GNU Octave - Implement JIT compiling

Max Brister

April 13, 2012

1 Your name

Max Brister (IRC: fisheater)

School: New Mexico Institute of Mining and Technology (New Mexico Tech)

Pursuing a Computer Science bachelors degree. By this summer I will have completed my junior year. By credit hours I am classified as a senior.

2 Your email address

max@2bass.com

3 The name of the project

GNU Octave - Implement JIT compiling

4 Summary

(Copied from the ideas page)
Octave's interpreter is very slow on loops. Implementing JIT compiling would dramatically speed up execution of these loops. This is a very big project, but a dedicated student might make a good attempt of doing this over a summer. There may be some work already in place by the time the summer comes along. The idea is to probably use LLVM to aid with the JIT compilation.

5 Benefits

5.1 Users

For fast execution time GNU Octave currently relies on users vectorizing their code. However, it is not always trivial, or even possible, to vectorize code. JIT

compilation will benefit the user community greatly by increasing GNU Octave's execution speed for code that has not been vectorized. JIT compilation should also provide a marginal benefit for vectorized code, as such code still often relies on loops.

The problem of vectorizing code is especially difficult for engineering students. At New Mexico Tech all engineering students are required to take an introductory programming course in matlab. As the course focuses on basic programming, they are not introduced to code vectorization. If these engineering students try GNU Octave, they are given the impression that GNU Octave is much slower than matlab.

A JIT compiler for GNU Octave will remove this problem. It will also make GNU Octave a more viable alternative for introductory engineering courses, as student code will be executed at a reasonable speed.

5.2 The GNU Project

It is clear that JIT compilation for GNU Octave will benefit GNU Octave's users. By benefiting GNU Octave's users, the entire GNU project benefits. This is because more people will be interested in using free software and using the GNU operating system for numerical computing. That is, improving GNU Octave increases the relevance of the GNU operating system by improving its numerical computing capabilities.

6 Deliverables

6.1 What software will be added or changed?

The GNU Octave interpreter will be modified to support JIT compilation using LLVM.

6.2 What parts of the projects code will be affected?

There will be some modification of tree_evalulator. The main modification will be adding a metric for determining when to JIT compile, and then calling the appropriate method to JIT compile.

A new class extending tree_walker will probably be added to convert Octave's IR to LLVM's IR. This new class will be modeled after tree_evaulator, and instead of executing code it will perform JIT compilation.

A new class will also need to be added in order to manage LLVM's JIT engine. This class will deal with creating the engine, compiling code, and executing code.

6.3 Which documentation will be updated?

1. User documentation

   If done correctly there should be little need to update GNU Octaves user documentation for JIT compiling. The JIT process should be transparent to users. It just makes the code execute faster.

   That being said, in the interpreter documentation Section 19: Vectorization and Faster Code Execution states that vectorization of GNU Octaves code leads to a performance increase of 10x-100x. This statement may need to be modified at the end of the project. Additional, there should be some mention as to what code can be JIT compiled.

   It also may be interesting to produce a report on a speed comparison of GNU Octaves interpreter, GNU Octave using JIT, and matlab.

2. Developer documentation

   Much of the code in pt-* is currently poorly documented. I will try to do a better job at documenting any code I add to the project.

   Additionally, I will produces a report at the end of the project. This report will be published on the mailing list and will detail the progress I made.

7 Plan

7.1 Midterm Goal

Simple expressions involving octave_scalar will JIT compile. For example,

$$x = y * 5 + z^{20}$$

should JIT compile if x, y, and z are octave_scalars.

At this point the project is more of a proof of concept. I expect that the overhead of checking types for JIT compilation and JIT compilation its self will be greater than the benefit of JIT compilation. Therefore, at this stage the code will have limited use for the GNU Octave project. However, the code should be a good basis for the implementation of the rest of JIT compilation.

7.2 Final Goal

Code generation will work for octave_scalar and loops. There should be a noticeable speed up for compiled code. Furthermore, there should be no noticeable slow down in code that can not be compiled. Code speed up will be measured using some simple numerical integration tests that deal only with scalars. Code

slow down will be measured by comparing the runtime of *make check*.

At this point in the project the code should have some practical use in GNU Octave. JIT compilation will be much more useful once matrices are supported. However, it should still be interesting to merge the code into the main GNU Octave development branch. This is because JIT compilation will make some code much faster without a major impact to the rest of the code.

7.3 Timeline

April 23 - May 20 (Community Bonding Period)
NOTE: I have finals from May 5 - May 10.

- Look into bugs - I will try to focus on bugs relating to the interpreter.
- Hang out on IRC
- Participate on the mailing list

May 21 - June 3 (Coding starts)

- Create branch for JIT development - Merges from the main GNU Octave branch will occur regularly throughout the project.
- Work on initial framework
- Add LLVM to build scripts

June 4 - June 17

- Implement codegen for octave_scalar - Simple expressions only.

June 18 - July 8

- Cleanup code - Make things pretty before midterm evaluations.
- Test cases - Ensure codegen works well for octave_scalar.

July 9 - July 13 (Midterm evaluations)

- Fix any issue my mentor finds
- Optimization - Don't even try to compile a subtree if there is a tree node that can not be compiled.

July 14 - July 27

- OctConf 2012 - Go to OctConf 2012. Possibly give a short presentation on project progress.
- Codegen loops - First for, then while.

July 28 - August 12

- Codgen loops - Finish codegen for loops.
- Basic performance profiling - Ensure the JIT branch is actually faster than the main branch.

