Compliance, Verification and Customization of RISC-V Cores and SoCs

Getting Started with RISC-V – North America Roadshow
Waltham (Boston) – Austin – Irvine – Milpitas (Bay Area) : April 2019

Imperas Focus

- “nobody designs a chip without simulation”,
  at Imperas we believe that:
  “nobody should develop embedded software without simulation”

- Imperas develops simulators, tools, debuggers, modelling technology, and models to help embedded systems developers get their software running…
  - and hardware developers get their designs correct

- 10+ years, self funded, profitable, UK based, team with much EDA (simulators, verification), processors, and embedded experience
- www.imperas.com/riscv

Agenda

- Compliance and verification
- Customization
- Summary

Agenda

- Compliance and verification
  - Why is compliance important for RISC-V?
  - RISC-V compliance checking
  - Verification
- Customization
- Summary

Compliance for RISC-V is Important

Q: What is meant by “compliance”?

A: The device works within the envelope of the agreed specifications

Q: Is there an easy process or path to follow to develop methodologies/tools for compliance?

A: NO – all established ISAs are single company controlled and those companies work extremely hard on proprietary solutions to ensure that all designs that go out their door work correctly – so RISC-V has to pioneer compliance collectively

Q: What happens if the RISC-V industry builds devices that are not complying with specifications?

A: Users cannot assume that tools like C compilers, operating systems, and application software will be transferable across devices and work correctly

The RISC-V Foundation's Compliance Task Group

- Jun 2017 group started
  - Compliance testing is a testing technique which is done to validate whether the system developed meets the prescribed standards or not. It is **not design verification** testing
    - compliance testing is looking for issues like missing registers, modes, instructions – not for bugs in RTL implementations...
  - The tests have to be written to ensure compliance/non-compliance is **observable in a test signature**. The signatures are published so that the user does not have to run a reference model and can compare the results of their target runs to the reference signature
- Jan 2018 initial rv32i test suite provided by Codasip
- Jun 2018 Imperas, Embecosm worked on GitHub, made repo public
- Oct 2018 Imperas improved test coverage, added new suites, ported 32bit riscv-tests

RISC-V Compliance Suite

- It is ‘work in progress’
- Two components
  - Test suites
    - Each suite focuses on a feature set of the RISC-V envelope
    - Initial focus is instructions, user mode spec, e.g. rv32i, rv32im, rv32imc, rv64i, …
    - Awaiting RISC-V platform specifications to subset privilege spec, before starting privilege suites
  - Framework
    - Make, bash, and scripts
      - Encapsulate compiler tools, linkers, simulators, and targets (Devices Under Test)
    - Includes simulator: as example target, and to generate reference signatures
    - Run: Select suite and target
      - Runs each test, target produces signatures, compares to saved golden reference signature
- Available: www.github.com/riscv/riscv-compliance

Compliance Suite Status (rv32i)

Instruction	Decode Coverage
lb	100%
lh	100%
lw	100%
lbu	100%
lhu	100%
sb	100%
sh	100%
sw	100%
sll	100%
slli	100%
srl	100%
srli	100%
sra	100%
srai	100%
add	100%
addi	100%
sub	100%
lui	100%
auipc	100%
xor	100%
xori	100%
or	100%
ori	100%
and	100%
andi	100%
slt	100%
sltu	100%
j	100%
jr	100%
mv	100%

beq	95%
bne	95%
blt	95%
bge	95%
bltu	95%
bgeu	95%
slti	77%
sltiu	77%
fence	0%
fence.i	0%
scall	0%
sbreak	0%
rdcycle	0%
rdcycleh	0%
rdtime	0%
rdtimeh	0%
rdinstret	0%
rdinstreth	0%

Need more tests

Investigate (more tests?)

no tests yet
(fence.i not in 2.3 RV32I)

pseudos+CSR, no tests yet
(not in 2.3 RV32I)

Notes on rv32i test suite:

- 54 hand coded directed tests (average 150 instructions each)
  - https://github.com/riscv/riscv-compliance/tree/master/riscv-test-suite/rv32i/src

