallel/ParallelIT/downloads/Pyjama/

then substitute old pyjama.jar file with new one. Please remember rename your downloaded jar file name from “Pyjama-1.x.x.jar” to “pyjama.jar”, otherwise PJPlugin cannot recognize library name then cannot automatically convert file for you.

- Classpath and library functions

You should include the pyjama.jar file in your classpath(If you used PJPlugin to create a pyjama project, plugin should already do this for you). The main class that you might interact with inside your code is Pyjama (e.g. to get thread ID Pyjama.omp_get_thread_num() etc).

- Refreshing

Sometimes you might get errors when you introduce new clauses or directives. What happens, is the OpenMP compiler introduces new classes sometimes. Therefore, your classpath might need to be updated to include those directories. The simplest approach, if you are using an IDE such as Eclipse, is that your can refresh the source folder view so that Eclipse sees those new classes. That way, your generated *.java files will be able to compile too. Also, whenever you modify the *.pj file, you of course need to run Generate again, and refresh the *.java file so that Eclipse sees the update.

Sometimes you may find plugin cannot convert file for you. This probably for reason that there are some semantic error in your *.pj file. Currently plugin cannot tell you where you made the mistake. Double check your pj code and rectify errors may solve this problem.

2 Specifications

The current implementation of the Java OpenMP compiler only contains support for some of the most common and important features of OpenMP. The following directives are believed to be fairly stable. With the current implementation, the ordering of some of the optional clauses matters (but in a full implementation, the ordering shouldn't matter). The ordering expected is specified. anything in square brackets is optional:

- //#omp atomic
- //#omp parallel ["if" "(" expression ")"] [dataClauseList]
- //#omp for [dataClauseList] [schedule] ["ordered"] ["nowait"]
- //#omp parallel for ["if" "(" expression ")"][dataClauseList] [schedule] ["ordered"] ["nowait"]
- //#omp ordered
- //#omp section
- //#omp sections [dataClauseList] ["nowait"]

- //#omp parallel sections [“async”] [“if” “(“ expression “)”] [dataClauseList] [“nowait”]
- //#omp single [dataClauseList]
- //#omp master
- //#omp critical [identifierName]
- //#omp barrier
- //#omp flush [“(“ argumentList “)”]
- //#omp freeguithread [parallel]
- //#omp target virtual (from v1.5.4)
- //#omp async-call (from v1.5.4)

Some of the above non-terminals are expanded below:

- dataClauseList -> (dataClause) +
- dataClause -> “firstprivate” “(“ argumentCopyList “)” | “lastprivate” “(“ argumentList “)” | “shared” “(“ argumentList “)” | “reduction” “(“ reductionOperator, argumentCopyList “)”

When using firstprivate dataclause for class type variables, you should explicitly define constructor that comfort with ClassName(ClassName C);

- schedule -> “schedule” “(“ kind [“,” chunk] “)”
- kind -> “static” | “dynamic” | “guided”
- chunk -> integer constant or integer variable
- argumentList -> variableName (“,” variableName)*
- argumentCopyList -> copyArgPair (“,” copyArgPair)*
- copyArgPair -> [copyVariable “#”] variableName
- copyVariable -> an instance of java_omp.Copy<T>
- reductionOperator -> “+” | “*” | userDefinedReduction
- userDefinedReduction -> function name should conform with <T> reductionFunctionName(<T> var1, <T> var2);

The follow are not supported or implemented yet:

- //#omp threadprivate
- //#omp dataclause: private

3 More detailed usage of using customized reduction operation with Pyjama (and Redlib)

Pyjama supports high-level reduction operation which traditional OpenMP does not provide. Programmers can define their own function to perform reduction onto a specified parameter.

