Towards a Demonstrably Correct Ada Compiler

Chris Nettleton, Xgc, www.xgc.com
Wilson Ifill, Colin Marsh, AWE plc, www.awe.co.uk

SIGAda 2007

Objectives

- To deliver an Ada compiler
  - For applications offering unprecedented levels of reliability and security
  - That is
    - Cost effective
    - Efficient
    - Reviewable (compiler is reviewable)
    - Demonstrably correct

Background

- AWE is “is one of the largest high technology research, design development and production facilities in the country”
- AWE funded DeCCo work, to develop a Demonstrably-Correct Compiler for subset Pascal
- XGC offers mission-critical Ada compilers
- XGC develops New Technology

The DeCCo Work

- Logica's formal methods team funded by AWE
- Z spec of subset Pascal
- Z spec of target ASP
- Z spec of transformations, Pascal -> ASP
- Formal mapping of Z transformations to Prolog DCTG
- PASP compiler in Prolog

XGC New Technology

- Internal feasibility study to fix all compiler problems
- To answer questions such as:
  - How suitable?
  - How fast?
  - How reviewable?
- Demonstrator already written

Previous work

- When we first looked at this, in 1988, our computer had 4MByte and ran at 1 MIPS
- It was a VAX 780
- We used Prolog (as in Poplog)
- Very slow using lots of memory
- Unsuitable for production use

Present Work

- Work is part funded by AWE plc
- Feasibility study complete (April 2006)
- SIGAda paper
- We have a prototype Subset Ada compiler

The Need for a New Compiler

- Today's compilers designed in 1980s
- Optimizing, and not reviewable
- Written in C, Ada, …
- Compiler is > 1,000,000 lines
- Compilation process insecure

Options

- Compiler has zillions of options
- Some options change the generated code
- Some options are dangerous
- Compiler is validated but not used with the options is was validated with
- Compiler has environment variables that select versions etc

Separate Compilation

- Disadvantages
  - allows units to be compiled with different options
  - allows a program to be linked with object code from different compilers
  - compiler ships with precompiled libraries and third-party code
  - compiler finds its source files using search lists

Generated Code

- Optimizations make generated code difficult to understand
- Compiler performs dangerous optimizations that rely on alias analysis
- Use of static link
- Saves and restores registers - therefore variables have > one home
- Non-trivial transformations that are difficult to demonstrate correct
- Register allocation with spilling

Generated Code

- Allows recursion across units, linkers don't check this
- Compiler fails to flag or remove unreachable code
- Stack space allocated in unpredictable manner, which requires Storage_Error checks
- Compiler generates unused subprograms
- Compiler generates unreachable code
- Obscure instruction sequences designed to reduce time

The Way Forward

- a well-defined HLL
- a well-defined compiler
- a well-defined target computer
- Demonstration of correctness
- Proof of correctness (later)

The Well-Defined HLL

- Unambiguous subset of Ada 95
  - AWE will not use floating point
- SPARK
- Will need a formal definition (in Z)
  - Maybe adapt Pascal spec

The Well-Defined Compiler

- Compiler has a unique identifier and is one program
  - No separate assembler or linker
  - No intermediate files
- Has no options that effect the generated code
- Always compiles from given source to binary program
  - No implicit files
  - No libraries
  - No search paths

The Well-Defined Target

- COTS products preferred (MoD policy)
- AWE have specified the ASP in Z
- IP offerings such as ARM, ZAP, but must be simple
- Pentium not an option (we have no spec) also much too complex

Correctness

- We will demonstrate correctness
  - a given Ada program has been correctly compiled
  - all programs will be correctly compiled
  - program synthesis?
- We would like to prove correctness
  - theorem proof, with tool support

New Technology

- Written in functional programming language
- No optimizations
- Compiler is one program
- ~ 5000 equations
- Many passes (> 20)
- Source can be supplied

Code quality achieved by

- Variables allocated statically (no recursion)
- Algebraic simplification
- Commoning within Ada statements
- Constant folding
- Flow analysis
- Simple register allocation

Static Analysis

- Compiler performs sufficient static analysis
  - Lexical analysis
  - Syntax analysis
  - Semantic analysis (largest part of compiler by far)
    - Identifies bad Ada
  - IL control flow/data flow analysis
    - Unreachable code
    - Uninitialized variables (pragma Normalize!!!)

No problems with …

- Aliasing, kill sets on assignments
- Stacks, frames, stack overflow
- Saving registers across calls
- Up-level references
- Unreachable code
- Unused subprograms

Results

- Are optimizations important? (we use algebraic simplifications)
  - Code size, about the same
  - Execution time, slightly slower
  - Reviewability, much much better, with “direct positional correspondence”

Results

- compiler size
  - About 6 M Byte
- compiler speed
  - For small to medium programs (<= 30,000 lines) > 1000 lines per second
- Demonstrations of correctness
  - None as yet
- Proof
  - None as yet