Notes on Decode Coverage

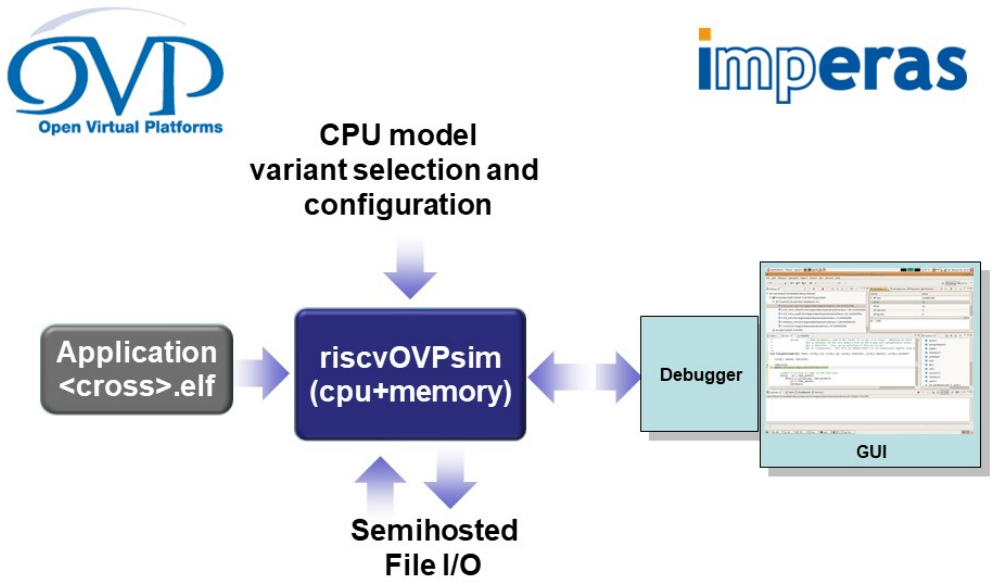
- Decode Coverage: observe changes on all bits of legal decodes
- Decode Coverage data from the Imperas Fault Simulation Coverage tool
  - ran 478,390 simulations in 308 secs

Similar status available on RV32IM, RV32IMC, RV64I, RV64IM suites

Coverage Metrics for Compliance Tests

- Coverage metrics used in RTL design verification are ***not applicable*** as they are often functional and connected to specific microarchitecture (RTL)
- Imperas' Code Coverage Tool provides coverage of model source, which is useful to see how much of model is exercised by tests but ***does not*** show how much of specification is covered
- Imperas' Fault Simulation Coverage tool provides ***Instruction Decode Coverage***
  - explores the decodes of the instructions and mutates legal bits
  - detects that there is a test that stimulates & observes each bit
  - tool is very fast, runs in parallel, provides other metrics including data coverage
  - very useful for users as ***provides coverage of custom instructions***

riscvOVPsim simulator



Imperas riscvOVPsim Compliance Simulator

- Instruction Accurate simulator using high performance Just-in-Time Code Morphing that executes RISC-V binaries and runs on Linux/Windows PC
  - Runs very fast, 1,000 MIPS
- Industrial quality for use in test development, software development, compliance testing and design verification
  - Includes capabilities to perform RISC-V compliance signature generation
- Maintained and supported by Imperas Software (www.imperas.com)
- Simulator restricted to single processor model and fully populated memory
- Includes Apache 2.0 Open Source model of complete RISC-V ISA envelope model of 2.2, 2.3, 1.10, 1.11 revisions of the RISC-V Foundation specifications (not hypervisor/vectors – not stable)
- ***FREE. No license keys or license management***
- Video: http://www.imperas.com/riscvovpsim-a-complete-risc-v-iss-for-bare-metal-software-development-and-specification-compliance

Issues to be sorted by RISC-V Foundation's Compliance Working Group

- Detailed specifications needed
  - the structure of the Compliance Suite Framework (and how used)
  - what content of test should be (how written)
  - list of all the tests needed in each test suite (what is covered of specification)
- Formal process for RISC-V Foundation to allow user to claim their processor is a RISC-V processor
  - i.e. process to explore and record 'is it a RISC-V compliant design?'
  - the 'rubber stamp: RISC-V inside'

Imperas: Exploring designs

Note: these notes are Imperas exploring designs brought into Imperas, not a discussion of what our customers are doing with our tools & test suites

