Pyjama (PJ) help

March 8, 2011

1 Using the Java OpenMP compiler

You should write your Java code inside a file with extension *.javamp, 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 *.javamp file you are editing. You should never modify the produced *.java files, since your changes will be lost when you compile the original *.javamp files using the OpenMP compiler:

```
> java -jar pyjama.jar:. package/MyClass.javamp
```

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

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

And run it:

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

Classpath

You should include the pyjama.jar file in your classpath. The main class that you might interact with inside your code is java_omp.Pyjama (e.g. to get thread IDs, 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 *.javamp file, you of course need to run Generate again, and refresh the *.java file so that Eclipse sees the update.

2 Known problems and limitations

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 *[*"async"*]* *[*"if" "(" *expression* ")"*]* *[dataClauseList]*
- //#omp for *[dataClauseList]* *[schedule]* *[*"ordered"*]* *[*"nowait"*]*
- //#omp parallel for *[*"async"*]* *[*"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* ")"*]*

Some of the above non-terminals are expanded below:

- dataClauseList -> (dataClause) +
- dataClause -> "private" "(" *argumentList* ")"
  | "firstprivate" "(" *argumentCopyList* ")"
  | "lastprivate" "(" *argumentList* ")"
  | "shared" "(" *argumentList* ")"
  | "reduction" "(" *reductionOperator*, *argumentCopyList* ")"
- 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 -> an instance of `paratask.runtime.Reduction<T>`

The follow are not supported or implemented yet:

- //#omp threadprivate
- Nesting of parallel clauses, etc. Although the compiler *might* accept some nesting, the runtime will ignore much of the nesting (e.g. nesting a paralel region within another parallel region will not cause a new team of threads to be created).
- You currently cannot specify the size of the team of threads for different areas. Unfortuntely, the current implementation will always create a team of threads equal to the number of detected processors.

In regards to unstable features, the biggest known problem is using data clauses with the above directives (e.g. parallel, for, etc). In most cases, if you use simple data clauses (e.g. declare a variable as shared amongst a team), then things will compile. However, if you try more complex things, like declaring it as shared in an outer clause, and then try to redefine it to private within a nested clause, then things will get ugly. So, you might have to rewrite your code to keep things as simple as possible and ideally minimise the use of data clauses.

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.

3 Contact

If you have any further questions, or you find some problems, suggestions or comments about the Java OpenMP compiler, please email ngia003@aucklanduni.ac.nz and o.sinnen@auckland.ac.nz.