Design and Implementation of a TriCore Backend for the LLVM Compiler Framework

Studienarbeit

Christoph Erhardt

Friedrich-Alexander-Universität Erlangen-Nürnberg

November 20, 2009

Overview

Motivation

- TriCore chips are omnipresent around here:
  - Quadcopter
  - High striker
  - Carolo Cup
  - ...
- The RTSC (Real-Time Systems Compiler) project:
  - Operating system aware compiler for real-time applications
  - Processes atomic basic blocks
  - Based on LLVM

Motivation

- TriCore chips are omnipresent around here:
  - Quadcopter
  - High striker
  - Carolo Cup
  - ...
- The RTSC (Real-Time Systems Compiler) project:
  - Operating system aware compiler for real-time applications
  - Processes atomic basic blocks
  - Based on LLVM
- RTSC should be able to generate TriCore machine code

Three-in-one architecture

- Real-time microcontroller unit
- DSP
- Superscalar RISC processor

The TriCore Processor Architecture

Three-in-one architecture

- Real-time microcontroller unit
- DSP
- Superscalar RISC processor

Basic features

- Load/store architecture
- 32-bit data, address, and instruction words
- Some special 16-bit instruction words for higher code density
- Little-endian byte order
- 16 data + 16 address registers

Peculiarities

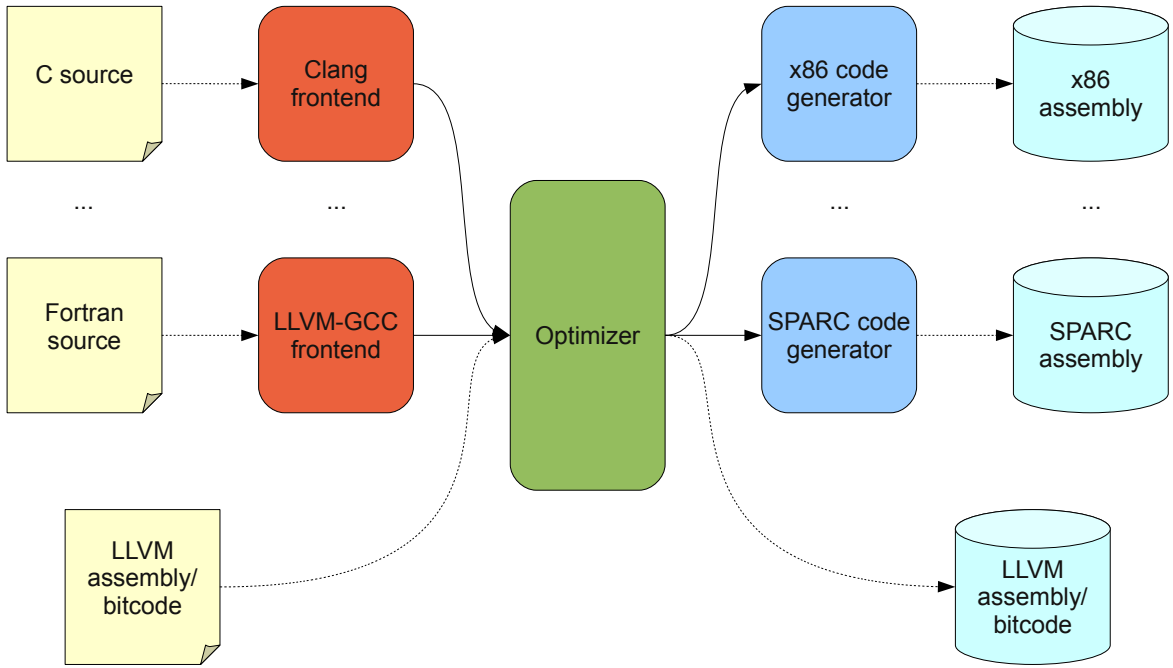
Some things that TriCore handles in an unusual way

- Strict distinction between data and address registers:
  - Also reflected in the calling conventions
  - Serious problem for the compiler!
- Data registers are also used for floating-point operands
- Special DSP-oriented instructions and addressing modes
- Task/context model:
  - Automatic context save/restore upon call/return
  - Context save areas (linked lists managed by hardware)

The LLVM Compiler Infrastructure

- Open-source compiler infrastructure project started in 2000
- Main sponsor: Apple Inc.
- Written in C++

