CSE 504: Assignment 1
Implementing the OpenMP Allocate Directive
Due date: Monday, October 9

In this assignment, you will study the front end of the LLVM/Clang compiler and extend it to implement a simplified version of the *allocate* directive in the upcoming OpenMP 5.0. This assignment is intended to give you some insight into the real-world front end used by LLVM/Clang.

To complete this assignment, you will perform the work described below. The report and the files are to be made available to the TA for evaluation by midnight on the due date. Do **not** delete interim versions of your files or anything else that documents how you have performed your assignment tasks.

As always, do **not wait** until the due date to do your work: otherwise the machine may become very full around the due date.

Here is what you should do:

For this work you will use your account on Seawulf. You should have installed LLVM/Clang and tested it with CG already in the homework.

Once you have logged in to Seawulf, go to the directory '/gpfs/projects/CSE504/hw2'. Here you will find the test programs used in this assignment. Note you will need to always pass the flag -fopenmp to clang in this assignment since it is needed to enable OpenMP translation. Also note that we will work on the front end in this assignment, so the code your will work on is located in *<LLVM source path>/tools/clang*. The things you need to accomplish are listed as follows.

1) Understand the semantic of the *allocate* directive. The syntax of the allocate directive is as follows:

#pragma omp allocate(A)

Here, A is a pointer defined in the program, and this statement specifies that memory space for the data pointed to by A should be allocated, using *malloc* internally. To make this task simpler, we assume for this task that the amount of memory to be allocated is always *(100*sizeof(int))*. You are encouraged to implement a version that supports various size allocation and we will give extra credit for that.

An example of the allocate directive is in /gpfs/projects/CSE504/hw2/test.c. It should achieve the same effect as the code in /gpfs/projects/CSE504/hw2/test_malloc.c. You can check the generated IR of these two programs using the IR print flags introduced on September 6th.

2) Find out about, and understand, how the OpenMP flush directive is implemented. The implementation of flush is similar to that of allocate. We will learn about its implementation as an example and then design your allocate implementation in a similar way later.

The syntax and semantics of the flush directive can be found here https://software.intel.com/en-us/node/524520 . It is described in the OpenMP specification (www.openmp.org/wp-content/uploads/openmp-4.5.pdf)

First, use the grep command to find all places related to flush. Assume you are in <LLVM source path>/tools/clang, use the commandline "*grep -R flush include/ lib/ tools*". This gives you a list of files that may relate to the implementation of flush. Then further go through those files and grep the keyword related to flush you find in them, such as OMPD_flush. Note that beside the flush **directive**, there is also a flush **clause**, which you should ignore. You should be able to tell which is for which; for instance, OMPC_flush is for the flush clause while OMPD_flush is for the flush directive.

An **incomplete** list of files that you should look at is as follows (if you don't find these, probably you should search again):

1. Related to definitions: include/clang/Basic/openmpkinds.def, lib/Basic/OpenMPKinds.cpp, include/clang/AST/openmp*

2. Related to parsing: lib/Parse/parseopenmp.cpp

3. Related to syntax analysis: lib/AST/StmtPrinter.cpp, ./lib/AST/StmtProfile.cpp, ./lib/Serialization/ASTWriterStmt.cpp, ./lib/Serialization/ASTReaderStmt.cpp, ./include/clang/AST/RecursiveASTVisitor.h

4. Related to semantic analysis: lib/Sema/SemaOpenMP.cpp

5. Related to IR generation: lib/CodeGen/CGStmtOpenMP.cpp, lib/CodeGen/CGOpenMPRuntime.*

An example of the flush directive is given in /gpfs/projects/CSE504/hw2/flush.c. You can check the IR generated for this program using the IR print flags introduced in lesson 3 on September 6th.

3) Implement the allocate directive. After you have gained an understanding of the implementation of the flush directive, you should be able to implement the allocate directive. While the flush directive will generate a call to __kmpc_flush() as you have already seen in the IR code of flush.c, your code should generate a call to malloc() for each allocate directive.

How to submit:

You will need to submit your files. Please also leave appropriately named files in an appropriately named directory in your course account in case we need to inspect them. Your report and documentation in the code should provide any explanations needed. Please only submit the diff file with your modifications. Assuming you are in <LLVM source path>/tools/clang, use "**git diff 559aa046fe3260d8640791f2249d7b0d458b5700 > hw2_<your net id>.diff**" to get your modifications since the stable 4.0 version. Please send the diff file to the TA by midnight on the due date.

Please also submit a report for your work. Your report should describe the implementation of the flush directive and the implementation of the allocate directive. It should be *no more than 4* letter-size pages in length. Email your report in pdf or word format to the TA by midnight on the due date. You are required to submit a **hard copy** of the report class immediately following (but not on) the due date.

Grading:

Grades from A through F will be assigned. Grades will be based upon the correctness of your work, completeness of your responses, and the overall quality of your work. Please do your own work: do NOT copy code or text from other class participants or from any other source.