- Many candidate DUTs (Device-Under-Test) = customers, partners, & for fun…
  - proprietary RTL, open source RTL
  - FPGA
  - Silicon
  - Simulators
- Process – goal is to load .elf file, run, write signature, compare with golden reference
- Develop encapsulation of DUT
  - so can run public GitHub Compliance Suite – and – Imperas internal compliance test suites
- Different levels of complexity and challenge
  - ISA Simulators relatively easy (as can read/write files)
  - RTL simulators a little harder, 2 approaches
    - Memory read/write
    - GDBserver – control as if hardware, inject, extract data
  - Hardware (FPGA, silicon)
    - Pod, JTAG, GDBserver type approaches to get data in/out

What did we find…

- Many candidate DUTs (Device-Under-Test) = customers, partners, & for fun…
  - proprietary RTL, open source RTL
  - FPGA
  - Silicon
  - Simulators
- Missing registers
- Missing instructions
- Floating point mode change issues
- PMP implementation issues
- …

Verification

- Two key issues in verification of SoCs based on open ISA processors:

1. Verification of the processor
2. Verification of the interface between the processor and the NoC/system bus

- Needed:
  - Reference model of the processor
  - Test generation
  - Processor instruction coverage measurement

Agenda

- Compliance and verification
- Customization
  - Why customize the ISA?
  - How to add custom features
- Summary

There are and will be many different RISC-V CPU developers:

- Deliver RTL IP as a business (like Arm, MIPS, just for RISC-V)
  - e.g. Andes, CloudBear, Codasip, Incore, Syntacore, …
- Develop internal cores for SoCs
  - e.g. Nvidia, …
- Develop for internal use but make available as open source
  - e.g. Western Digital, …
- Develop and use open source as a business opportunity
  - e.g. SiFive, …
- Download and use open source RTL in their products
  - e.g. Greenwaves, Google, …
- …

RISC-V CPU Adopters: (above & below the line)

New architectures (ML, IoT), arrays of processors, custom features, high performance, feature control (efficiency), large SoC design, architecture innovation, …

Key requirement: ‘freedom to innovate’

Researchers, education, open source community, freely available, no license cost or restriction, …

Key requirement: ‘free’

Many (above the line) adopters of RISC-V want to add their own custom extension instructions

- Traditional ISA choice has been hard if you want to add your own custom processor instructions to an ISA
- RISC-V as an open standard has specific regions of instruction decode space specifically allocated for users to add their own instructions

- Challenges
  - How to choose the processor IP starting point
  - How to add instructions to processor RTL
  - How to verify the complete RTL
  - How to evaluate effectiveness and performance gains of new instructions
  - How to enable software development utilizing the new instructions

Key Challenge of Optimizing Custom Instructions Requires New Methodology, Tools

- How to choose the processor IP starting point
- How to add instructions to processor RTL
- How to verify the complete RTL
- **How to evaluate effectiveness and performance gains of new instructions**
- How to enable software development utilizing the new instructions

➢ **Extend instruction accurate simulation tools and models to support analysis and optimization of custom instructions**

