Proposal for Google Summer of Code 2021
Pydata/Sparse

LLVM Back-end for the Tensor Algebra Compiler

Personal Information:

Name: Roberto Gomes Rosmaninho Neto
GitHub: https://github.com/robertorosmaninho

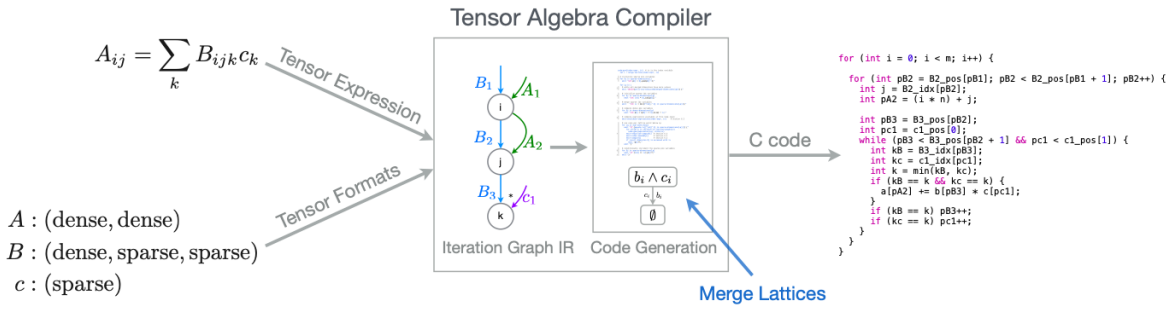
About me:

As mentioned, my name is Roberto, I'm currently studying Computer Science at Federal University of Minas Gerais in Brazil, and I'm also working as an undergraduate research assistant in the Compilers Laboratory (http://lac.dcc.ufmg.br/). A few of my most recent projects include: (1) Developing the MLIR backend for D Programming Language, which was awarded as the best project in Symmetry's 2019 Autumn of Code; (2) Developing a static analysis in LLVM to automatically infer function arguments that could profit from being marked with the "lazy" modifier, in the D Programming Language; (3) Working with C++, developing LLVM optimization passes in class projects; (4) Using Python to perform data analysis and train machine learning models in an attempt to improve static branch prediction methods, comparing them to state-of-the-art approaches; and (5) Working as an intern DevOps Engineer at Synergia, a Brazilian software company. Moreover, before enrolling in the university, I graduated with an Associate's Degree in Computer Technology, with a focus on Web and Mobile Development. During this time I studied and worked with HTML, CSS and other web related languages. I also used Java as my main language for almost 2 years when developing other projects at the time.

The Project:

An overview: TACO is a C++ library to automatically generate a kernel for any compound linear and tensor algebra operation on dense and sparse tensors. Its performance is competitive with best-in-class hand-optimized kernels in popular libraries, while supporting far more tensor operations. Today taco provides a command-line tool to generate C kernels to let users include it in their applications to serve as the starter point for further development. The code generation is shown below:

The Tensor Algebra Compiler (taco)



The main goal of this project is to provide a new back-end for TACO, that generates LLVM bytecode instead of C code, using the LLVM C++ API to do it. The main benefits of outputting LLVM IR are enabling optimizations at the LLVM IR level and to target any machine code available in the LLVM framework, including NVPTX. Generating code for CUDA is one of the extra goals that I'll pursue in this project along with the implementation of JIT compilation for the bytecodes generated.

As part of the work is already implemented, a first revision of the files `codegen.h`, `codegen.cpp` and `module.h` from `taco/src` will be necessary. However, most of the work will be implemented into `codegen_llvm.h` and `codegen_llvm.cpp`. The code already done in `codegen_c.h`, `codegen_c.cpp` and in an old PR with the same goal of implementing a backend to TACO using LLVM 7.0 will be taken as inspiration.

The work will be focused on the implementation the visitors that are not implemented yet, doing something like:

```
void CodeGen_LLVM::visit(const Add *op) {
        llvm::errs() << "LLVM CodeGen Visiting Add\n";
}
```

```
void CodeGen_LLVM::visit(const Add *op) {
        llvm::errs() << "LLVM CodeGen Visiting Add\n";
        auto *a = codegen(op->a);
        auto *b = codegen(op->b);
        value = this->Builder->CreateAdd(a, b);
}
```

The example above shows the simplest implementation of an operation, much more complex operations are expected to be implemented in this work. The following timeline shows the order of implementation that I will follow to finalize the generation of the code for the IR LLVM:

The project Timeline:

- **Week 1 (06/07 - 06/13):** Review and implementation of Module, Function, Block and Scope.
- **Week 2 (06/14 - 06/20):** Implementation of loop operations: For and While.
- **Week 3 (06/21 - 06/27):** Implementation of conditional branch operations: If, Else, IfThenElse.
- **Week 4 (06/28 - 07/04):** More implementation of conditional branches: Switch, Case and implementation of Call operation.
- **Week 5 (07/05 - 07/11):** Implementation of logic operations: NOT, AND, OR, BitOr and BitAnd.
- **Week 6 (07/12 - 07/18):** Implementation of comparison operators: EQ, NEQ, GT, LT, GTE, LTE.
- **Week 7 (07/19 - 06/25):** Implementation of memory operations: Load, Store, Malloc and SizeOf.
- **Week 8 (07/26 - 08/01):** Implementation of binary arithmetic operations: ADD, SUB, MUL, DIV, REM.
- **Week 9 (06/02 - 08/08):** More implementation of arithmetic operations: MAX, MIN, NEG, SRQT.
- **Week 10 (06/09 - 08/15):** Tests and bugfixes

Time Commitment:

My Summer vacations will finish before the beginning of the GSOC code phase on 15th May. Therefore, I plan to work with the LLVM IR Code Generation between 20 and 30 hours per week and reserve some time for college assignments. Besides college and GSOC I won't have any other commitments until the end of the program to focus.

Bibliography:

- Taco research paper and the talk at Microsoft Research by Fredrik Kjolstad.
- The research paper that moved to the current method of code generation.
- Partial work on the back-end.
- An old pull request with an old LLVM version.