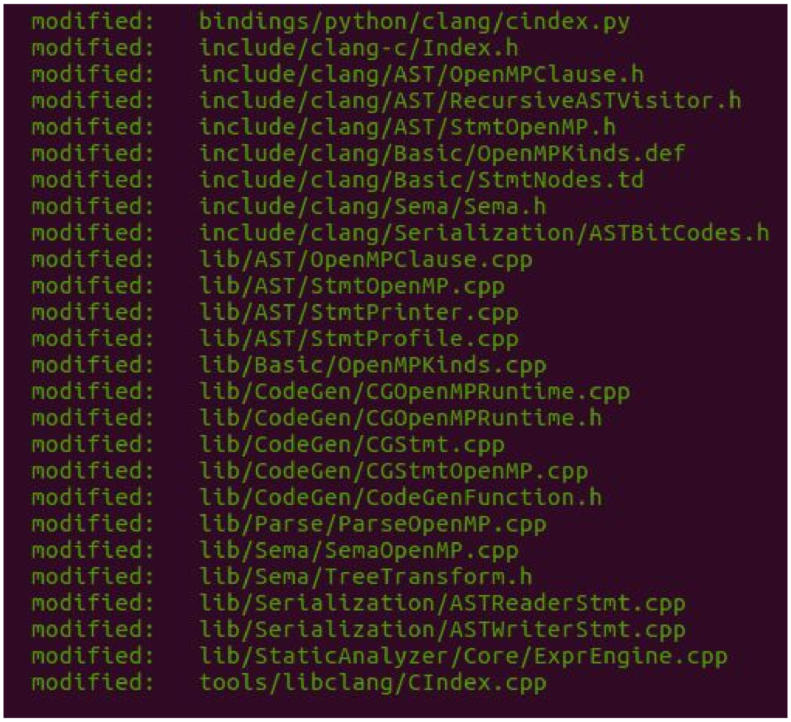
Assignment 1: Implementing the OpenMP “inc” Directive

The **FLUSH** directive identifies synchronization points at which the implementation must provide a consistent view of memory. It must appear at the precise point in the code at which the synchronization is required. To avoid flushing all variables, we specify a list. Thread-visible variables are written back to memory at the point at which this directive appears. Modifications to thread-visible variables are visible to all threads after this point. Subsequent reads of thread-visible variables fetch the latest copy of the data. The flush operation provides a guarantee of consistency between a thread’s temporary view and memory.

Running `grep -iR flush` in the clang directory gives a number files. Out of which, the following are of importance:



```
modified:   bindings/python/clang/cindex.py
modified:   include/clang-c/Index.h
modified:   include/clang/AST/OpenMPClause.h
modified:   include/clang/AST/RecursiveASTVisitor.h
modified:   include/clang/AST/StmtOpenMP.h
modified:   include/clang/Basic/OpenMPKinds.def
modified:   include/clang/Basic/StmtNodes.td
modified:   include/clang/Sema/Sema.h
modified:   include/clang/Serialization/ASTBitCodes.h
modified:   lib/AST/OpenMPClause.cpp
modified:   lib/AST/StmtOpenMP.cpp
modified:   lib/AST/StmtPrinter.cpp
modified:   lib/AST/StmtProfile.cpp
modified:   lib/Basic/OpenMPKinds.cpp
modified:   lib/CodeGen/CGOpenMPRuntime.cpp
modified:   lib/CodeGen/CGOpenMPRuntime.h
modified:   lib/CodeGen/CGStmt.cpp
modified:   lib/CodeGen/CGStmtOpenMP.cpp
modified:   lib/CodeGen/CodeGenFunction.h
modified:   lib/Parse/ParseOpenMP.cpp
modified:   lib/Sema/SemaOpenMP.cpp
modified:   lib/Sema/TreeTransform.h
modified:   lib/Serialization/ASTReaderStmt.cpp
modified:   lib/Serialization/ASTWriterStmt.cpp
modified:   lib/StaticAnalyzer/Core/ExprEngine.cpp
modified:   tools/libclang/CIndex.cpp
```

The implementation of INC is similar to FLUSH, barring a few files. Therefore first we clone the flush code replacing flush with inc and then include the logic for increment. Here is the **COMMON IMPLEMENTATION of flush and inc -**

- `Basic/OpenMPKinds.def` defines the list of supported OpenMP directives and clauses.
  The flush directive is defined as OPENMP_DIRECTIVE(flush) and the flush clause is defined as OPENMP_CLAUSE(flush, OMPFlushClause) in Basic/OpenMPKinds.def. I made similar definitions for INC.
- The Parse folder does the Syntax Analysis or Parsing.

