MASS Preprocessor

Sean Wessels

Computing and Software Systems, University of Washington, Bothell

Table of Contents

Introduction 3

ExchangeBulk Optimization 3

Reflection Optimization 3

Executing the MASS preprocessor 5

Files generated by the MASS preprocessor 5

Building MassPrePro 6

Changes to logic 6

Changes to grammar 6

Implementation 7

Limitations and Issues 7

Test Program Execution 9

Wave2D 9

Computational Fluid Dynamics (CFD) 9

Files included in MassPrePro (~/SensorGrid/MASS/preprocessor) 11

Introduction

Given a MASS program that contains an Agents or Places constructor, the preprocessor attempts to perform two distinct optimizations which will result in a functional and efficient MASS program.

The following two constructor types are recognized:

```
1. Places(int handle, String classname, Object argument, int…
   size)
2. Places(int handle, String primitiveType, String classname,
   Object argument, int…size)
```

ExchangeBulk Optimization

The first optimization replaces exchangeBulk() method calls with exchangeAll()/callAll() pairs. Method calls from a MASS variable with the following format:

```
exchangeBulk( handle, array, neighbors );
```

are replaced by calls with the following format:

```
exchangeAll( handle, "exchangeArray", neighbors );
callAll( "putArray" );
```

If the exchangeArray() and putArray() methods do not exist in the code being optimized, simple stub methods are created. As an example of the format of the methods being created, an exchangeBulk(1, P, neighbors) results in the following accessors:

```
public Object exchangeP( Object src ) {
      return (Object)P.getBoundary( (int[])src );
}

public Object putT( Object arg ) {
      T.putBoundary( inMessages );
      return null;
}
```

Reflection Optimization

The second, reflection optimization, modifies the method calls from MASS variables to use constant integer values corresponding to methods in place of string arguments which are resolved using Java reflection. It will append a callMethod() method to map the integer values to their corresponding methods. If no Agents or Places are recognized, the input program will be output unaltered.

The following methods are recognized and modified when called from a Places or Agents variable with an appropriate String parameter:

```
callAll()

callSome()

exchangeAll()

exchangeSome()

exchangeBulk()
```

Each string argument found in these methods at a position where a function is expected, is replaced with an integer constant. These constants are unique for each function being replaced and take the following form:

```
public static final int [functionName]_ = n;
```

In practice, the following method calls:

```
exchangeAll( handle, "exchangeArray", neighbors );
callAll( "putArray" );
```

would be altered to:

```
cubicles.exchangeAll( 1, exchangeP_, neighbors );
cubicles.callAll( putP_ );
```

and additional supporting code would be added to the end of the class:

```
public Object callMethod( int funcId, Object args ) {
      switch( funcId ) {
            case exchangeP_ : return exchangeP( args );
            case putP_ : return putP( args );
      }
      return null;
}

public static final int exchangeP_ = 0;
public static final int putP_ = 1;
```

The actual return types of callMethod() is determined by the initial MASS constructor. Constructor #1 from the list above, will yield the preceding callMethod. If constructor #2 had been used with “int” as the primitiveType, the following callMethod() would be generated:

```
public int callMethod( int funcId, int[] size, int[] index, int[]
        wave, int arg ) {
    switch( funcId ) {
        case exchangeP_ : return (int)exchangeP( size, index,
                wave, arg );
        case putP_ : return (int)putP( size, index, wave, arg
                );
    }
    return 0;
}
```

Executing the MASS preprocessor

From a Linux or Windows command prompt:

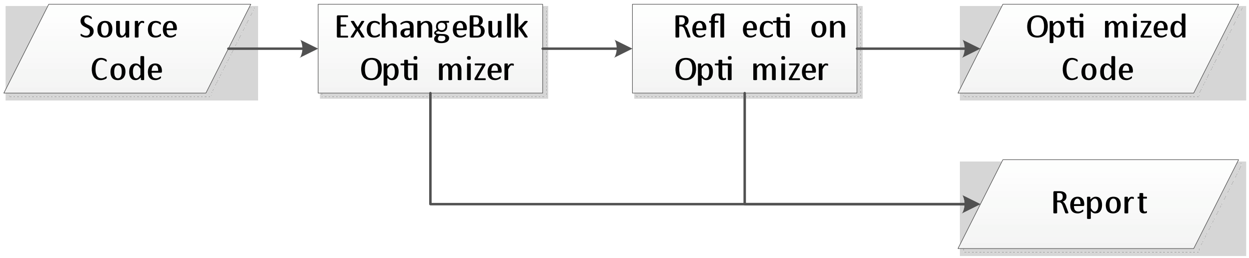
```
java MassPrePro <inputFile >outputFile
```

where `inputFile` is the MASS program coded using reflection and `outputFile` is the translated MASS program. Alternatively, the inputFile may be passed as a parameter:

```
java MassPrePro inputFile >outputFile
```

In both cases, code output is directed to stdout.

Execution adheres to the following sequence:



Files generated by the MASS preprocessor

The report and intermediate files generated by the preprocessor are located in the directory from which the preprocessor was executed.

_prepro_debug.txt	A report file which details the modifications made to the input source code by the preprocessor. It also includes listings of variables/fields recognized by the preprocessor by scope.
_prepro_stage_1.java	A temporary file containing the output of the first pass of the preprocessor.

Building MassPrePro

Changes to logic

If changes were only made to OptimizingVisitor, ScopeManager, MethodVar, MassVar, or MassPrePro, then only the classes that have been modified need to be rebuilt. Compilation is straightforward:

```
javac MassPrePro.java
```

Changes to grammar

However, when changes must made to the Java grammar, additional tools are required to build the MASS preprocessor. JavaCC 5.0 and JJTree are both available from the JavaCC website:

> http://javacc.java.net

Building the executable from the source can be a multi-step process depending on the changes that were made. Most files in the project have been auto-generated by JJTree or JavaCC. Do not attempt to the change the auto-generated files directly. See below for a complete list of project files.

If changes have been made to Java1.1.jjt, the following steps must be taken to update the project:

1. Use "jjtree Java1.1.jjt" in Linux or "jjtree.bat Java1.1.jjt" in Windows to generate Java1.1.jj

2. Use "javacc Java1.1.jj" in Linux or "javacc.bat Java1.1.jj" in Windows to generate *.java files.

3. If new rules were added to Java1.1.jjt, the correspondingly methods must be added to UnparseVisitor.java. Rules that have been removed from Java1.1.jjt should be removed from UnparseVisitor.java.

4. Once the methods have been added to UnparseVisitor.java, they can be overridden in ReflectionOptimizer.java or ExchangeBulkOptimizer if desired.

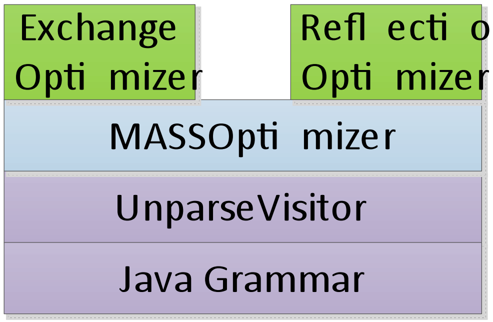
5. Remove existing *.class files and rebuild. (In Eclipse, select Project:Clean to rebuild the class files.)

Implementation

The preprocessor performs its optimizations by running the input code through a Java parser. The parser emits tokens in response to the input code. Actions act on specific tokens to check conditions, set flags, modify output, etc.

A grammar (Java1.1.jjt) defines a roughly correct version of Java. The grammar has been modified to create Abstract Syntax Trees. From the grammar, a parser is generated (UnparseVisitor) which by default will output any input which matches the Java language as defined by the grammar.

