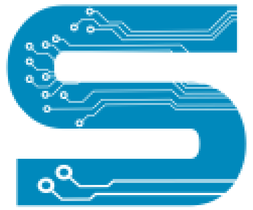


Syntacore™
Custom cores and tools

Compiler Optimizations Challenges for High-Performance RISC-V Cores

August 25, 2023

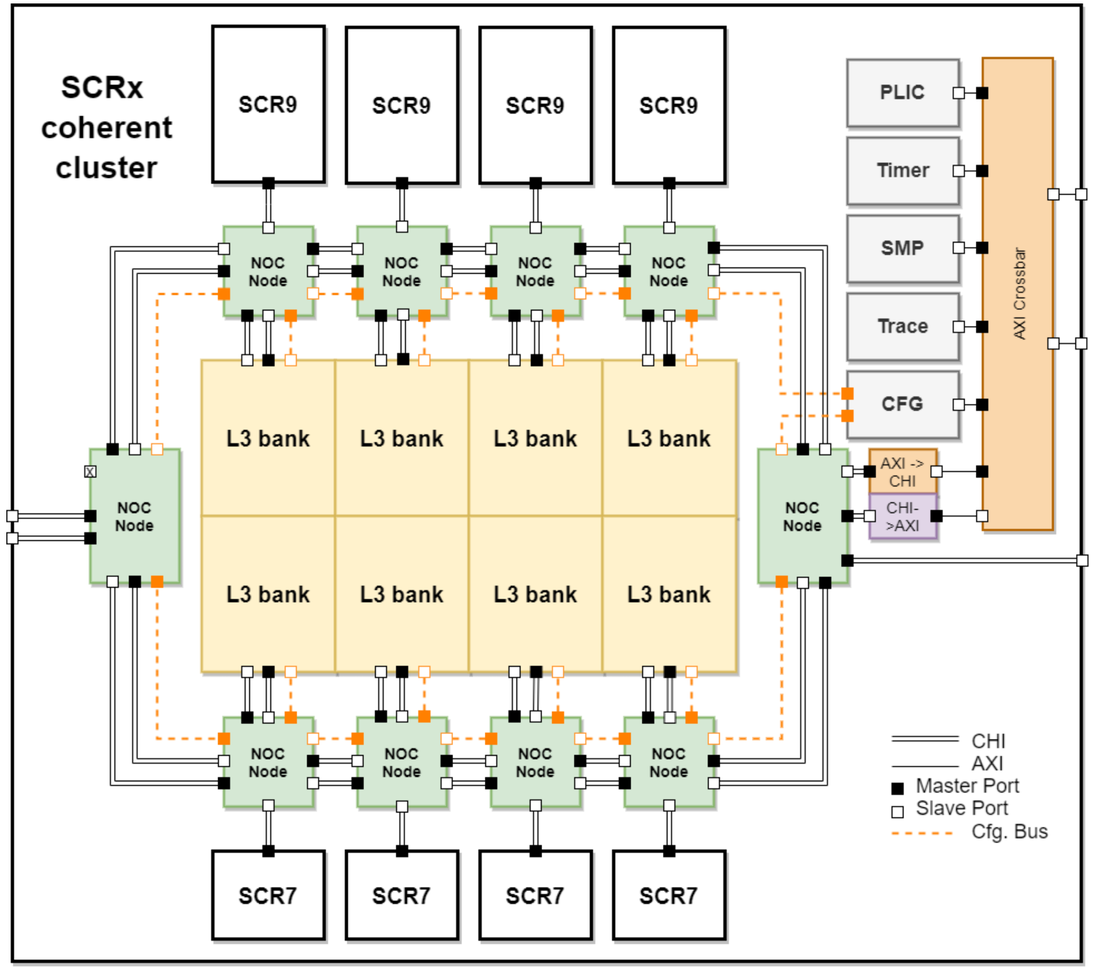
Presenter: Sergey Yakushkin
Contributors: Konstantin Vladimirov and Syntacore Compiler Team
info@syntacore.com