- `Parse/ParseOpenMP.cpp`: If Dkind is OMPD_flush, and lookahead token is left parenthesis, then consume token and push copy of the current token back to stream to properly parse pseudo-clause OMPFlushClause. I wrote a similar case OMPD_inc below it.
- The Sema folder does the Semantic Analysis - Checks for errors that are not found in syntax analysis – Incompatibilities and mismatches, especially type checking.
- `ActOnOpenMPVarListClause()` is located in `Sema/SemaOpenMP.cpp` and calls `ActOnOpenMPFlushClause()` which returns `OMPFlushClause::Create(..)`.
- `StmtNodes.td`: The line `def OMPFlushDirective : DStmt<OMPExecutableDirective>` adds a Statement Node. Clang reads this file and generates a StmtNodes.inc file, which is used to define different statement classes and read by several classes to define their node visitor function. I define an OMPIncDirective node, which extends the OMPExecutableDirective class - a basic class for representing single OpenMP executable directive.
- `Sema/SemaOpenMP.h`: The function ActOnOpenMPFlushDirective is called on well-formed '#pragma omp flush' after parsing of the associated statement. I call the OMPFlushClause::Create(Context, StartLoc, LParenLoc, EndLoc, VarList); from ActOnOpenMPFlushDirective function. We add our case OMPD_flush where we call the ActOnOpenMPAllocateDirective function. A similar OMPD_inc has to be added.
- AST has code related to abstract syntax tree generation. `AST/StmtOpenMP.cpp` has the `OMPFlushDirective *OMPFlushDirective::Create()` which creates the directive. It takes the AST context, start and end location of directive and list of clauses as parameters. The class also has a method to create an empty directive with the place for specifies number of clauses (NumClauses). Similarly include OMPIncDirective *OMPIncDirective::Create() in this file.
- `RecursiveASTVisitor.h` - This file defines the RecursiveASTVisitor interface, which recursively traverses the entire AST.
- `StmtPrinter.cpp` - This file implements the Stmt::dumpPretty/Stmt::printPretty methods, which pretty print the AST back out to C code. We add our definition of VisitOMPIncDirective() here looking at VisitOMPFlushDirective(). `VisitOMPFlushDirective()` function in `AST/StmtPrinter.cpp` visits the given flush node in the tree and prints it.
- `StmtProfile.cpp` - This file implements the Stmt::Profile method, which builds a unique bit representation that identifies a statement/expression.
- `ASTReaderStmt.cpp` - Implements Statements and Expression deserialization. This implements the ASTReader::ReadStmt method.
- `ASTWriterStmt.cpp` - Implements serialization for Statements and Expressions. In ASTWriterStmt.cpp we define the function VisitOMPIncDirective preferably after the definition of VisitOMPExecutableDirective.
- `TreeTransform.h`: We also need to define a tree transformation in `TreeTransform.h` - This file implements a semantic tree transformation that takes a given AST and rebuilds it, possibly transforming some nodes in the process. Using StmtNode.td, this class will already declare the TransformOMPIncDirective function.

- CodeGen/OpenMPRuntime.cpp: The `emitFlush()` function calls EmitRuntimeCall() of class CodeGenFunction. It builds runtime call `void __kmpc_flush(ident_t *loc)`. __kmpc functions are defined in the openmp repository.
- Serialization/ASTBitCodes.h: We need to create a record for our statement in the enum StmtCode. These constants describe the records that describe statements or expressions. These records occur within type and declarations block, so they begin with record values of 128. Each constant describes a record for a specific statement or expression class in the AST. To add our own record we modify the StmtCode enum in ASTBitCodes.h file to add STMT_OMP_INC_DIRECTIVE.

