Porting EagleEye to a LLVM-based toolchain

Daniel Cederman (Presenter)
Daniel Hellström (Project Manager)
Jorge Lopez Trescastro (ESA Technical Officer)

Activity

Overview

- Improved Clang/LLVM Sparc/LEON backend
  - improvements to integrated assembler
  - more performance optimizations
  - improved timing information
  - missing features added
  - make Clang a valid alternative or complement to GCC
    - without sacrificing performance or features
- ESA EagleEye CSW used as input
  - All validation tests pass
- LLVM-based compiler for RTEMS-5
  - RCC-1.3 (or through open-source repositories)

Activity

Overview

- One year activity (Jan 2018 – Dec 2018)

- MTR-1 (Mar 2018)
  - Test plan and requirement specification [D1, D2]
- MTR-2 (Nov 2018)
  - Improved Clang/LLVM Toolchain for RTEMS-5 [SW1]
  - Description and design of toolchain improvements [D4]
  - User manual with build instructions [D3, D6]
  - Test reports for EagleEye system, AOCS acceptance, Compiler comparison, RTEMS/Newlib [D5, D7, D8, D9]
- AR (Dec 2018)
  - Final report and presentation [FR, FP]

Related work

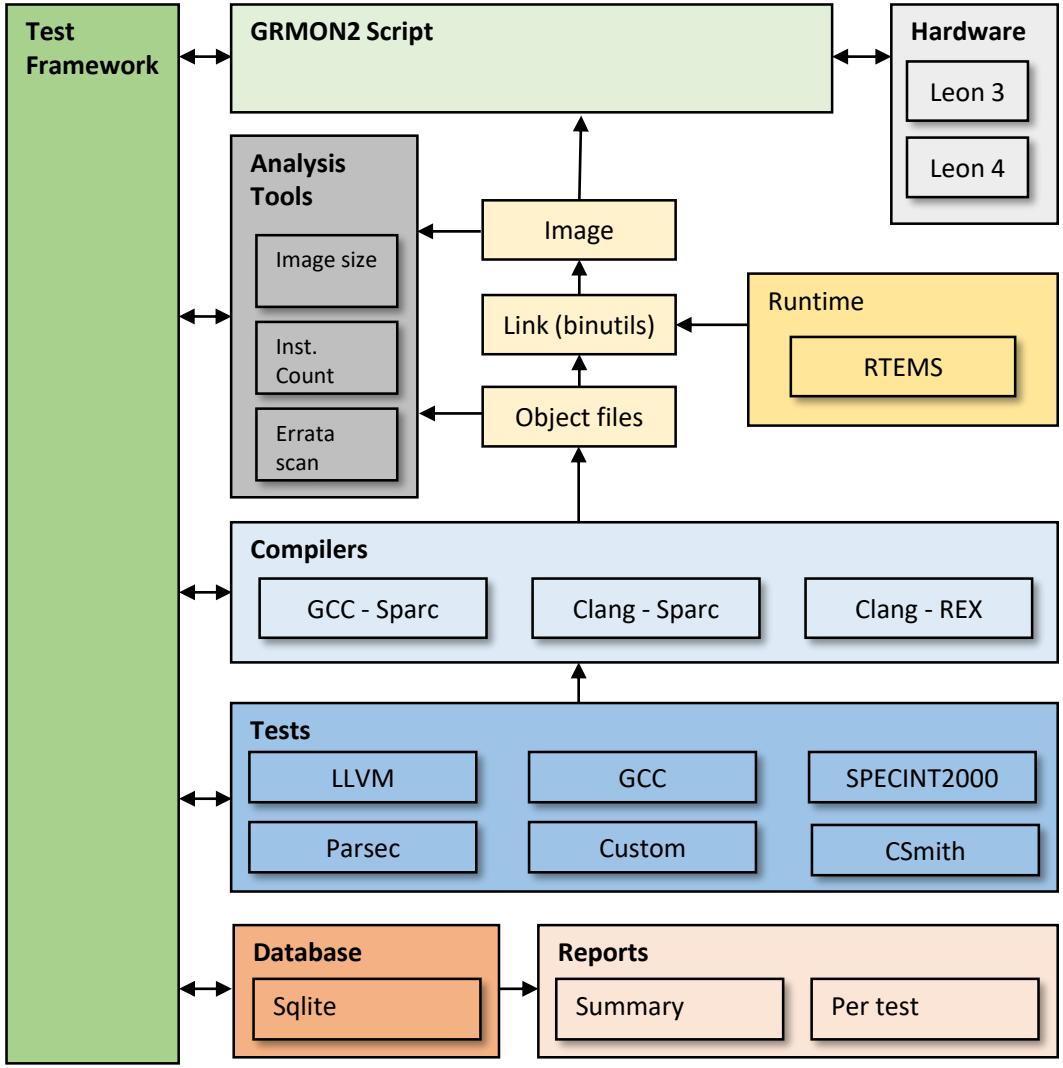
Overview

- LLVM for in flight SW development and validation process (2015)
  - Improved correctness for Sparc backend
  - Better support for Leon processors
  - Cobham Gaisler developed test suite framework
- Definition and Design of Software Components for LEON3 FT Microcontroller (GR716) and LEON-REX Instruction Set Architecture (2016)
  - Cobham Gaisler extended Sparc backend with LEON-REX support
  - Test suite framework reused

01
Tests and requirements

LEON Toolchain test suite

Performance and correctness evaluation plan



Test suite updates

Performance and correctness evaluation plan

- Runtime now uses RTEMS-5 (RCC-1.3)