Basic Architecture



- Language-specific frontends
- Optimizer: generic IR, analysis/transformation passes
- Several backends for machine code generation

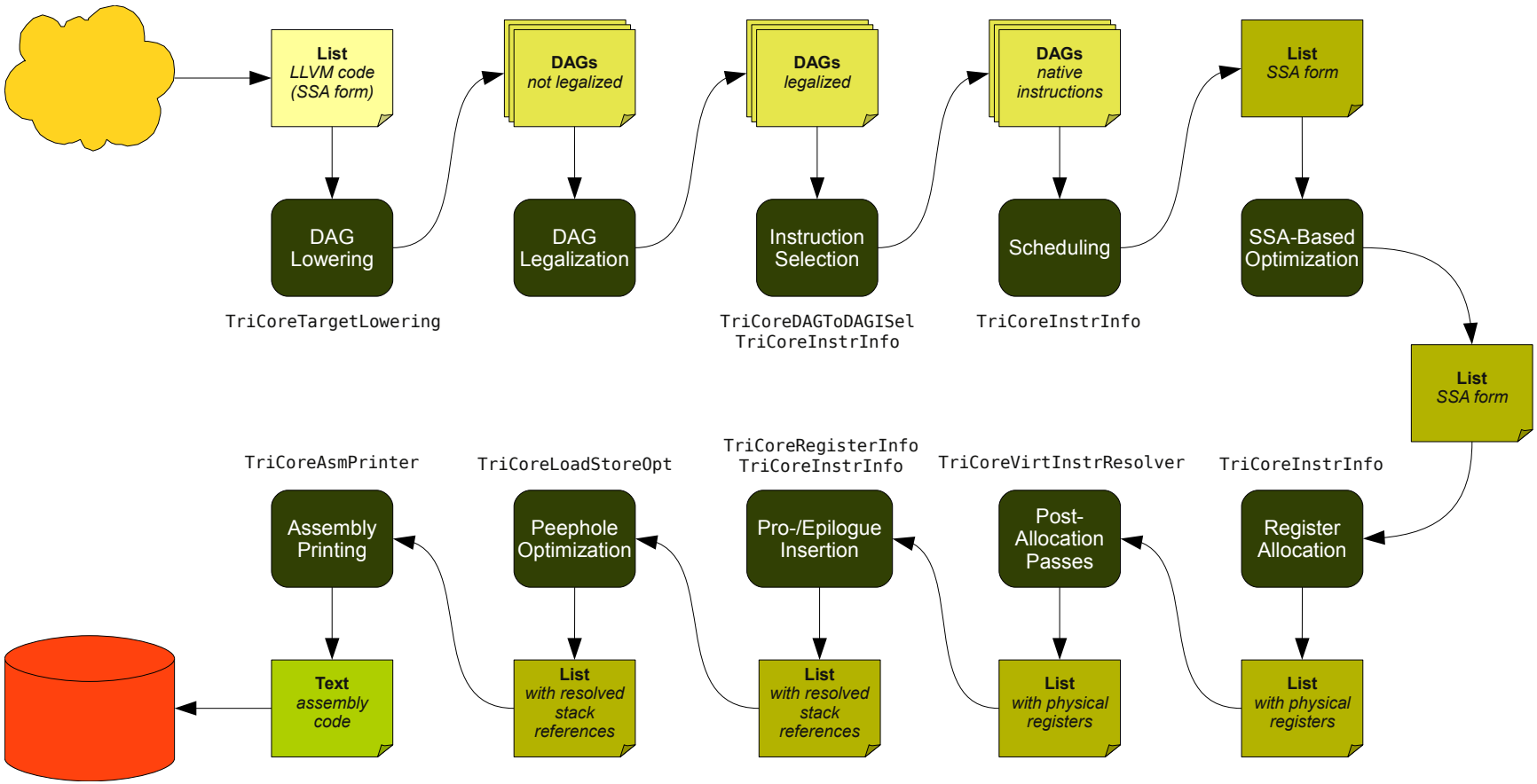
Unique Characteristics

- Not merely a compiler, but a *compiler infrastructure*:
  - Static compilation
  - Just-in-time compilation
- Strictly modular, library-based architecture:
  - Easily extendible
  - Possibility to incorporate parts of LLVM in other projects
- BSD-style licence
- Produces highly optimized machine code in an efficient way:
  - Memory-efficient
  - Time-efficient