Additional files to be changed for INC directive -

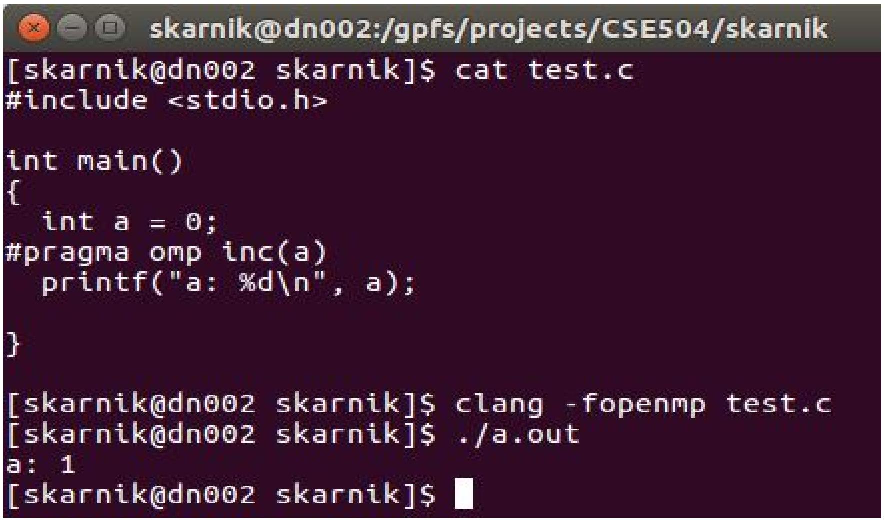
- CodeGen/CGExprScalar.cpp: This file has a function `EmitScalarPrePostIncDec()` which contains implementation of unary increment/decrement operations on variables. Taking this as reference, I change the below file.
- CodeGen/CGOpenMPRuntime.cpp: I have placed the actual logic for the inc directive in the the `emitInc()` function. The first element of the input array(ArrayRef<const Expr *> arr) is the lvalue. We add 1 to this lvalue and store the result back to source location(SourceLocation Loc).
- The integer type checking can be done by (type->isIntegerType()), where type is an instance of QualType class.

test.ll: (Comparing IRs)

IR for unary post increment operation on a variable (a++)	**IR generated for incrementing with inc pragma**	**Comments**
. . .	. . .	
define i32 @main() #0 {	define i32 @main() #0 {	
%1 = alloca i32, align 4	%1 = alloca i32, align 4	//allocate mem for variable
store i32 0, i32* %1, align 4	store i32 0, i32* %1, align 4	//set var = initial_value 0
%2 = load i32, i32* %1, align 4	%2 = load i32, i32* %1, align 4	//load var to %2
%3 = add nsw i32 %2, 1	%3 = add i32 %2, 1	//add 1 to %2 and store in %3
store i32 %3, i32* %1, align 4	store i32 %3, i32* %1, align 4	//store %3 back to src location
ret i32 0		
}		
	%4 = load i32, i32* %1, align 4	//remaining code for printf etc. after pragma
	%5 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([7 x i8], [7 x i8]* @.str, i32 0, i32 0), i32 %4)	
	ret i32 0 }	

The table shows that both IRs match and thus the pragma works!

OUTPUT



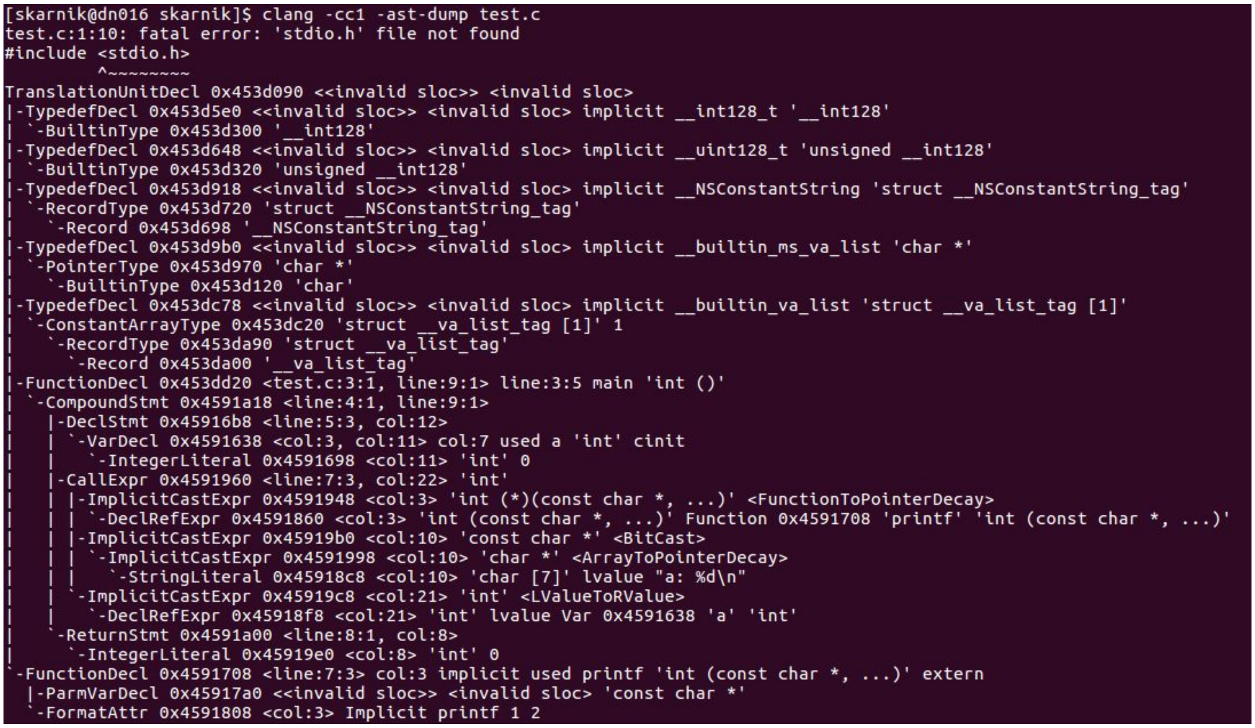
```
[skarnik@dn002 skarnik]$ cat test.c
#include <stdio.h>

int main()
{
  int a = 0;
#pragma omp inc(a)
  printf("a: %d\n", a);

}

[skarnik@dn002 skarnik]$ clang -fopenmp test.c
[skarnik@dn002 skarnik]$ ./a.out
a: 1
[skarnik@dn002 skarnik]$ 
```