RISC-V

Example High-Performance Core - SCR9

Linux-capable application CPU with entry-level server class features:

- 8-16 cores per cluster (SMP and heterogeneous)
- Multi-issue OOO uArch
- Coherent NoC-based L3
- CHI external i/f
- SV39, SV48, SV57
- RVV
- Hypervisor
- AIA
- Accelerators support



Early access program*

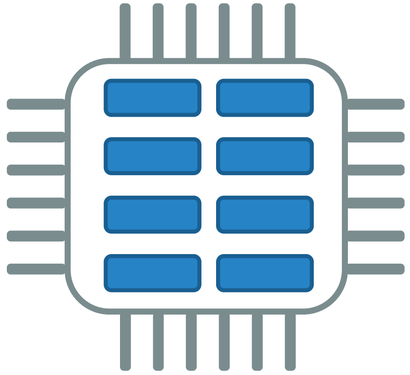
(*) some features may be not available in the initial release

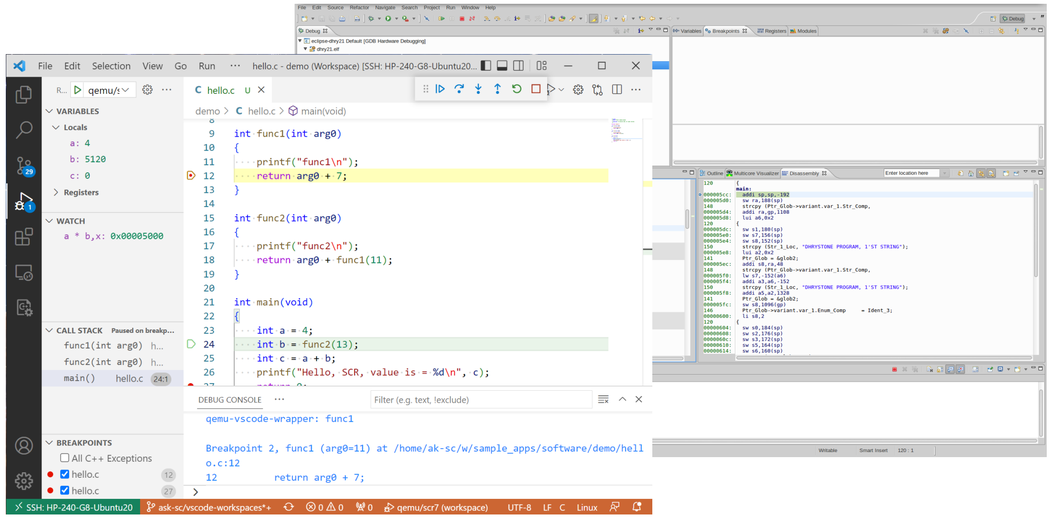
SCR7/9 FPGA-based DevKit

Fully-integrated system and DevKit based on

- VCU118 https://www.xilinx.com/products/boards-and-kits/vcu118.html or
- UltraScale+ VU19P http://www.hitechglobal.com/Boards/VirtexUltraScale+_VU19P_Board.htm

- Multi-core, up to 24GB RAM, up to 100-150 MHz, 1GB Ethernet, PCI/SSD storage
- Boots upstream Debian Linux kernel 5.15/6.1 LTS
- Integrated toolchain with IDE (supports Bare Metal and Linux targets)
- Extra SW including OpenJDK stable builds