Agenda – Custom Instructions

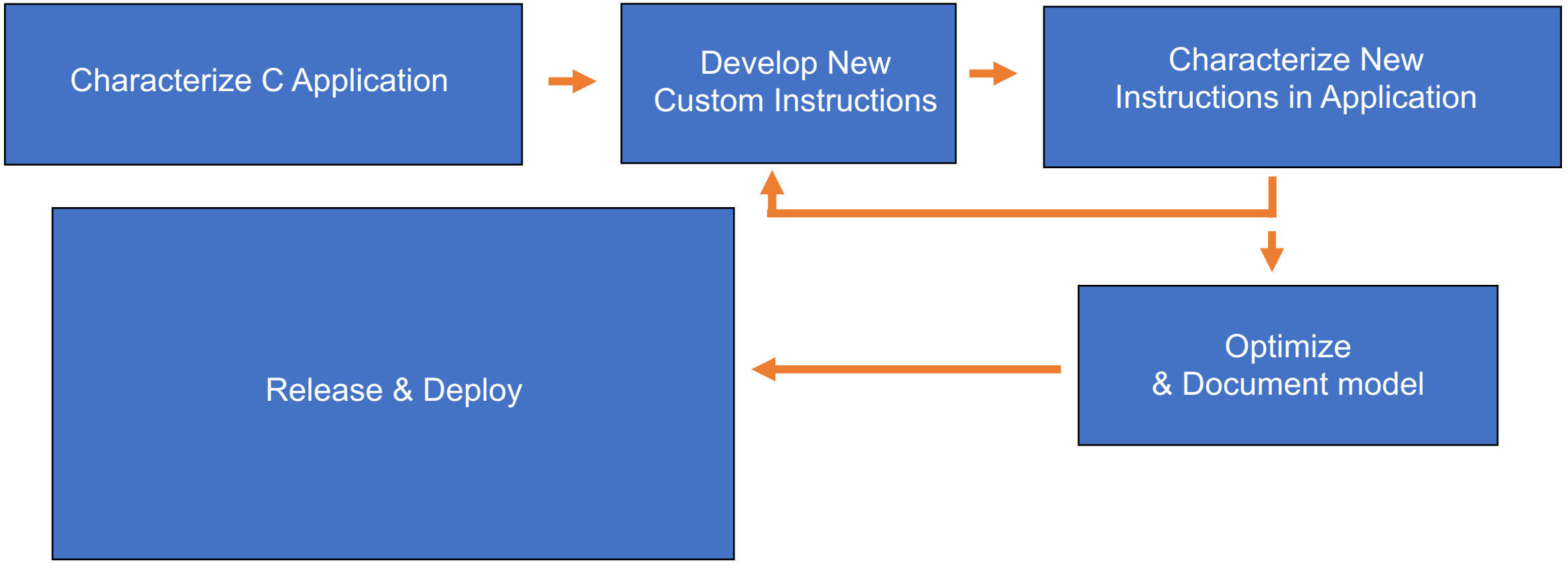
- Who is adopting RISC-V and why…
- Adding custom instructions to RISC-V processors
- Custom instruction optimization flow

Custom Instruction Optimization Flow

- Show the flow used when designing new instructions to improve performance of applications running on a RISC-V processor
  - Allows evaluation and firming up of the instructions
  - Gets to the specification of the instructions needed to be implemented in RTL
- Introduce the technologies & tools needed for each stage

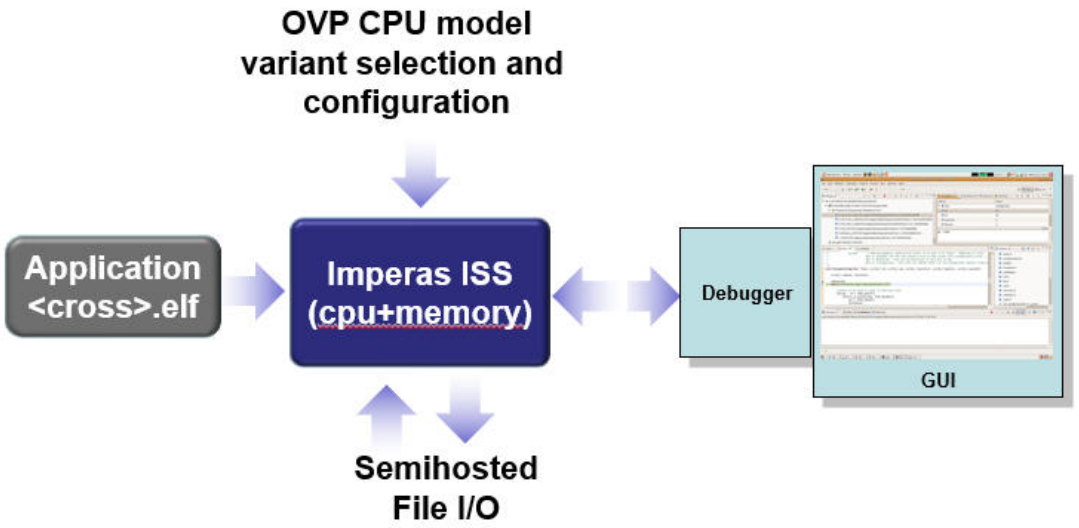
- Application software used for this walk-through is a character stream encoder, based on ChaCha20 encryption algorithm
  - Instruction Extensions to RISC-V courtesy of Cerberus Security Laboratories Ltd
  - https://cerberus-laboratories.com

