Pyjama (PJ) help - v2

March 6, 2017

This help documentation is used for Pyjama version from v2.0.0.

1 Using the Pyjama compiler

In default, Pyjama v2.x.x compiler expects a standard .java file, and it compiles the .java file into the bytecode (.class file). In the .java file, users can use any comment-like OpenMP directives (see all Pyjama supported directives in Section 2) to parallel the original sequential Java code. In another word, if users compile the Java code by a standard javac compiler, the program will run sequentially. If the Java code is compiled by Pyjama compiler, the program will run in a parallel way.

compiling options [v2.x.x]

From version 2.x.x, Pyjama compiler provides more compiling options. In default, the compiler expects a .java file which contains OpenMP-like comment. Moreover, users can specify other input file and output file options.

usage: java -jar Pyjamav2.0.0.jar [options]

-cp, --classpath <PATH> specify where to find user class files and annotation processors;

-d, --directory <DIR> output file directory;

-h, --help print usage of Pyjama compiler;

-j2c, --javatoclass (default) compile .java file to paralleled .class file;

-j2j, --javatojava compile .java file to paralleled .java file. Remember new parallel java file will overwrite old sequential java file, if there is no target directory specified;

-p2c, --pjtoclass compile .pj file to paralleled .class file;

-p2j, --pjtojava compile .pj file to paralleled .java file.

No IDE plugin supported

Unfortunately, Pyjama v2 does not have a paired IDE plugin at the moment, but programmers can easily use scripts to compile a Java project by calling the Pyjama command line.

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
- //#omp async-call

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.

8. When using *//#omp target virtual* directives, it is not allowed to return any value in the target virtual construct. It may cause compiling error or make unexpected execution behavior.

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.