The parser methods can be overridden with MASSOptimizer, ExchangeBulkOptimizer, or ReflectionOptimizer to perform MASS optimizations. For example, when parsing a MethodDeclaration token, a flag will be set to indicate that a new method is being parsed and that a new scope must be placed on the stack. Subsequently, if a ResultType token is parsed while the MethodDeclaration flag remains set, the return type of the method being parsed can be recorded.



In general, flags are set in the various OptimizingParser.visit() methods. The logic to respond to those conditions occurs in the OptimizingParser.find(Token) method.

Limitations and Issues

The preprocessor has been tested on a limited set of MASS programs that were manually converted from their existing format to use Java reflection.

MASS programs that are to be preprocessed should not contain a method named "`callMethod`" or use a trailing underscore ("_") as part of a method name. The preprocessor will append an underscore to method names as part of the callMethod() generation. If the preprocessor encounters any naming conflict, it will report an error and halt.

All methods that are to be called as part of the MASS Agents or Places method must accept the same set of parameters in the same order as defined by the MASS constructor.

It is not guaranteed that all MASS variables will be recognized as such. Straightforward variable declarations will be identified, but variable assignment through casting or other classes may not be. Additionally, it cannot be guaranteed that non-MASS method calls which are identical to MASS method calls will not be altered by the optimizer.

The preprocessor has been tested against a very limited set of Java programs. When the parser encounters a language construct that is not defined in its grammar file, it will crash.

More testing against valid Java programs should be done. Java programs for validating a Java compiler can be found in the openjdk project at http://openjdk.java.net (openjdk/langtools/test/tools/javac)

Test Program Execution

Included as part of the MASS preprocessor are modified versions of Wave2D and CFD (Computational Fluid Dynamics).

All tests were executed on a 64-bit Windows 7 machine with an Intel Core i5 M430@2.27GHz.

Wave2D

Compiled and executed with MASS-Thread.

```
javac -cp MASS-Thread; Wave2DMass.java
```

```
java -cp MASS-Thread.jar; Wave2DMass 100 1000 100 1 2
```

Original Program	Time (ms)	exchangeAll (ms)	callAll (ms)
1	10187	4728	1590
2	9719	4746	1384
3	9500	4321	1667
4	9516	4626	1381
Average	**9730.5**	**4605.25**	**1505.5**

Modified for preprocessor	Time (ms)	exchangeAll (ms)	callAll (ms)
1	10343	4371	1900
2	9578	4275	1683
3	9313	4302	1593
4	9906	4517	1647
Average	**9785**	**4366.25**	**1705.75**

Computational Fluid Dynamics (CFD)

Compiled and executed with MASS-Thread.

```
run.bat 10 10 2
```

Original Program	Time (ms)
1	16536
2	14618
3	14384
4	13587
Average	**14781.25**

Modified for preprocessor	Time (ms)
1	13552
2	15179
3	14134
4	15850
Average	**14678.75**

Files included in MassPrePro (~/SensorGrid/MASS/preprocessor)