Design and Implementation of the Backend

- Extensive generic code generation framework:
  - Makes work a lot easier
  - ... but also imposes some problems in specific cases
- Fixed class hierarchy
- Many target-independent algorithms:
  - Instruction scheduling
  - Register colouring
  - ...
- Code generation process executed by a series of passes

Code Generation Process



Problem

- Backend contains large portions of descriptive data
- C++ obviously not suitable

TableGen

Problem

- Backend contains large portions of descriptive data
- C++ obviously not suitable

TableGen

- Language for domain-specific modelling
- Similar to object-oriented approach:
  - Classes, records (objects), attributes
  - Inheritance
- Definition files (`.td`) preprocessed by `tblgen` tool
  $\rightarrow$ Auto-generation of C++ code
- Used for description of:
  - Subtargets, registers
  - Calling conventions
  - Instruction set

SelectionDAG Construction

Directed acyclic graph

- Per basic block
- Nodes: instructions
- Edges:
  - Data dependencies
  - Control flow dependencies

Example

```
%mul  = mul i32 %a, %a
%mul4 = mul i32 %b, %b
%add  = add nsw i32 %mul4, %mul
ret i32 %add
```

SelectionDAG Construction

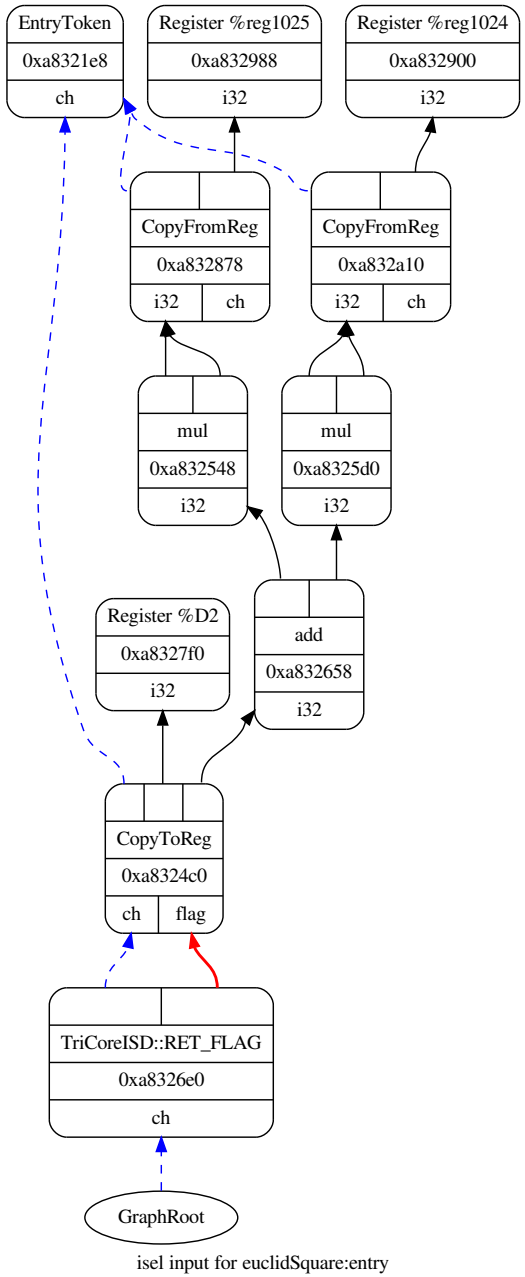
Largely automated

Directed acyclic graph

- Per basic block
- Nodes: instructions
- Edges:
  - Data dependencies
  - Control flow dependencies

Example

```
%mul  = mul i32 %a, %a
%mul4 = mul i32 %b, %b
%add  = add nsw i32 %mul4, %mul
ret i32 %add
```



Troubles

Problem

- TriCore strictly distinguishes between addresses and data integers
- Have to be put into separate register files → calling conventions!
- LLVM's backend framework treats pointers just like integers...