August 13 - August 20

- Code cleanup
- Status report to community

7.4 Behind Schedule

As the schedule is fairly leisurely, I doubt I will get behind schedule. If I do get behind schedule it will probably be because I end up needing to write a large section of existing code. I do not expect this to be likely, as I have already looked at most of the code involved, and there is nothing major preventing JIT compilation.

However, if this problem does occur, I will have a discussion with my mentor as to if the rewrite is absolutely necessary. If it is necessary, then at that point we will consider modifications to my project schedule in order to get back on track.

7.5 Ahead of Schedule

If I get ahead of schedule I will try to finish the main portion of the project early. After that is complete I will start working on some of the following subprojects.

- matrices - Allow for code with matrices to be compiled. This can start with basic support for code like $A(i) = v$ where i and v are scalars.
- struct and class - Similar to matrices, JIT code that uses struct and class.
- scripts - It should be possible to JIT code with script calls. This will involve treating scripts like #include statements. Then if the script changes, the code will be recompiled. This is complicated by the fact that the script may fail to parse. In order to keep the same behavior as non-JIT code these errors must only be reported if the script is actually executed.
- user functions - It should be possible to JIT most user function given the types of varargin and nargout. The code for these automatic function specializations should be stored and used whenever the function is called. This may be complicated by the need to check to see if a function is old (just like with scripts).

- mex functions - In order to do this properly we need to know what types a mex function receives and returns. This means changing how mex functions are defined. Such a rewrite should also address bug #35694.
- Review copy on write mechanics - There may be a performance benefit from doing some static analysis to determine where/when variables are mutated. It might be interesting to experiment with this.
- And more

7.6 Existing Code

There may already be some code for JIT compilation in place before the summer begins. If this is the case I will continue implementing from the existing code. If there is already a significant amount of work done, we may consider adjusting my schedule. Most likely this would mean including basic matrix support as part of final goal.

8 Communication

I already frequent the #octave IRC channel. This will allow my mentor to quickly communicate with me. Additionally, I will respond to email. I plan on updating my mentor on the progress I have made after each step of my plan. All development will be done on a public hg repository, so my mentor or anyone else can check my progress.

9 Qualification

9.1 Why did this project appeal to you?

I am an avid user of free software, and have used both GCC and emacs since I started programming. When thinking about which project I would like to contribute to for GSoC, I focused on GNU projects for this reason. GNU Octave is particularly intriguing to me, because two of my favorite courses are Numerical Methods and Compilers. Getting involved in GNU Octave development will allow me to pursue both interests.

Additionally, a course I took at New Mexico Tech, Computational Mechanics, used matlab. The proprietary nature of matlab caused several issues during the course. For example, when doing image analysis I came up with a solution, but when helping the person next to me they got an error “There are no licenses left for the requested toolbox.” By contributing GNU Octave I hope to help GNU Octave become the de facto standard for numerical computing, preventing such issues.

9.2 How will you benefit from it?

Implementing JIT compilation in GNU Octave will give me experience working on a major free software project. This will provide me with invaluable experience in working on large projects. It should also give me experience in working with JIT compilation, something that I have been interested after taking a course on compilers.

9.3 What will you do once this project is "finished"?

I will continue working on the project in some of the following areas

- Bug Fixes - I will undoubtedly introduce new bugs to GNU Octave, as well as uncover existing ones over the summer. I plan on helping to fix these bugs.
- Continue development based on final report - There will still be lots of of work left on the JIT compiler. I plan on trying to continue this work during the school year.
- Interested in parallelization of the interpreter/JIT compiler - The GUI Octave interpreter requires a lot of work before parallelization can be done successfully. I have experience both writing parallel code and parallelizing sequential code.
- Possibly contribute to GUI development - I have a lot of experience developing GUIs. While I find GUI development less interesting than interpreter development, I recognize the need for a good GUI for GNU Octave adoption. I am specifically interested in developing GUI functionality that is not easily done in the commandline. For example, a GUI creator GUI.

Personally, I have one more year of my bachelors degree left. After finishing my bachelors degree I will probably be seeking employment in industry, with a strong preference for working on free software.

9.4 Have you worked on any free software before?

Yes

- pycandis

  In my first year at New Mexico Tech I was hired by Dr. Raymond to provide a python interface to a file format used for atmospheric research at New Mexico Tech, called Candis. The Candis distribution can be found at here, and my work is specifically in the pycandis folder. Documentation on how to use pycandis can be found here.

- GNU Octave

  In an effort to learn more about the GNU Octave source code and development process I have contributed several patches. Most notably my

patch which implements nested function support has been accepted. For a completed list of my accepted patches run *hg log -u "Max Brister"* on a up to date copy of the octave development sources.

9.5 Of the skills that you will need to complete the project, which do you already have?

I already have a good basis in the theory of compiler writing. I have taken a compiler writing course at New Mexico Tech. For the compilers course we had groups of four people. We then had to write an optimizing compiler for a subset of C89 in C++. Code generation was for x86_64. The class provided us with an AST, but no other code. Working on a compiler gave me a good introduction to the types of IRs I see in GNU Octave and LLVM.

Additionally, I am also already well versed in C++. I have used C++ extensively in the compiler for compilers, a computer game for a class in software engineering, and on several hobby projects. This knowledge has allowed me to quickly understand the GNU Octave codebase.

9.6 What will you need to learn?

I have no experience using LLVM. I will have to learn the LLVM IR and the API used to construct the LLVM IR. I have already read the LLVM tutorial. The LLVM IR is similar to IRs I have already worked with. Likewise, the LLVM JIT API appears to be fairly straightforward.

While I have a good idea of how the GNU Octave interpreter works, I will be much more familiar with its internal workings by the end of the project.