Flow to add new custom instructions

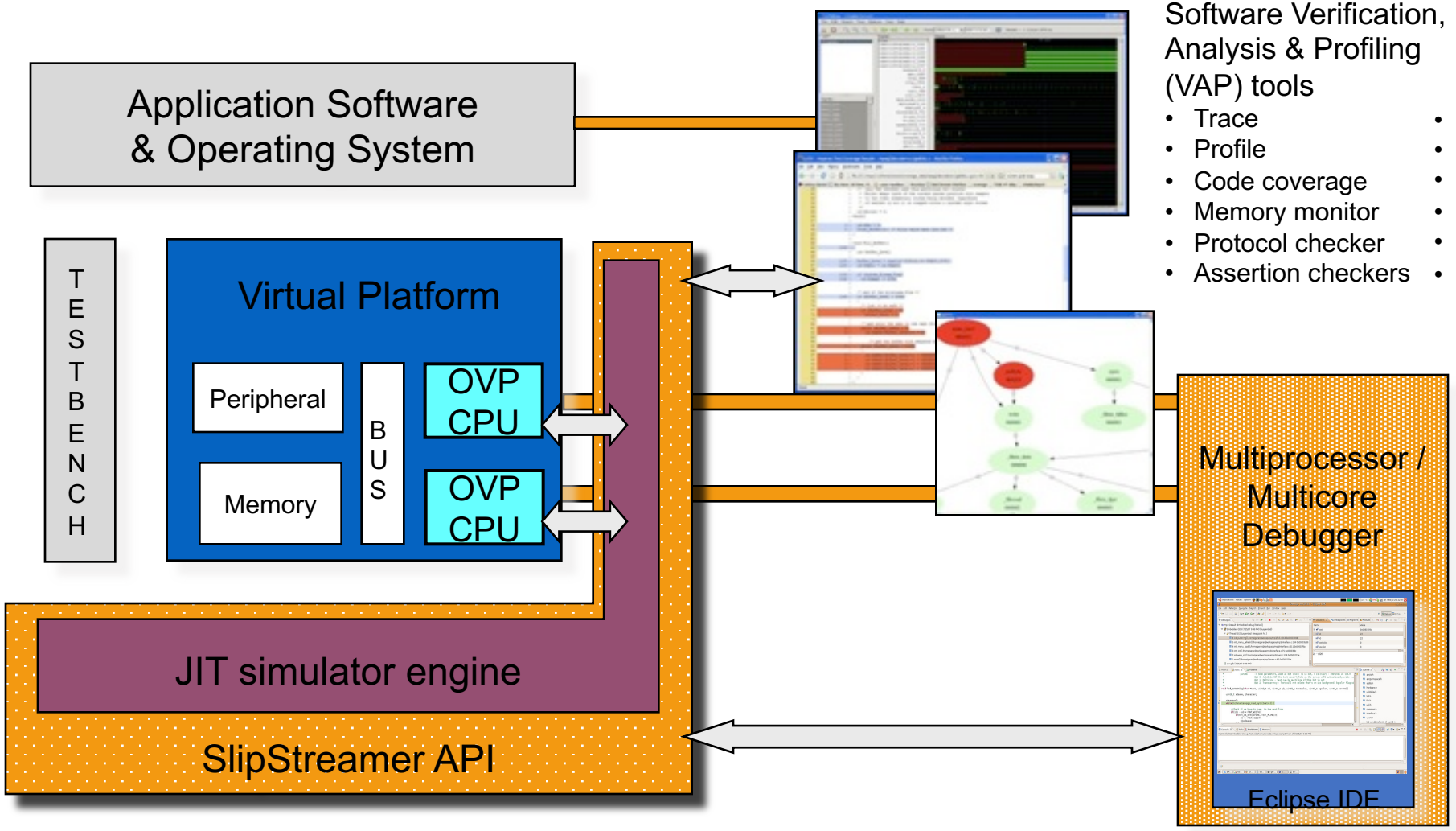


The first thing needed is a simulator – think ISS, however …

- Simulator and model need to have ability to extend to allow custom instructions and new tools
- Ease of use also important
- Need to have a business-friendly open source license