- Improved quality of tests
  - Remove tests triggering undefined behavior
  - Fix uses of uninitialized memory
  - Remove tests where Clang gives error instead of warning
  - …

- Generic updates to test framework [SW1]

Test plan

Performance and correctness evaluation plan

- Test suite configurations (~6000 test programs)
  - With performance (-O3), size (-Os), or no optimization (-O0)
  - With flat register window mode
  - With errata workarounds
- Benchmarking configurations (160 benchmarks)
  - GR740 (quad-core LEON4) and GR712 (dual-core LEON3)
  - Measure cache misses (D/I)
  - Measure write buffer holds
  - Measure branch prediction misses
- Developed unit tests for problems found

Features/Correctness requirements

Requirement specification

- Inputs to requirement specification
  - The performance and correctness evaluation plan
  - Requested features
  - List of known problems from the SoW
  - Compiling EagleEye, XtratuM, and LVCUGEN
  - RTEMS, VxWorks 7, and the Linux kernel

Performance requirements

Requirement specification

- Source code analysis of the LLVM Sparc backend and associated components
  - Missing features
  - Missing information to middle-end
  - Other missing support routines
- Comparative analysis of Clang and GCC output
  - Analysis of benchmarks
  - Automated performance comparison of autogenerated code with csmith and creduce

Deliverables

Tests and requirements

- Performance and correctness evaluation plan [D1]
- Comparison tests (Source code and scripts) [SW1]
- Requirement specification document [D2]

02

LLVM/Clang toolchain upgrade

Overview

LLVM/Clang toolchain upgrade

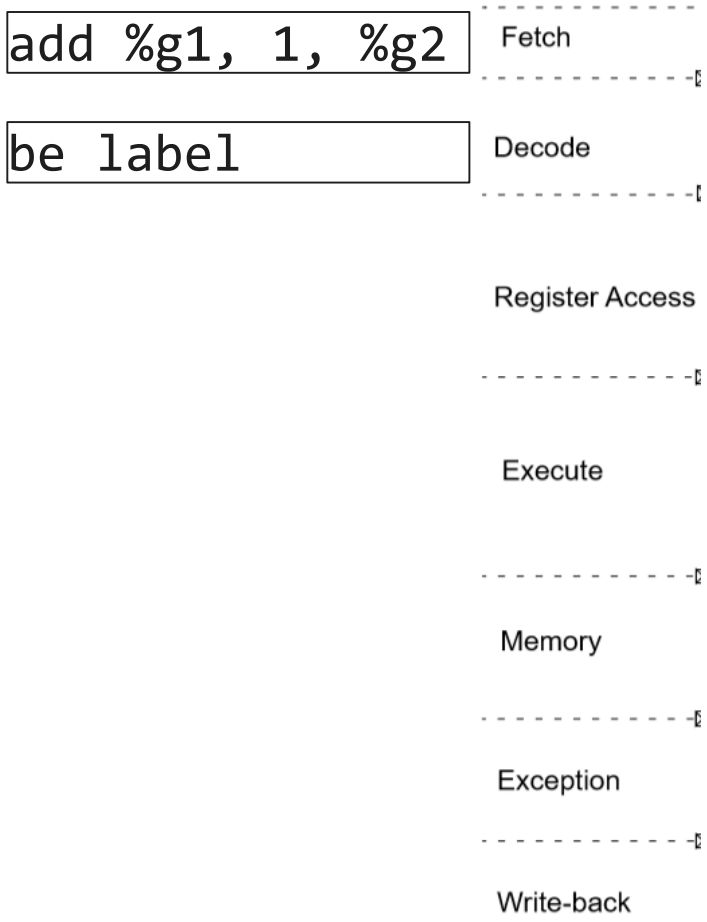
- Optimizations
- Flat register window mode
- Errata workarounds
- Assembler improvements
- Bug fixes