AST FOR INC DIRECTIVE



```
[skarnik@dn016 skarnik]$ clang -cc1 -ast-dump test.c
test.c:1:10: fatal error: 'stdio.h' file not found
#include <stdio.h>
         ^~~~~~~~~
TranslationUnitDecl 0x453d090 <<invalid sloc>> <invalid sloc>
|-TypedefDecl 0x453d5e0 <<invalid sloc>> <invalid sloc> implicit __int128_t '__int128'
| `-BuiltinType 0x453d300 '__int128'
|-TypedefDecl 0x453d648 <<invalid sloc>> <invalid sloc> implicit __uint128_t 'unsigned __int128'
| `-BuiltinType 0x453d320 'unsigned __int128'
|-TypedefDecl 0x453d918 <<invalid sloc>> <invalid sloc> implicit __NSConstantString 'struct __NSConstantString_tag'
| `-RecordType 0x453d720 'struct __NSConstantString_tag'
|   `-Record 0x453d698 '__NSConstantString_tag'
|-TypedefDecl 0x453d9b0 <<invalid sloc>> <invalid sloc> implicit __builtin_ms_va_list 'char *'
| `-PointerType 0x453d970 'char *'
|   `-BuiltinType 0x453d120 'char'
|-TypedefDecl 0x453dc78 <<invalid sloc>> <invalid sloc> implicit __builtin_va_list 'struct __va_list_tag [1]'
| `-ConstantArrayType 0x453dc20 'struct __va_list_tag [1]' 1
|   `-RecordType 0x453da90 'struct __va_list_tag'
|     `-Record 0x453da00 '__va_list_tag'
|-FunctionDecl 0x453dd20 <test.c:3:1, line:9:1> line:3:5 main 'int ()'
| `-CompoundStmt 0x4591a18 <line:4:1, line:9:1>
|   |-DeclStmt 0x45916b8 <line:5:3, col:12>
|   | `-VarDecl 0x4591638 <col:3, col:11> col:7 used a 'int' cinit
|   |   `-IntegerLiteral 0x4591698 <col:11> 'int' 0
|   |-CallExpr 0x4591960 <line:7:3, col:22> 'int'
|   | |-ImplicitCastExpr 0x4591948 <col:3> 'int (*)(const char *, ...)' <FunctionToPointerDecay>
|   | | `-DeclRefExpr 0x4591860 <col:3> 'int (const char *, ...)' Function 0x4591708 'printf' 'int (const char *, ...)'
|   | |-ImplicitCastExpr 0x45919b0 <col:10> 'const char *' <BitCast>
|   | | `-ImplicitCastExpr 0x4591998 <col:10> 'char *' <ArrayToPointerDecay>
|   | |   `-StringLiteral 0x45918c8 <col:10> 'char [7]' lvalue "a: %d\n"
|   | `-ImplicitCastExpr 0x45919c8 <col:21> 'int' <LValueToRValue>
|   |   `-DeclRefExpr 0x45918f8 <col:21> 'int' lvalue Var 0x4591638 'a' 'int'
|   `-ReturnStmt 0x4591a00 <line:8:1, col:8>
|     `-IntegerLiteral 0x45919e0 <col:8> 'int' 0
`-FunctionDecl 0x4591708 <line:7:3> col:3 implicit used printf 'int (const char *, ...)' extern
  |-ParmVarDecl 0x45917a0 <<invalid sloc>> <invalid sloc> 'const char *'
  `-FormatAttr 0x4591808 <col:3> Implicit printf 1 2
```

References:

https://www.ibm.com/support/knowledgecenter/SSGH2K_12.1.0/com.ibm.xlc121.aix.doc/compiler_ref/prag_omp_flush.html
www.openmp.org/wp-content/uploads/openmp-4.5.pdf
http://freecompilercamp.org/