Directory	Filename	Description
MASSPrePro	MassPrePro.java	Executable front-end that takes a MASS program from stdin and outputs an optimized version to stdout.
	Java1.1.jjt	A grammar for the Java language. Originally based on 1.1 and extended to handle some newer Java syntax.
	MassVar.java	Stores MASS variables (Agents, Places).
	MethodData.java	Holds data related to the method currently being parsed.
	MASSOptimizer	Extends UnparseVisitor to provide common functionality to other optimizers.
	ReflectionOptimizer.java	Extends MASSOptimizer to provide preprocessor reflection functionality. Creates callMethod() and all required class constants. Provides the logic at the parser token level. Controls output based on parser tokens.
	ExchangeOptimizer.java	Extends MASSOptimizer to substitute exchangeBulk() calls into exchangleAll()/callAll() pairs. Creates get and put accessor function for the array given as an argument in exchangeBulk(). Provides the logic at the parser token level. Controls output based on parser tokens.
	ScopeManager.java	Store the variables associated with a particular scope. Adds and removes scopes from the scope-stack.
	UnparseVisitor.java	Provides basic functionality for all parser tokens in the Java1.1.jj grammar.
	SimpleNode.java	Implements Node
	Java1.1.jj	Auto-generated by JJTree from Java1.1.jjt.
	ASTAdditiveExpression.java	Auto-generated by JJTree
	ASTAllocationExpression.java	Auto-generated by JJTree
	ASTAndExpression.java	Auto-generated by JJTree
	ASTArgumentList.java	Auto-generated by JJTree
	ASTArguments.java	Auto-generated by JJTree
	ASTArrayDimsAndInits.java	Auto-generated by JJTree
	ASTArrayInitializer.java	Auto-generated by JJTree
	ASTAssignmentOperator.java	Auto-generated by JJTree

	ASTBlock.java	Auto-generated by JJTree
	ASTBlockStatement.java	Auto-generated by JJTree
	ASTBooleanLiteral.java	Auto-generated by JJTree
	ASTBreakStatement.java	Auto-generated by JJTree
	ASTCastExpression.java	Auto-generated by JJTree
	ASTCastLookahead.java	Auto-generated by JJTree
	ASTClassBody.java	Auto-generated by JJTree
	ASTClassBodyDeclaration.java	Auto-generated by JJTree
	ASTClassDeclaration.java	Auto-generated by JJTree
	ASTCompilationUnit.java	Auto-generated by JJTree
	ASTConditionalAndExpression.java	Auto-generated by JJTree
	ASTConditionalExpression.java	Auto-generated by JJTree
	ASTConditionalOrExpression.java	Auto-generated by JJTree
	ASTConstructorDeclaration.java	Auto-generated by JJTree
	ASTContinueStatement.java	Auto-generated by JJTree
	ASTDoStatement.java	Auto-generated by JJTree
	ASTEmptyStatement.java	Auto-generated by JJTree
	ASTEqualityExpression.java	Auto-generated by JJTree
	ASTExclusiveOrExpression.java	Auto-generated by JJTree
	ASTExplicitConstructorInvocation.java	Auto-generated by JJTree
	ASTExpression.java	Auto-generated by JJTree
	ASTFieldDeclaration.java	Auto-generated by JJTree
	ASTForEach.java	Auto-generated by JJTree
	ASTForEachStatement.java	Auto-generated by JJTree
	ASTForInit.java	Auto-generated by JJTree
	ASTFormalParameter.java	Auto-generated by JJTree
	ASTFormalParameters.java	Auto-generated by JJTree
	ASTForStatement.java	Auto-generated by JJTree
	ASTForTraditional.java	Auto-generated by JJTree
	ASTForUpdate.java	Auto-generated by JJTree
	ASTIdentifier.java	Auto-generated by JJTree
	ASTIfStatement.java	Auto-generated by JJTree
	ASTImportDeclaration.java	Auto-generated by JJTree
	ASTInclusiveOrExpression.java	Auto-generated by JJTree
	ASTInitializer.java	Auto-generated by JJTree
	ASTInstanceOfExpression.java	Auto-generated by JJTree
	ASTInterfaceDeclaration.java	Auto-generated by JJTree
	ASTInterfaceMemberDeclaration.java	Auto-generated by JJTree
	ASTLabeledStatement.java	Auto-generated by JJTree
	ASTLiteral.java	Auto-generated by JJTree
	ASTLocalVariableDeclaration.java	Auto-generated by JJTree
	ASTMethodDeclaration.java	Auto-generated by JJTree
	ASTMethodDeclarationLookahead.java	Auto-generated by JJTree
	ASTMethodDeclarator.java	Auto-generated by JJTree
	ASTMultiplicativeExpression.java	Auto-generated by JJTree
	ASTName.java	Auto-generated by JJTree