Troubles

Problem

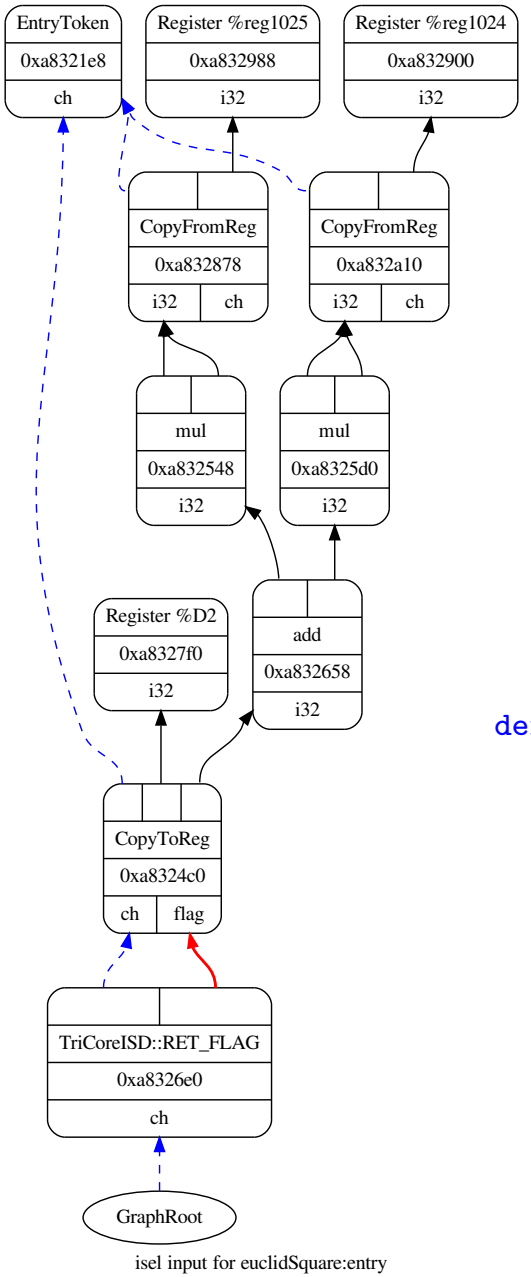
- TriCore strictly distinguishes between addresses and data integers
- Have to be put into separate register files → calling conventions!
- LLVM's backend framework treats pointers just like integers...

Solution

- Annotation of "pointer / no pointer" flag in value type class
- Promotion of this flag throughout the DAG construction phase (required some hacks...)
- Case differentiations in all relevant situations

Instruction Selection

Largely auto-generated



isel input for euclidSquare:entry

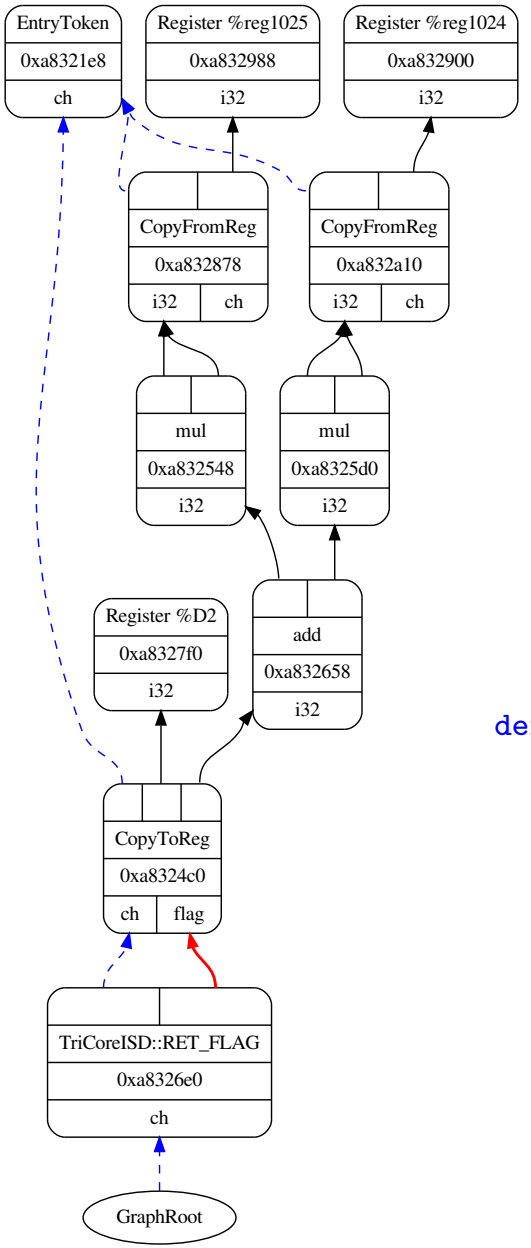
Pattern matching

$\longrightarrow$

```
def MULrr2 : Rr2Instr<0x0a,
  (outs DR:$c), (ins DR:$a, DR:$b),
  "mul\t$c, $a, $b",
  [(set DR:$c, (mul DR:$a, DR:$b))]>;
```

