CS 301 Spring 2016
Project Phase 5
April 11 – May 16

IC Compiler/Runtime Extension

Plan

To wrap up the semester, you will extend your IC compiler with an interesting new feature of your choosing. I provide some sample suggestions below, but there are many more possibilities. Find something that excites you and talk to me to form a plan. Almost anything compiler, runtime, or programming languages is acceptable. Feel free to branch off on your own, form new teams (of up to 3), or keep your existing teams.

When choosing topics and teams, aim for a non-trivial project, but keep time limits in mind:

- You likely will be busy with the IC optimizer assignment until less than a week before the end of classes. We will hold project presentations either on the last day of classes (at which point they would be more presentations of plans) or later during reading period/finals.
- You probably know much of your IC compiler implementation fairly well now. Regrouping can have some extra code-learning costs, but some projects may not depend much on your work so far.
- Some fascinating project topics do not appear until the last couple weeks of the course, so look ahead for interesting topics, not just behind.

Project Shape

The goal of the project is explore an additional compiler or runtime system feature we have not covered in the course. Your final products will include some combination of implementation artifact (code) and a short paper. There are two tracks: implementation and a research-focused literature survey. Implementation projects are encouraged, but literature surveys on topics of interest can also work if you are coded out. Both options will involve some reading and the production of a paper.

Implementation Focus In this type of project, you will implement and experimentally evaluate a new-to-us compiler/runtime feature in your IC compiler or another platform. Depending on the feature, the evaluation will likely focus on performance or accuracy, or possibly some other measure of efficacy fitted to the feature in question. You will submit your code, raw experimental data, a *short* technical paper on your work: perhaps 2 pages or longer. The paper should follow an outline roughly matching the following:

1. Introduction: briefly describe the work and its context and give an overview of what the paper will communciate.
2. Background (optional separate): describe the foundations of this feature, citing relevant references. (You may combine this inline with the next section.)
3. Design/Theory: describe abstractly how the feature works.
4. Implementation: describe how you implemented the feature.
5. Evaluation: describe experiments performed and results observed. Discuss.
6. Conclusions.

I will help tweak the specifics for your topic. Many projects will implement a well-known feature. In this case, the focus will be on the implementation and evaluation sections. All others will be brief. Some projects may implement less-studied features or recent ideas in research. In this case, there will be more balance among the sections.

Literature Focus As an alternative to an implementation-focused project, you may conduct a literature survey on a relevant topic of interest. The survey should include some research literature. The most interesting topics would be areas of recent research advances. You will submit a paper as your final product. Compared to the implementation option, this paper will be longer: perhaps 5 pages. We can discuss the

Format I encourage you to use LaTeXto prepare the paper. We can even use the ACM SIGPLAN (Special Interest Group for Programming Languages) template to make it look like a real compilers paper!

Project Ideas

I do not assume that you know what all of these are. We will cover some briefly later in the course. The ambition level varies widely. Some ideas have a high bar for "making it work at all." Others have several potential (and reasonable) stopping points along the way. I am happy to talk about any and all of these (and your own ideas too).

Language Extensions: add a language feature to IC and implement it in the compiler and runtime.

- Exceptions and exception handling.
- A generic type system.
- Limited local type inference.
- First-class functions and closures.
- Nullable and non-nullable types and checking.
- Threads, tasks, or some other construct for concurrent or parallel programs.
- Dynamic or static information flow control.
- Java-style interfaces. Evaluate the performance penalties for using interface methods. Improve their performance by extending the IC runtime to include polymorphic in-line caching.

More Optimizations: improve your compiler or runtime system to produce more efficient IC programs or execute them more efficiently. (Browse EC 9 – 11 or 13 for more ideas.)

- Add partial redundancy elimination or loop optimizations to your optimizer and evaluate the performance effects.
- Implement low-level instruction selection with tiling and intelligent register allocation for basic blocks and evaluate the performance effects.
- Implement a garbage collector. Start with a copying or mark-sweep algorithm. Implement a generational collector if you are really into it.
- Implement Class Hierarchy Analysis and evaluate the performance effects.
- Implement method inlining or specialization and evaluate the performance effects.
- Implement conservative type-based alias analysis and use it for elimination of redundant field or array accesses. Evaluate the performance effects.
- Translate TAC to SSA form, compute dominators, implement one or two SSA-based optimizations, and evaluate the performance effects.
- Generate SIMD/vector instructions for suitable array codes.

Alternative Back Ends: target a different language/architecture.

- Implement a code generator targeting JavaScript, WebAssembly, `asm.js`, or similar, to run IC programs in the browser.
- Implement a code generator targeting LLVM bitcode and compare the performance of your IC optimizer and backend to the LLVM optimizer and backend.
- Implement a code generator targeting ARM to run IC programs on a Raspberry Pi.

Tools and Analysis: extend your compiler or runtime system to report additional useful information to programmers.

- Implement a performance profiler to measure how much each basic block contributes to the running time of a program.
- Track "blame" for runtime errors like null pointer dereferences. Where did that `null` come from? What code did it flow through (without being checked) to get here?
- Detect constant or "stationary" fields. Or recast this as a language feature for explicit definitions of constants, plus optimizations that can constant-fold uses of constant fields.

Schedule and Products

Week of April 11-15: Come to tutorial meetings or Friday class with some ideas. I will discuss with each team to settle on a reasonable focus and scope. After this meeting, I will help you gather resources on your topic

Friday, April 22: Submit a one-paragraph description of your project, confirming (briefly) what you plan to build and how you plan to evaluate it.

April 19-29 I will take time in tutorial meetings or class to discuss/plot project design and implementation strategy with teams.

Tuesday, May 3: Submit a (minimum) 1-page paper outline / design document. This will become the final project paper. For now, it should include a brief introduction, a full section on motivation and background, and an outline of plans for design, implementation, and evaluation.

May 10-13, (exact date TBD): Gathering and presentations. We will find a time, hopefully early in exam week, to gather, eat food, and present what everyone has built. Aim to have a working artifact and at least cursory evaluation by this time. Each team will give a 10-minute presentation.

May 16: End of exam period, final project document and code due.