Imperas Environment for Embedded Software Development, Debug & Test



Software Verification, Analysis & Profiling (VAP) tools

- Trace
- Profile
- Code coverage
- Memory monitor
- Protocol checker
- Assertion checkers
- OS task tracing
- OS scheduler analysis
- Fault injection
- Function tracing
- Variable tracing
- …

Flow to add new custom instructions

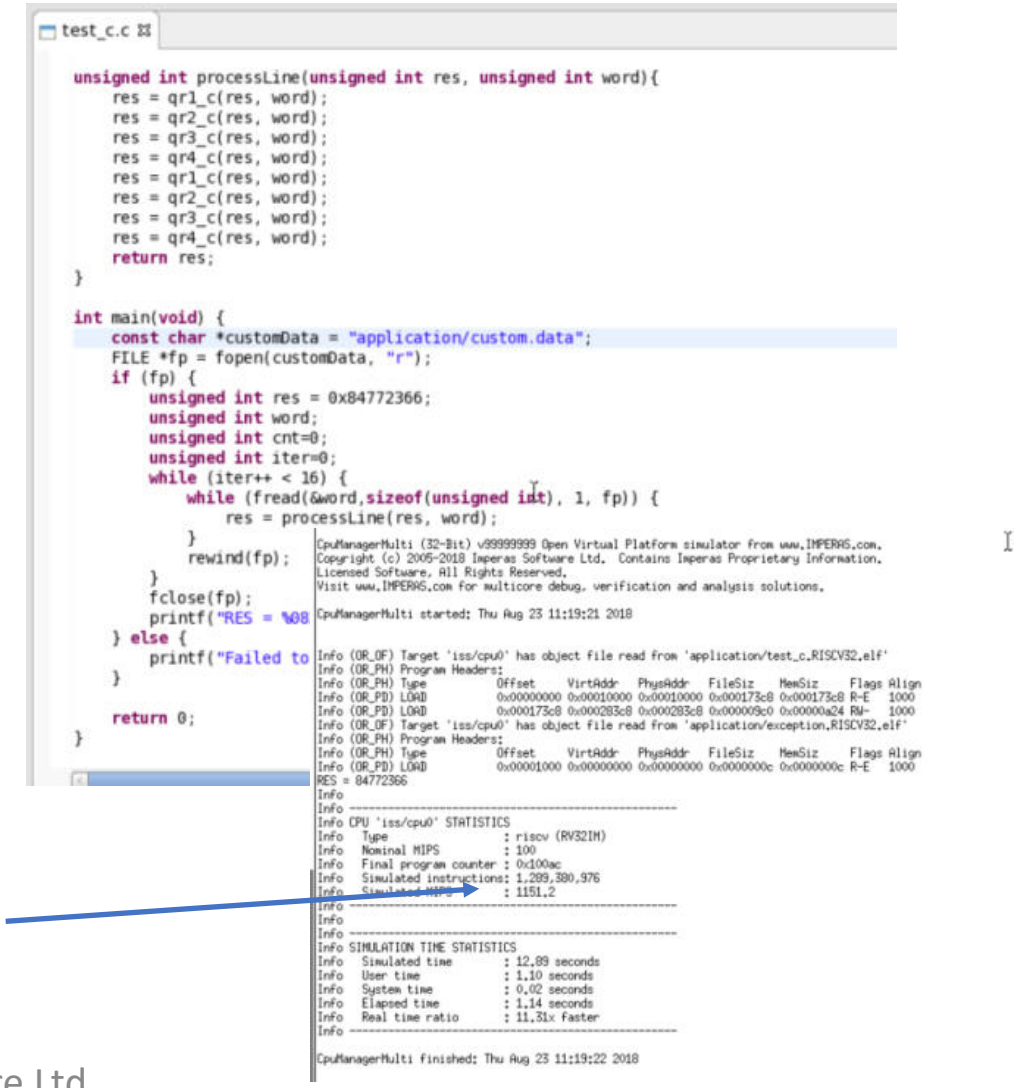
Characterize C Application

- Instruction Accurate Simulation
- Trace / Debug
- Timing Simulation
- Function Timing / Profiling