Instruction Selection

Largely auto-generated

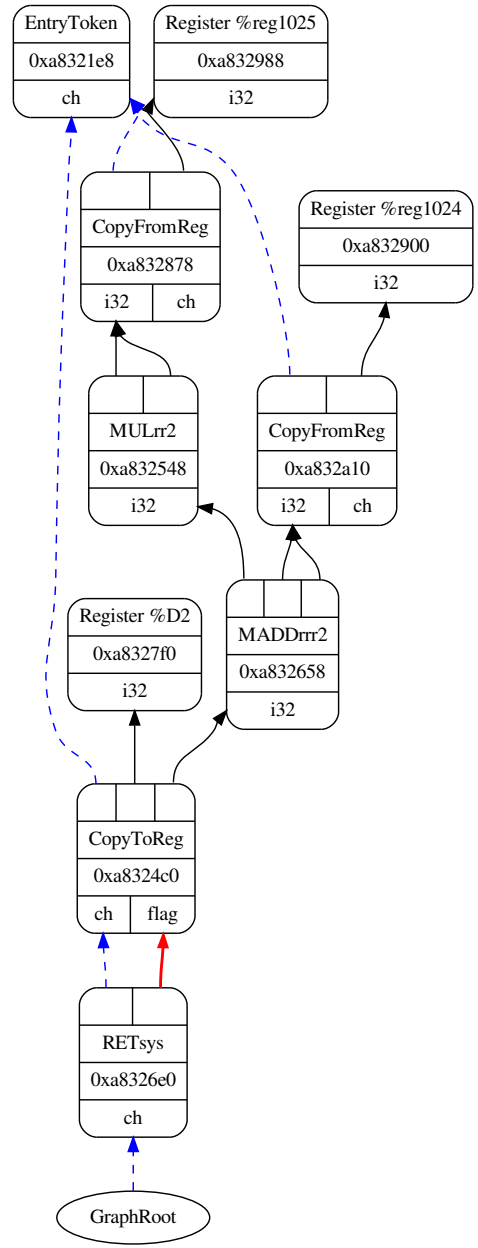


isel input for euclidSquare:entry

Pattern matching

$\longrightarrow$

```
def MULrr2 : Rr2Instr<0x0a,
  (outs DR:$c), (ins DR:$a, DR:$b),
  "mul\t$c, $a, $b",
  [(set DR:$c, (mul DR:$a, DR:$b))]>;
```



scheduler input for euclidSquare:entry

Scheduling

- DAGs $\rightarrow$ list (SSA form)
- Target-independent algorithm using data from the instruction description table

Scheduling

- DAGs → list (SSA form)
- Target-independent algorithm using data from the instruction description table

Register Allocation

- Virtual registers → physical registers
- SSA deconstruction
- Target-independent colouring algorithm using the register information table

Final passes

Virtual Instruction Resolution

- Some instructions (e. g., moves) had operands of unknown register classes at the time of their creation
- Now that physical registers have been assigned, these instructions can be resolved

Final passes

Virtual Instruction Resolution

- Some instructions (e. g., moves) had operands of unknown register classes at the time of their creation
- Now that physical registers have been assigned, these instructions can be resolved

Pre-/Epilogue Insertion

- Insertion of pre-/epilogue code to entry/exits of all functions
- Virtual stack slots $\rightarrow$ physical stack frame references

Code Emission

Peephole Optimization

Merging of two subsequent 32-bit loads/stores into a single 64-bit load/store

```
# Before:
st.w [%a10]4, %d9
st.w [%a10]0, %d8
```

```
# After:
st.d [%a10]0, %e8
```

Code Emission

Peephole Optimization

Merging of two subsequent 32-bit loads/stores into a single 64-bit load/store

```
# Before:
st.w [%a10]4, %d9
st.w [%a10]0, %d8
```

```
# After:
st.d [%a10]0, %e8
```

Assembly Printing

- Output of assembly code in text form
- Large parts auto-generated from the instruction description table