Similar to primitive data reduction, a reduction data clause is followed with the operator and its related variable. For example, //#omp parallel reduction(+:a) will perform a plus reduction after all threads finish their own executions. For a customized reduction, programmers can simply replace the binary operator to a function name, e.g. //#omp parallel reduction(yourReductionFunction:a). However, the function definition should in conform with the following format:

- T reductionFunctionName(T var1, T var2)

This function takes two parameters with same data type T, and return the reduced value with type T. For example:

```
public class xPoint{
    private int x;
    private int y;
    public static void main (String[] argc) {
        xPoint p = new xPoint(0,0);
        //#omp parallel reduction(xPointReduction:p)
        {
            //parallel region code
        }
    }
    public xPoint xPointReduction(xPoint p1, xPoint p2) {
        // user defined reduction operation here
        ...
    }
    ...
}
```

Should be noticed that, if your reduction function is another class function call, you should also tell Pyjama compiler which class the function belongs to. Say ClassA.reductionFunction.

You can also use Redlib build-in reduction operations to facilitate your programming. A simple example using Redlib and Pyjama is here:

```
public Class T SetUnion implements Reduction<Set<T>>{
    public Set<T> reduce (Set<T> first , Set<T> second){
        for (T t : second){
            first.add(t);
        }
    return first;
    }
}
...
double[] array = ...;
Set<Point> set = new Set<point >();
//#omp parallel for shared(array) reduction(SetUnion.reduce:set)
```

```
for(int i=0; i<max; i++) {
    Point p = computation(array[i]);
    set.add(p);
}
...
```

4 Limitations

Since Pyjama is a research project, there are several limitations both for Pyjama compiler and Pyjama runtime support. Hereby we list all the limitations and disability Pyjama has as far we known.

1. You may find that Pyjama is poor for detecting errors in your source code when compiling. When Pyjama compiler encounters the first token it does not expect, it will throw a parsing exception and stop the compiling. Because current Pyjama parser is very weak in error recovery. What you can do is double check your code before compiling or use our Pyjama plugin for eclipse IDE, which can detect several type of errors on the fly when you are coding.

2. Currently in Pyjama, *//#omp atomic* directive is implemented the same as *//#omp critical*. There is no difference between these two. In another word, in Pyjama now, OpenMP atomic is implemented as lock based manipulation on variable, which is the exact same way how critical directive implemented. Because in java language, source code cannot be directly converted into machine code, in which atomic hardware synchronization instructions can be used (CMPXCHG or LL/SC). Java standard library provide atomic data type for people to use non-block data update, such as AtomicInteger class. This may be the substitution for real atomic data operation in your program.

3. Pyjama does not support nested parallel. When nested parallel regions are used, only the outermost parallel region will be activated. In the further Pyjama will concern about nested parallel activation. But for now, runtime only activate once when it encounter different level of parallel regions.

4. There are limitations that class data fields are not allowed to appear in any data clauses. Class data fields variables, including static and non-static, are in nature shared. In parallel region, class data fields can be used, but proper synchronization is necessary.

5. One limitation in *//#omp parallel for* directive, if *num_ threads(n)* clause is going to be used, please put it before any data clauses and schedule clause (including shared, firstprivate, lastprivate, schedule).

6. The global lock, when using critical directive, identifier is invalid, this limitation may confine the fine grained locking for several experiments.

7. There could be compiling error when using //#omp for directive. Sometimes, the expression like `for(int i=0;i<n-1;i++)` cannot go through compiling, try to use `for(int i=0;i<(n-1);i++)`, by adding brackets. This problem occurs because of current defect of Pyjama compiler.

5 Contact

It would be great if you could send code that fails to do as you expect, please see contacts below. If you receive any unimplemented feature exceptions, or any exceptions that you feel should be working but aren't, it would be great to hear from you so that they can be corrected. Thanks.

If you have any further questions, or you find some problems, suggestions or comments about the Java OpenMP compiler, please email fxin927@aucklanduni.ac.nz or n.giacaman@auckland.ac.nz or o.sinnen@auckland.ac.nz.