	ASTNameList.java	Auto-generated by JJTree
	ASTNestedClassDeclaration.java	Auto-generated by JJTree
	ASTNestedInterfaceDeclaration.java	Auto-generated by JJTree
	ASTNullLiteral.java	Auto-generated by JJTree
	ASTPackageDeclaration.java	Auto-generated by JJTree
	ASTPostfixExpression.java	Auto-generated by JJTree
	ASTPreDecrementExpression.java	Auto-generated by JJTree
	ASTPreIncrementExpression.java	Auto-generated by JJTree
	ASTPrimaryExpression.java	Auto-generated by JJTree
	ASTPrimaryPrefix.java	Auto-generated by JJTree
	ASTPrimarySuffix.java	Auto-generated by JJTree
	ASTPrimitiveType.java	Auto-generated by JJTree
	ASTRelationalExpression.java	Auto-generated by JJTree
	ASTResultType.java	Auto-generated by JJTree
	ASTReturnStatement.java	Auto-generated by JJTree
	ASTShiftExpression.java	Auto-generated by JJTree
	ASTStatement.java	Auto-generated by JJTree
	ASTStatementExpression.java	Auto-generated by JJTree
	ASTStatementExpressionList.java	Auto-generated by JJTree
	ASTStringLiteral.java	Auto-generated by JJTree
	ASTSwitchLabel.java	Auto-generated by JJTree
	ASTSwitchStatement.java	Auto-generated by JJTree
	ASTSynchronizedStatement.java	Auto-generated by JJTree
	ASTThrowStatement.java	Auto-generated by JJTree
	ASTTryStatement.java	Auto-generated by JJTree
	ASTType.java	Auto-generated by JJTree
	ASTTypeDeclaration.java	Auto-generated by JJTree
	ASTUnaryExpression.java	Auto-generated by JJTree
	ASTUnaryExpressionNotPlusMinus.java	Auto-generated by JJTree
	ASTUnmodifiedClassDeclaration.java	Auto-generated by JJTree
	ASTUnmodifiedInterfaceDeclaration.java	Auto-generated by JJTree
	ASTVariableDeclarator.java	Auto-generated by JJTree
	ASTVariableDeclaratorId.java	Auto-generated by JJTree
	ASTVariableInitializer.java	Auto-generated by JJTree
	ASTWhileStatement.java	Auto-generated by JJTree
	JavaCharStream.java	Auto-generated by JavaCC
	JavaParser.java	Auto-generated by JavaCC
	JavaParserConstants.java	Auto-generated by JavaCC
	JavaParserTokenManager.java	Auto-generated by JavaCC
	JavaParserTreeConstants.java	Auto-generated by JavaCC
	JavaParserVisitor.java	Auto-generated by JavaCC
	JJTJavaParserState.java	Auto-generated by JavaCC
	Node.java	Auto-generated by JJTree
	ParseException.java	Auto-generated by JavaCC
	Token.java	Auto-generated by JavaCC
	TokenMgrError.java	Auto-generated by JavaCC

demo/CFD	Flow.java	changed to use modified Solver3D
	MASS.jar	unchanged
	Mesh.java	unchanged
	compile.sh	build on Linux
	run.bat	run on Windows
	run.sh	run on Linux
demo/CFD/FDM	BasicInfo.java	unchanged
	Global.java	unchanged
	Grid.java	unchanged
	MatrixBase.java	unchanged
	MatrixP.java	unchanged
	MatrixT.java	unchanged
	MatrixV.java	unchanged
	MatrixV3D.java	unchanged
	Solver3D.java	Modified to include Cubicle.java and use preprocessor
demo/Wave2D	MASS-Thread.jar	unchanged
	Wave2DMass.java	output of preprocessor
	Wave2DMassModified.java	modified to use preprocessor
	Wave2DMassOrig.java	the original Wave2D