In practice

Required software

- Clang compiler frontend (support has been integrated)
- GNU Binutils for TriCore:
  - Assembler
  - Linker
- Headers and libraries from TriCore-GCC
- Small Perl wrapper script

Comparison to GCC

Criteria

1. Compilation speed
2. Code size
3. Code performance

Comparison to GCC

Criteria

1. Compilation speed
2. Code size
3. Code performance

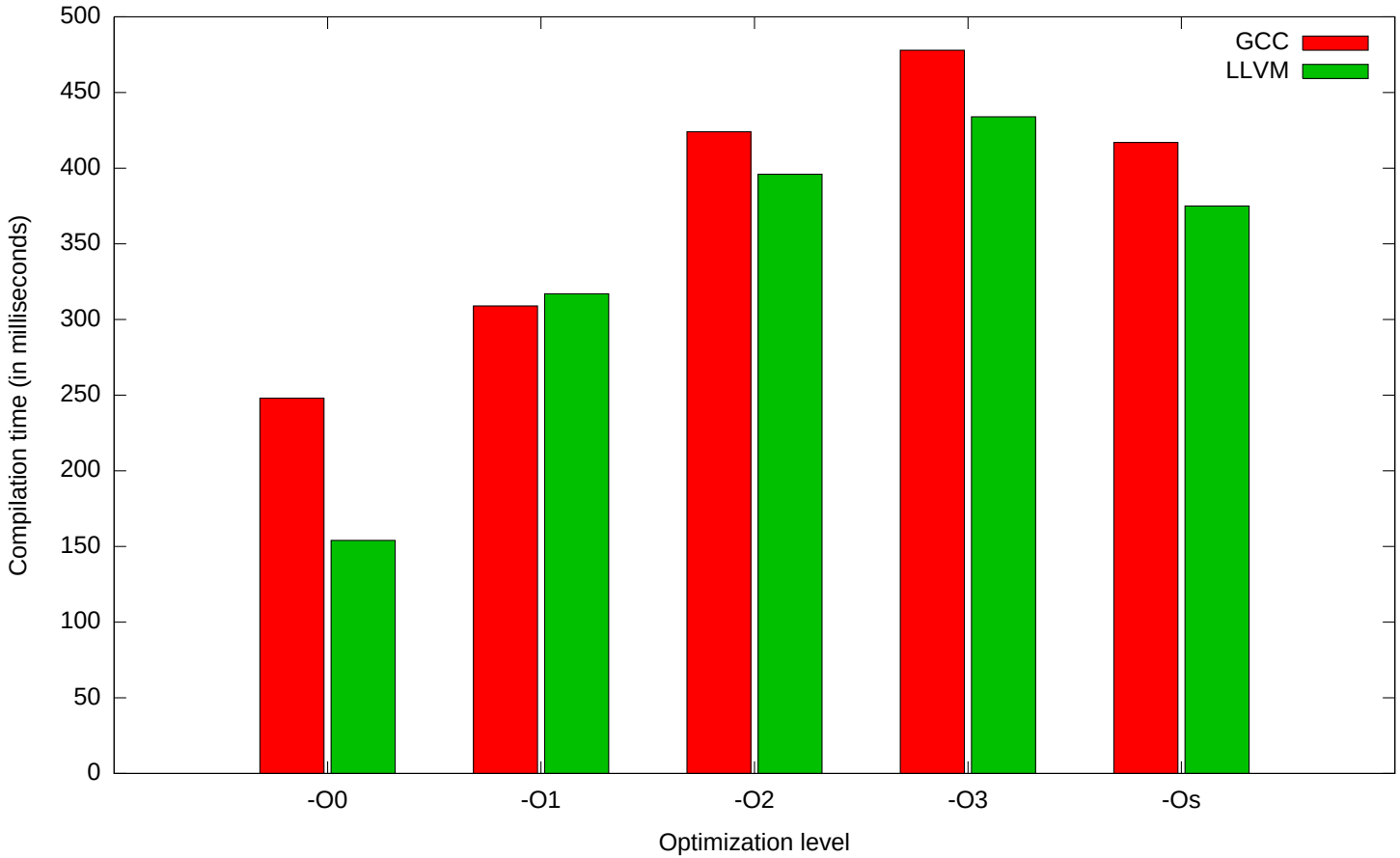
Testing system

- Benchmark application: CoreMark
- Compilation PC: Core 2 Quad Q6600 (2.40 GHz)
- Runtime system: TC1796 board (40 MHz)

