

C/C++ Clang/LLVM Compiler Adaptation for Customer's Processors Based on its Own Architecture



ORGANIZATION: Chipmaker



INDUSTRY: semiconductors

PROBLEM STATEMENT:

The customer specializes in the development and manufacture of microprocessors with DSP support of own proprietary architecture cores. There is a full set of development tools based on GCC compiler and GNU bitnutils complex, however, the compiler is limited in code generation with RISC instructions of own proprietary architecture core only. The development for vector co-processors for this core can only be performed with an assembler.

Grovety studied Clang LLVM compiler features to find out whether automatic vectorization could be applied to customer architecture. According to their findings, Clang and LLVM framework normally support operations with vectors and algorithms of automatic loop unrolling for vectors with a size of up to 1024 elements for 1-bit vectors; 256 elements for 8-bit integer vectors; 128 elements for 16-bit integer vectors; 64 elements for 32-bit integer vectors; 32 elements for 64-bit integer vectors; 16 elements for 32-bit float vectors; and 8 elements for 64-bit double vectors.

The compatibility check of the source codes for Inverse, nmblas, and nmpls libraries showed that automatic vectorization for vectors with a size of 2 is performed every time optimizations for arm64 and x64 platforms are called directly in almost all source codes. If deeper vectorization is forcefully specified, then the compiler can generate code with vectors up to 64 elements. Most likely those limitations in automatic mode on arm64 and x64 platforms are related to operation "weights" specified in code generation rules for a particular platform. These "weights" can be changed for new platforms.

The customer assigned us the task of adapting the LLVM Clang compiler and developing specialized optimizations as tools for automatic code generation using vector functionality provided by its own architecture.

SOLUTION:

Within the project timeframe (5 months, team: architect and 2 developers), the following tasks were completed:

Compiler adaptation:

- Porting Grovety solutions in CHAR32 mode from LLVM Clang 3.4 version to LLVM Clang 10.0 version

The current LLVM Clang version supports the latest C/C++ standards and provides an advanced set of optimizations. Current frontend version with CHAR32 support is based upon LLVM Clang 4.0 was ported to up-to-date version.

- Development of basic code generation mechanisms

Development of code generator for customer platform "from scratch" which provides RISC subsystem functionality including:

- Definition of registers;
- Definition of scalar instructions;
- Definition of calling convention;
- Rules for assembler code generation.

- Support of vector code generation
Code generator elaboration and implementation of vector instruction generation including:
 - Support of vector types;
 - Generation of vector instructions for float operators;
 - Generation of vector instructions for integer operators.

- Stabilization

The largest part of the compiler adaptation task is the stabilization of code generation mechanisms based on a preliminary developed test suite.

Extension of the test suite:

- Developing testing system based on LLVM Test-suite
To guarantee the quality of the resulting compiler, LLVM Test-suite is adjusted to check the functionality of the developed compiler and to automate the test process.

RESULT

The customer received a new version of Clang/LLVM for C/C++ languages featuring assembler code generation for software-compatible processors, built on its own architecture.

TOOLS & TECHNOLOGIES:

Linux; C/C++; Clang/LLVM; customer architecture



grovety.com



hi@grovety.com