Instruction Accurate simulation of C application

- Cross compiled C application targeting RV32IM
  - Character stream encoder, with ChaCha20 encryption algorithm
- IA simulation
  - Imperas RISC-V ISS with configurable model of RISC-V specification selecting RV32IM
- Semihosting
  - Enables bare metal application to very simply access host I/O

➢runs fast
- Over 1 billion instructions a second (standard PC)
  - Linux and Windows supported host OS



Cycle Approximate simulation C application

- Same C application
- IA simulation + timing Estimation (IA+E)
  - Includes annotated timing estimation for RV32IM processor
- Same simulation data results, different timing as now counting cycles
- Shows how long algorithm will take to execute

➢Extends simulated time
  - Was 12.89 secs now takes 16.59 secs

```
CpuManagerMulti (32-Bit) v99999999 Open Virtual Platform simulator from www.IMPERAS.com.
Copyright (c) 2005-2018 Imperas Software Ltd.  Contains Imperas Proprietary Information.
Licensed Software, All Rights Reserved.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

CpuManagerMulti started: Thu Aug 23 11:27:16 2018


Info (OR_OF) Target 'iss/cpu0' has object file read from 'application/test_c.RISCV32.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type           Offset     VirtAddr   PhysAddr   FileSiz    MemSiz     Flags Align
Info (OR_PD) LOAD           0x00000000 0x00010000 0x00010000 0x000173c8 0x000173c8 R-E   1000
Info (OR_PD) LOAD           0x000173c8 0x000283c8 0x000283c8 0x000009c0 0x00000a24 RW-   1000
Info (OR_OF) Target 'iss/cpu0' has object file read from 'application/exception.RISCV32.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type           Offset     VirtAddr   PhysAddr   FileSiz    MemSiz     Flags Align
Info (OR_PD) LOAD           0x00001000 0x00000000 0x00000000 0x0000000c 0x0000000c R-E   1000
Info (CMD_CC) calling 'iss/cpu0/exTT/cpucycles'
Info (CPUEST_CMD) iss/cpu0: cpucycles on: time stretch enabled
RES = 84772366
Info
Info ------------------------------------------------------
Info CPU 'iss/cpu0' STATISTICS
Info   Type                 : riscv (RV32IM)
Info   Nominal MIPS         : 100
Info   Final program counter : 0x100ac
Info   Simulated instructions: 1,289,380,976
Info   Simulated MIPS       : 156.3
Info ------------------------------------------------------
Info
Info ------------------------------------------------------
Info SIMULATION TIME STATISTICS
Info   Simulated time       : 16.59 seconds
Info   User time            : 8.02 seconds
Info   System time          : 0.23 seconds
Info   Elapsed time         : 8.58 seconds
Info   Real time ratio      : 1.93x faster
Info ------------------------------------------------------

CpuManagerMulti finished: Thu Aug 23 11:27:25 2018


CpuManagerMulti (32-Bit) v99999999 Open Virtual Platform simulator from www.IMPERAS.com.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

Info (CPUEST_RSLT) Estimated execution time 16.59 seconds, clock cycles 1,659,204,454
Use script lastRun.sh to rerun with current settings
```

Function Profile C Application

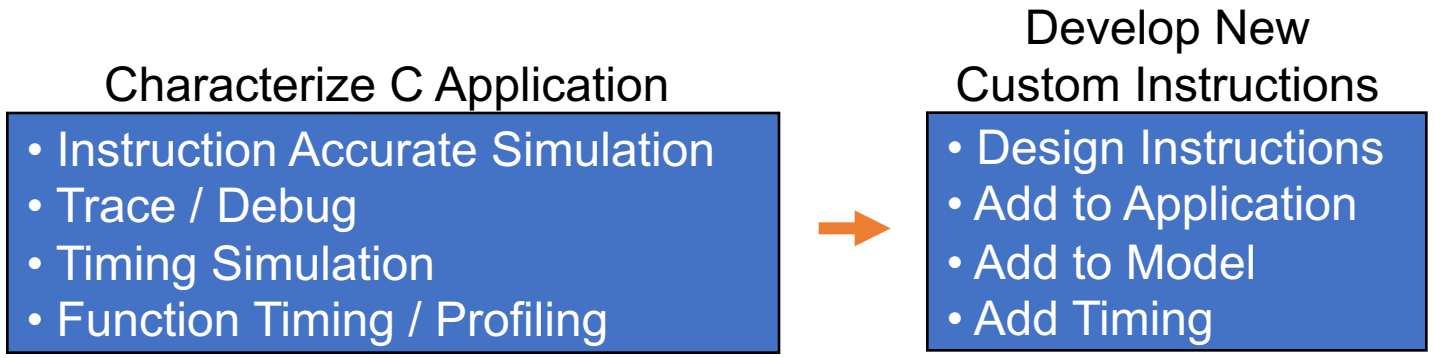
- Same C application
- IA+E simulation
- Sampled profiling with call stack analysis