The CoreMark Benchmark

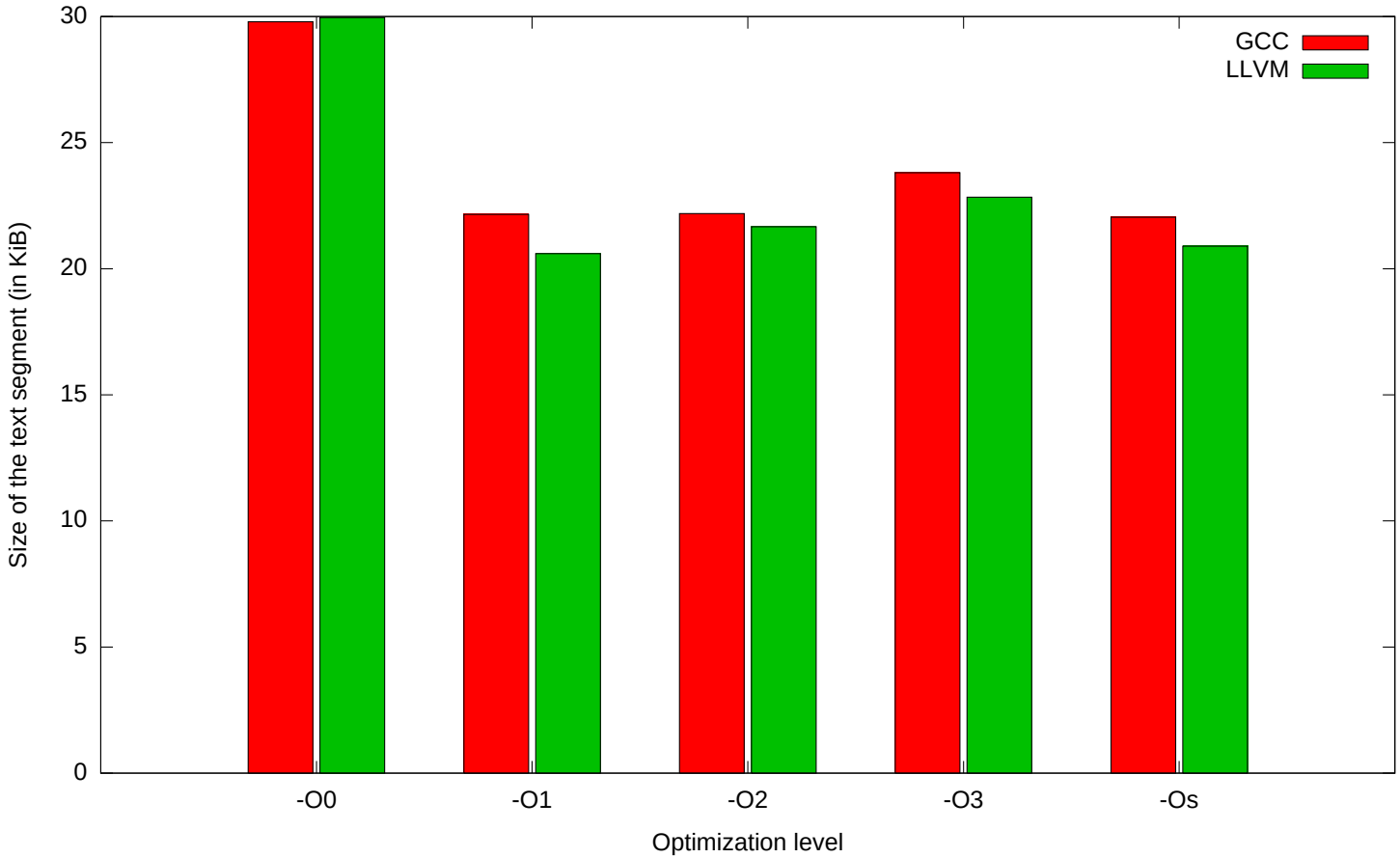
- Alternative to the well-known Dhrystone benchmark
- C source code publicly available (albeit not *open source software*)
- Easily portable
- Prevents compilers from "cheating" by optimizing away unused computation results
- Operations:
  - Linked list processing
  - Matrix manipulation
  - State machine operations
  - CRC computation
- Results can be validated