Syntacore™
Custom cores and tools

RISC-V

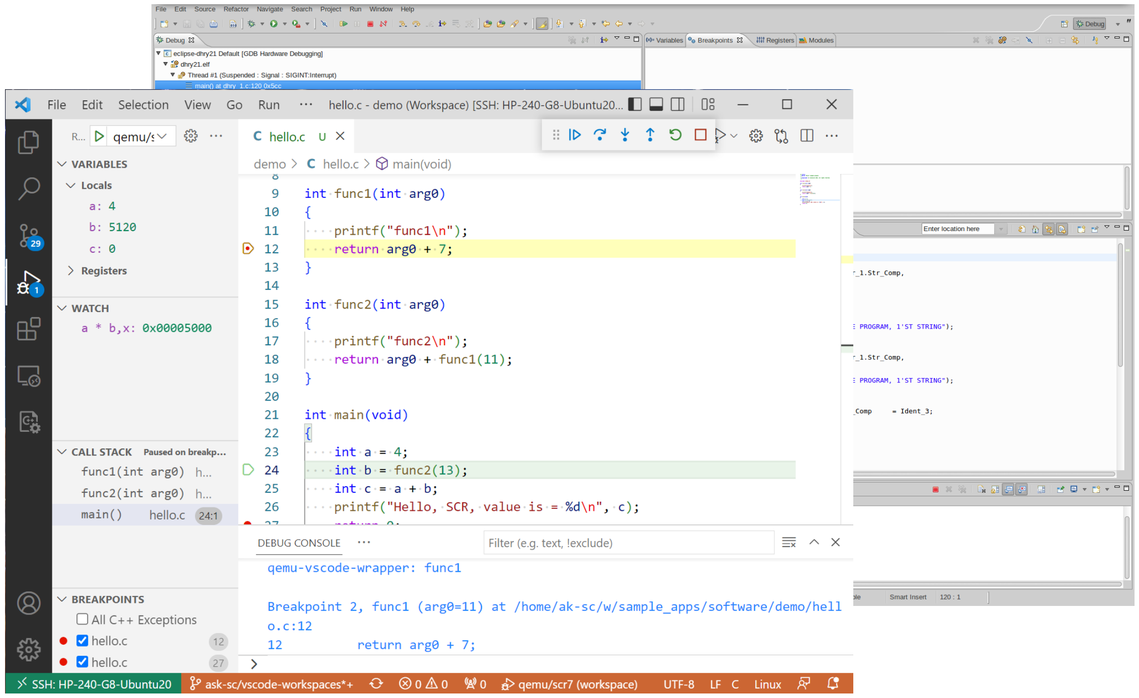
Syntacore Development Toolkit

SC-DT 2023.08 release:

- LLVM 16.x with optimizations
- GNU GDB 13.2
- Open On-Chip Debugger 0.12.x
- QEMU 8.0.3
- GCC 12.2.1
- GNU Binutils 2.38
- Newlib 4.10
- Visual Studio Code and Eclipse

Hosts

Bare Metal + Linux



Also available:

- COMPCERT
- Profiling tools
- OpenJDK

Simulators:

- QEMU
- Spike
- SAIL
- 3rd party vendors

JTAG-based debug solutions:

- Segger J-link
- Olimex ARM-USB-OCD family
- Digilent JTAG-HS2
- more vendors soon

https://syntacore.com/page/products/sw-tools

Syntacore™ Custom cores and tools

RISC-V

LLVM Compiler and Optimizations

Upstream LLVM for RISC-V in 2022

- Gaps vs AArch64/GCC,
  >10% diffs on some workloads
- ABI and conventions are evolving
- Functional issues, e.g. vectorization