- Toolchain driver (RCC-1.3)

Use annulling branches

Performance

- Branch instructions have a delay slot
- Avoids stalling fetch stage



- Not always possible to find candidate
- LLVM inserts NOP in delay slot

```
be label
 add %g1, 1, %g2
```

Use annulling branches

Performance

- If unable to fill delay slot, try branches with annulled delay slot

- For branch always the delay slot can be ignored
- Reduces code size

```
ba .LBB1
 nop
```



```
ba,a .LBB1
```

Use annulling branches

Performance

- Conditional branches use delay slot when branch taken
  - Copy branch target instruction to delay slot
  - Then skip it when branching

```
.loop
sub %o0, 12, %o0
…
tst %o0
bne .loop
 nop
clr %o0
```



```
.loop
sub %o0, 12, %o0
.loop_1
…
tst %o0
bne,a .loop_1
 sub %o0, 12, %o0
clr %o0
```

Use annulling branches

Results

Change	Number of benchmarks
>= %10 faster	**3**
>= %5 faster	**7**
>= %1 faster	**48**
No change	110
>= %1 slower	2
>= %5 slower	0
>= %10 slower	0

- 160 benchmarks in total

Use ANDN to encode constants

Performance

- Found with csmith/creduce
- x & 0x7fffff => x & ¬0xff800000

```
sethi %hi(0x7ffc00), %g1
or %g1, 0x3ff, %g1
and %o0, %g1, %o0
=>
sethi %hi(0xff800000), %g1
andn %o0, %g1, %o0
```

Use ANDN to encode constants

Results

Change	Number of benchmarks
>= %10 faster	0
>= %5 faster	0
>= %1 faster	1
No change	158
>= %1 slower	1
>= %5 slower	0
>= %10 slower	0

- 160 benchmarks in total

Anti-dependency breaker

Performance

- Register allocator tries to reuse registers as much as possible
- Can create unnecessary dependencies

- For example
  - fadds %f0, %f2, %f2
  - ld [%fp], %f0
  - fadds %f0, %f1, %f3

- Load instruction forced to wait for fadds

- Anti-dependency breaker runs after register allocation
- Tries to avoid immediately reusing registers

Anti-dependency breaker

Results

Change	Number of benchmarks
>= %10 faster	**14**
>= %5 faster	**31**
>= %1 faster	**66**
No change	78
>= %1 slower	16
>= %5 slower	2
>= %10 slower	1

- 160 benchmarks in total

Use cc instructions when possible

Performance

- Comparisons with 0 can be replaced by cc version of instruction

- For example:

```
add %g2, %g1, %g1
cmp %g1, 0
bne NotZero
 nop
```

```
addcc %g2, %g1, %g1
bne NotZero
 nop
```

```
subcc %g1, 0, %g0
```

Use cc instructions when possible

Results

Change	Number of benchmarks
>= %10 faster	**1**
>= %5 faster	**6**
>= %1 faster	**32**
No change	125
>= %1 slower	3
>= %5 slower	0
>= %10 slower	0

- 160 benchmarks in total

Prioritize few instructions in loops

Performance

- Loop strength reduction pass
- Uses heuristics to search among loop candidates
- Sparc has a high number of registers
- Branches are cheap
- Prioritize few instructions

Prioritize few instructions in loops

Results

Change	Number of benchmarks
>= %10 faster	**0**
>= %5 faster	**8**
>= %1 faster	**29**
No change	126
>= %1 slower	5
>= %5 slower	0
>= %10 slower	0

- 160 benchmarks in total

Improved scheduling information

Performance

- Avoid branch interlock
  - Avoid branch directly after cmp
- Avoid data delay
  - Do not access data directly after it has been loaded into a register
- Avoid mul delay
  - Do not access result directly after multiplication
- Avoid write buffer hold
  - Spread out store instructions
- All FPU instructions except division and square-root are pipelined
  - Not enabled for GR716 (uses GRFPU Lite)

