ACSE: Advanced Compiler System for Education

Software Compilers: Evaluation

Software Compiler Written Exam

- 2 points over a total of 10
- February 1st
- Time: 1h 15m
- Three exercises on the topics covered by lectures
- See the exercises for last year's exam on the web site

Software Compiler Projects: Overview

- 8 points over a total of 10
- Project Rules
- Types of Assignments
- The projects, at last!

Software Compiler Projects: Rules

- Selection: FIFO!
  - Go to my homepage (http://home.dei.polimi.it/agosta)
  - Go to the projects page (link at the end of the course page)
  - Register to the system (login link on the left column)
  - Edit the page and reserve your choice
  - Do **not** overwrite non-blank entries!!!
- How many people per group?
  - Preferred: **2**
  - Also possible: **1**
  - **Not** possible: 3+

Software Compiler Projects: Rules

- What to send in? (standard projects)
  - Start from the standard ACSE distribution, plus all the patches
  - Add your own code, documentation and tests in the appropriate dirs
  - Use `diff -Naur ACSE ACSE-myproject` to build a patch
  - Zip/tar it and send to us
- Documentation:
  - Comment the code using the same style as the rest of the machine
  - Add a new folder in the documentation with a description of your project work in .tex
  - Remember to use/update makefiles
  - Add significant tests to help verification of your project work

Software Compiler Projects: Rules

- Evaluation
  - Early submissions (arriving before January 24th) get a +1 bonus
  - Submissions within February 1st are evaluated normally (maximum mark: 7)
  - Marks for late submissions drop by 1 per week, up to 4
  - Late submissions after February 14th will not get more than 4 regardless of quality

Types of Assignments

- Standard Projects
  - Based on ACSE
  - Conventional (improve the compiler seen in lab)
- Special Projects
  - Based on other compilers
  - More advanced (work on different compilers)
  - You get a +1 mark bonus for a special project

Project 1: Type system extension

- Goal: add support for all C-style numeric types (float, double, long long)
- Syntax: `float` type, `1.0` FP constants, `1L` long constants
- This project requires the modification of all the three components to implement FP arithmetics and 64-bit values

Project 2: Scope management

- Goal: introduce a more complete management of scopes and variables
- Syntax: variables may now be declared at the beginning of any code block (`{}` pair)
- Semantics: variables declared within one block are only visible there
- This project requires the modification of all the three components of the ACSE system, and a re-organization of the Symbol Table

Project 3: Code Generation

- Goal: rewrite the code generation part to produce assembly for a different platform
- This project is available in several flavors – the assembly may be chosen among the following:
  - Java bytecode (project 3)
  - You can propose a different platform (not previously done)
- In addition to the back-end part, you must also implement the switch and break/continue constructs in ACSE

Project 4: Intermediate Representation

- Goal: introduce an intermediate language between front-end and back-end
- Design the intermediate representation as a tree or graph of instructions
- Each instruction will be represented by a data structure
- A visit on the graph or tree will then produce the assembly code
- An API for modifying the intermediate representation must be provided

Project 5-9: Multi-target back-end

- Goal: restructure the back end to support multiple back ends (using those already developed in past projects)
- The back-end used should be selectable via compilation switches or command line options
- Note: past projects may have limits! You'll need to overcome them in some cases
- The project is available in several flavours:
  - External back-ends: LLVM and GNU Lightning
  - LanCE with functions: Arm and x86/nasm (3 students)

Project 10-11: Struct and Union

- Goal: extend the front-end with a Struct and Union constructs
- Syntax: as C struct and union
- In addition to the front-end part, you must also re-write the ACSE back-end to a subset of MACE:
  - Without binary operations (ADDI, SUBI, etc; project 19)
  - Without the Scc operation (project 20)

Project 12: Front-End extensions

- Goal: extend the front-end with the following constructs
  - Typedef (with type checking and casting)
  - Goto/Label
- In addition to the front-end part, you must also re-write the ACSE back-end to a subset of MACE:
  - With accumulator-based arithmetic operations (i.e., dest and src1 must be the same for ADD, SUB, ADDI, etc.;

Project 13: Pointers

- Goal: extend the front-end with the following constructs
  - Pointers
  - Goto/Label
- In addition to the front-end part, you must also target x86

Project 14: Vector Operations

- Support vector operations on fixed length vectors in ACSE
- The back-end, based on the existing x86/NASM back-end, must translate vector operations using SSE extensions

Project 15: Dynamic Memory Allocation

- Support dynamic memory allocation
- The back-end must be based on the existing x86/NASM back-end, with function extensions

Project 17-18: OpenMP Compiler (Special)

- Goal: extend a C source-to-source compiler to support OpenMP constructs
  - Limited to parallel for constructs
  - Translation options: to pthreads (project 17) or to CUDA (project 18)

Project 19-20: CUDA Compiler (Special)

- Goal: extend a C source-to-source compiler to support CUDA constructs
  - Translation options: to pthreads (project 19) or to OpenMP (project 20)

Project 21-22: OpenCL Compiler (Special)

- Goal: extend a C source-to-source compiler to support OpenCL constructs
  - Limited to parallel for constructs
  - Translation options: to pthreads (project 21) or to OpenMP (project 22)

Project 23-24: Cryptolang (Special)

- Cryptolang is a Python-based domain specific language for cryptographic applications development
- Goal: produce a translator from Cryptolang to either VHDL (project 29) or C (project 30)

Project 25: ACSE 2 Front-End (Special)

- ACSE 2 is the ongoing development of an improved version of ACSE
- Goal: complete the frontend support for the new version of the LanCE language