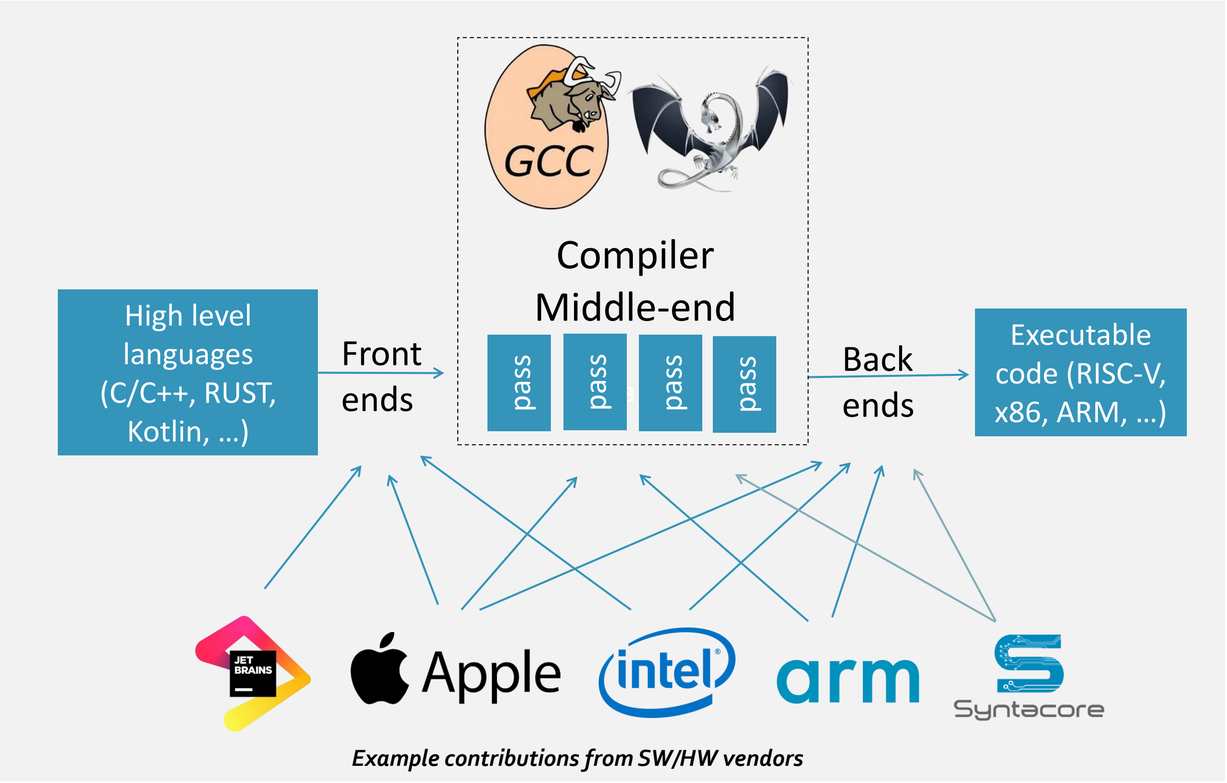
Syntacore LLVM Toolchain

- Improved stability
- SCR uArch-aware optimizations
- Advanced optimizations for RISC-V
- Enhanced RVV auto-vectorization

LLVM is open-source compiler framework:

20+ years, 500 000+ commits, 1500+ developers

primary for OS (Android), languages (Swift, Kotlin, RUST),
commercial tools (Intel C++ Compiler), and code analyzers (Coverity)



Example contributions from SW/HW vendors

Loop Optimizations: Unsigned Wrap

Compiler optimization improves codegen for 32-bit loop counters:

- Identify code fragments
- Add dynamic range checks
- Generate multiple specialized code versions for different ranges
- Apply further transformations for optimal code generation

Improves some industry benchmarks up to 18% (config without Bitmanip)

Patch submitted to LLVM
https://reviews.llvm.org/D132208

Source

```
for (unsigned i = 0; i < N; ++i)
  for (unsigned j = 0; j < N; ++j)
    C[i*N+j] = foo();
```

Extra 32/64 conversions inside loop

Dynamic range checks (new)

```
for (unsigned i = 0; i < N; ++i)
  for (unsigned j = 0; j < N; ++j)
    if (overflow(N*N))
      *(C + zext(i*N+j)) = foo();
    else
/* nuw */ *(C + zext(i*N+j)) = foo();
```