Compilation Time



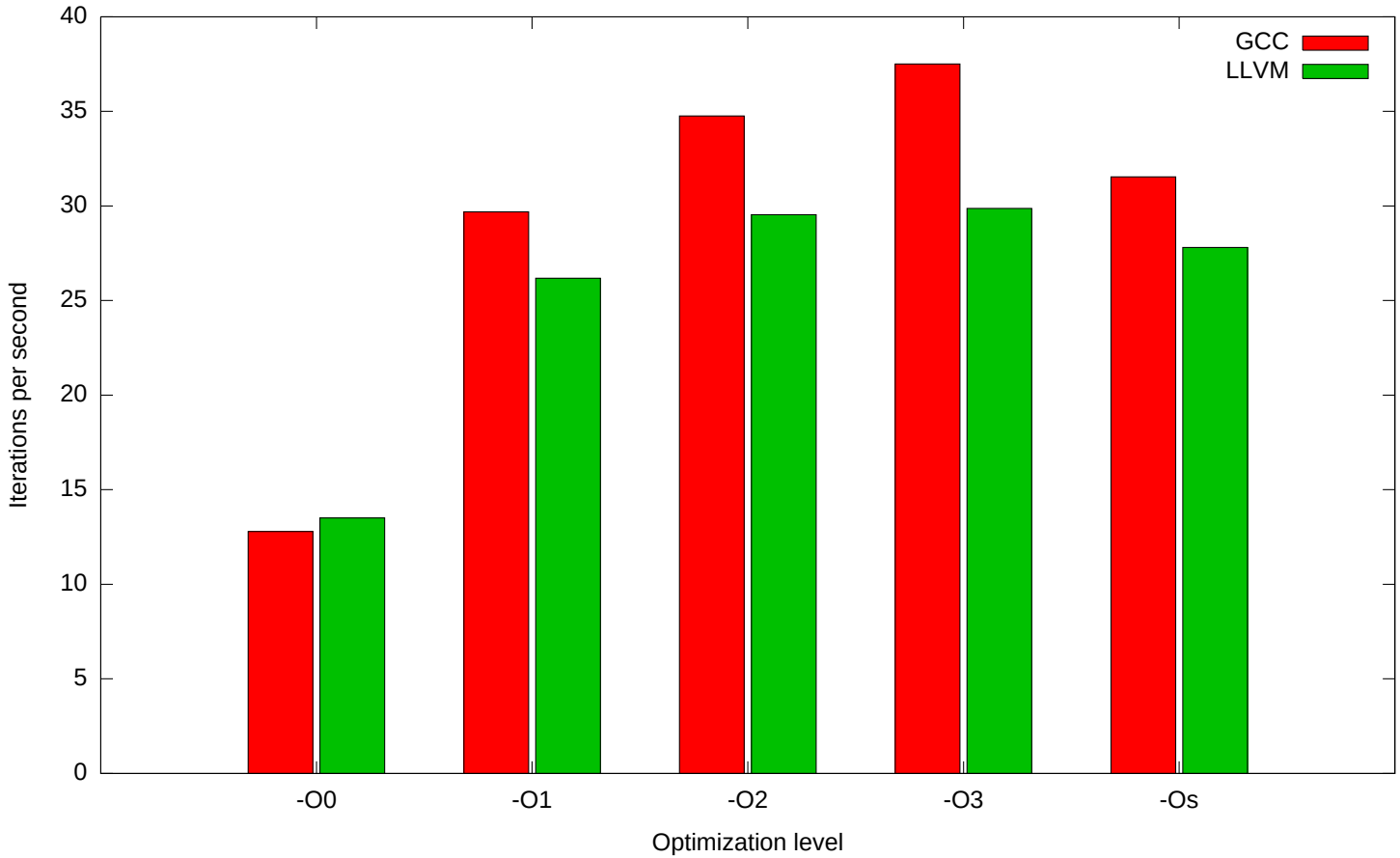
- Takes about 10 % less time than GCC
- Even faster when compiling at -O0

Code Size



Generates slightly smaller code

Code Performance



- Code 12–20 % slower than the code generated by GCC
- Further work needed to become fully competitive

Summary

- All of the basic functionality is there and is working reliably
- Space for further optimizations and extensions
- First TriCore compiler to be released under a BSD-style licence!
- End goal: inclusion into LLVM's repository

The End

Any Questions?