Improved scheduling information

Results

Change	Number of benchmarks
>= %10 faster	**9**
>= %5 faster	**25**
>= %1 faster	**77**
No change	65
>= %1 slower	18
>= %5 slower	2
>= %10 slower	1

- 160 benchmarks in total

Flat register window mode

Features

- No save/restore
- If %i or %l register is used:
  - Store to stack in prologue
  - Restore from stack in epilogue
  - Standard method used by other architectures
- Enabled with –mflat flag

GR712RC errata workarounds

Features

- New compiler pass
  - Inserts NOPs to prevent sensitive sequences
    - For example std followed by std
  - Prevents stores and div/sqrt in delay slot
- Enabled when
  - compiling for –mcpu=gr712rc
  - or using –mfix-gr712rc (for compatibility with GCC)

GR712RC errata workarounds

Lessons learned

- Implemented for both GCC and Clang by Cobham Gaisler
  - Reused design from GCC
- Simple pass - no change in logic, only timing
- Working with LLVM compared to GCC
  - Easier to understand internal representation
  - More accessible API
  - Possible to write pass specific unit tests
  - Informative commit messages
- Straightforward to implement the pass in both compilers
  - Would start with LLVM for more complex passes

Assembler

Features

- Add support for the cycle counter available in GR740 and GR716
  - __builtin_readcyclecounter() returns %ASR23
- Add support for the partial write PSR instruction
  - No need for trap

- Syntax, directives, aliases, and fixups
  - Allow symbols for the immediate field
  - Add .uahalf and .uaword directives
  - Allow tls_add/tls_call syntax in assembler parser
  - Support relocatable expressions in the assembler
  - Add fixups for the GOTdata_op model
  - Add mnemonic aliases for flush, stb, stba, sth, and stha
  - …

RCC/BCC

Features

- New Gaisler toolchain driver for RCC and BCC
  - RCC and BCC can use same compiler
- Multilib selection based on compiler flags
  - Matches GCC's behavior
- New "-qbsp=" flag to select BSP libraries and include files
  - Available for both BCC and RCC
- Uses integrated assembler by default
- Enables leaf optimization by default

03
Results

Toolchain testsuite results

Results

- The toolchain test suite ran 6418 tests in 5 configurations

Test	Description
inline-2-6x, alloca-1-x, vararg-1, vararg-2, pr51354	Combines alignment and dynamic stack allocation.
pr63594-2, vector-compare-1	Uses a vector type consisting of two integers. The Sparc backend uses this special type to represent a 64-bit integer internally and due to this special handling it generates bad code.

Benchmarks

Results

- Comparison between Clang and GCC before and after activity

Change	Before	After
>= %10 faster	15	21 (+6)
>= %5 faster	19	39 (+20)
>= %1 faster	33	**63** (+30)
No change	17	32 (+15)
>= %1 slower	109	**64** (-45)
>= %5 slower	84	32 (-52)
>= %10 slower	57	20 (-37)

- 160 benchmarks in total

RTEMS/Newlib upgrade

RTEMS toolchain

- RTEMS and the RTEMS test-suite
  - Compiled with UP, SMP, Flat, Soft-float
  - Tested on GR740, GR712RC, TSIM
- Patches to kernel, build-script, and tests
  - Upstreamed or pending to be upstreamed to official repository
- Newlib required no patches
- User manual with LLVM instructions

04
Conclusions

Future work – CG roadmap

Conclusions

Mission: To support all recent LEON devices on all OSes

Past releases:

- BCC-2 LLVM/Clang 4.0 (mid 2017)
- RCC-1.3 LLVM/Clang 8.0 (November 2018)

Short-term roadmap:

- Upstream existing optimization patches to llvm/clang and RTEMS master (on-going)
- More optimizations are under development (on-going)
- Update RCC-1.3 to LLVM/Clang 8.0 stable-release once available (2019)
- Release VxWorks 7 LLVM/Clang 8.0 toolchain (2019)
- Rebase LEON-REX backend to LLVM/Clang master and resubmit it
- Update BCC-2 to Clang 8.0 with LEON-REX support

Long-term roadmap:

- Linux toolchain
- LEON5 multi-issue pipe-line will require updates
- Investigate optimizations for size (-Oz)

Conclusions

Conclusions

- The project was completed on time with all deadlines met
- Clang/LLVM can be used as alternative or complement to GCC
  - can compile EagleEye
  - better code on many benchmarks
  - more permissive license
  - easier to integrate with other tools
- LLVM toolchain for RTEMS-5 available for ESA space projects
  - Download precompiled RCC-1.3
  - or through open-source repositories