Unswitching

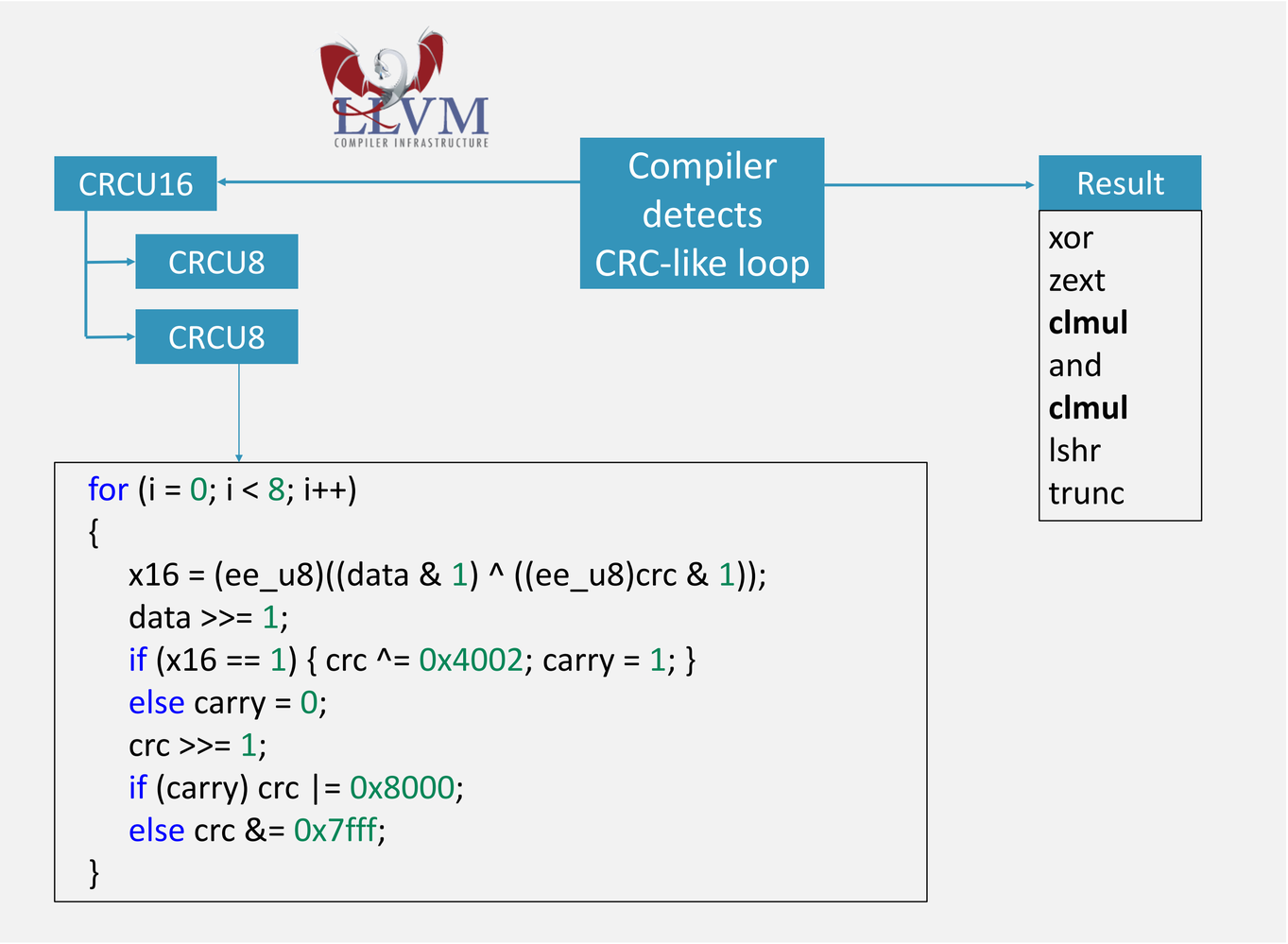
```
if (overflow(N*N))
  two loops with unsigned wrap
else
  for (unsigned i = 0; i < N; ++i)
    for (unsigned j = 0; j < N; ++j)
      *(C + zext(i*N+j /*nuw*/))=foo();
```

IndVar simplify

```
if (overflow(N*N))
  two loops with unsigned wrap
else
  for (uint64_t i = 0; i < N; ++i)
    for (uint64_t j = 0; j < N; ++j)
      *(C + (i*N+j)) = foo();
```

Loop Optimizations: CRC-like Patterns

- Compiler detects loop patterns with generalized CRC computations
- Transforms code and utilize Bitmanip instruction CLMUL
- Improves function by 10x, and some industry benchmarks up to 15%



Global Array Access Optimization

Relaxation: linker attempts to find shorter instructions

- Example:

```
char arr[42];
char foo(int i) { return arr[i];}
```

- Assembler:

```
lui a5, %hi(arr)      # R_RISCV_HI20, R_RISCV_RELAX
addi a5, a5, %lo(arr) # R_RISCV_LO12_I, R_RISCV_RELAX
add  a5, a5, a0
lbu a0, 0(a5)
ret
```

- Optimization attempt (RISCVMergeBaseOffset):

```
lui a5, %hi(arr)      # R_RISCV_HI20, R_RISCV_RELAX
~~addi a5, a5, %lo(arr)~~
add  a5, a5, a0
lbu a0, %lo(arr)(a5)  # R_RISCV_LO12_I, R_RISCV_RELAX
ret
```

- Original GP-relaxation breaks code in this example

```
lui a5, %hi(arr)      # R_RISCV_HI20, R_RISCV_RELAX
add  a5, a5, a0
lbu a0, %lo(arr)(a5)  # R_RISCV_LO12_I, R_RISCV_RELAX
ret
```

- Wrong (original):

```
~~lui a5, %hi(arr)~~
add  a5, a5, a0
lbu a0, offset(gp)
ret
```

- Correct (expected):

```
~~lui a5, %hi(arr)~~
add  a5, gp, a0
lbu a0, offset(a5)
ret
```

New GPREL_* relocations

- Three new relocation types are supported in Syntacore LLVM toolchain:
  - R_RISCV_GPREL_ADD
  - R_RISCV_GPREL_LO12_I
  - R_RISCV_GPREL_LO12_S

- Generated code example (3 instructions instead of 4):

```
lui a5, %hi(arr)                          # R_RISCV_HI20, R_RISCV_RELAX
add  a5, a5, a0, %gprel_add(arr)          # R_RISCV_GPREL_ADD, R_RISCV_RELAX
lbu a0, %gprel_lo(arr)(a5)                # R_RISCV_GPREL_LO12_I, R_RISCV_RELAX
ret
```

- Similar proposal to fold a non-constant offset into an lui/addi/lw sequence in RISC-V GNU toolchain: link

- Example impact: ~3% for 445.gobmk SPEC2k

More Compiler Optimizations

- Partial Redundancy Elimination for Loads: 11 patches (D141664 … D143255)
  - ~3% on SPEC2k6 471.omnetpp
- Fold terminating condition for any icmp (-lsr-term-fold D145929)
  - ~0.5% overall on SPEC2k6
- RVV Fixes and Improvements
  - Fixed unwinding for RVV spills using DWARF CFA (D136263 D136264)
  - Proposed further optimizations for tail-agnostic policy VMV/VFMV (D130895)

Summary

- Syntacore LLVM is close/better than GCC, many patches submitted upstream
  - 10%+ on some SPEC benchmarks
  - Fixed functional bugs, e.g. RVV spill/unwinding
- Areas for further compiler improvements vs AArch64/x86 and GCC
  - Enhancement of generic LLVM passes, MachineCombiner, Peephole, new ISA extensions
  - RVV generic and TA-specific optimizations
  - Loop optimizations and recognition of patterns
- Opportunities for new link-time optimizations with GPREL_* relocations

Thank you!

謝謝

info@syntacore.com