➢Shows proportion of time spent in each application function

  ➢21.35% spent in processLine

Name (location)	Arcs in	Samples in	Arcs out	From/To this	Percentage (%)
Platform: iss					
Processor: iss/cpu0					
Process: 0_None		1659204277			
_fread_r	598189652	596534872	1654780		35.95%
processLine	925852930	354236017	571616913		21.35%
qr4_c	150627639	150627639	0		9.08%
qr1_c	146083640	146083640	0		8.8%
qr2_c	137682652	137682652	0		8.3%
qr3_c	137222982	137222982	0		8.27%
__libc_init_array	0	135154865	1524049412		8.15%
__srefill_r	1654780	1024985	629795		0.06%
__sread	629637	321116	308521		0.02%
_read_r	308521	308521	0		0.02%
_fseeko_r	2706	2126	580		0.0%
_vfprintf_r	1874	764	1110		0.0%
__sfvwrite_r	848	752	96		0.0%
rewind	3267	561	2706		0.0%
_close_r	357	357	0		0.0%
_malloc_r	323	297	26		0.0%
__sseek	528	288	240		0.0%
_lseek_r	240	240	0		0.0%
__sfmoreglue	399	224	175		0.0%
_fclose_r	734	204	530		0.0%

Flow to add new custom instructions



Add custom instructions to application

- Inline assembly using new instructions replacing C code
- 4 new instructions
- Cross Compile using standard tool
- Run on IA simulator

➢ Unimplemented instruction exception
  - As the instructions have not yet been added to the simulator model

```
// Custom instruction test for Chacha20
#include <stdio.h>

unsigned int processLine(unsigned int input, unsigned int word){
    unsigned int res = input;
    asm __volatile__("mv x10, %0" :: "r"(res));
    asm __volatile__("mv x11, %0" :: "r"(word));
    asm __volatile__(".word 0x00B5050B\n" ::: "x10");      // QR1
    asm __volatile__(".word 0x00B5150B\n" ::: "x10");      // QR2
    asm __volatile__(".word 0x00B5250B\n" ::: "x10");      // QR3
    asm __volatile__(".word 0x00B5350B\n" ::: "x10");      // QR4
    asm __volatile__(".word 0x00B5050B\n" ::: "x10");      // QR1
    asm __volatile__(".word 0x00B5150B\n" ::: "x10");      // QR2
    asm __volatile__(".word 0x00B5250B\n" ::: "x10");      // QR3
    asm __volatile__(".word 0x00B5350B\n" ::: "x10");      // QR4
    asm __volatile__("mv %0,x10" : "=r"(res));
    return res;
}

int mai
    con
    FIL
    if
```

```
CpuManagerMulti (32-Bit) v99999999 Open Virtual Platform simulator from www.IMPERAS.com.
Copyright (c) 2005-2018 Imperas Software Ltd.  Contains Imperas Proprietary Information.
Licensed Software, All Rights Reserved.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

CpuManagerMulti started: Thu Aug 23 11:34:51 2018

Info (OR_OF) Target 'iss/cpu0' has object file read from 'application/test_custom.RISCV32.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type           Offset     VirtAddr   PhysAddr   FileSiz    MemSiz     Flags Align
Info (OR_PD) LOAD           0x00000000 0x00010000 0x00010000 0x00017270 0x00017270 R-E   1000
Info (OR_PD) LOAD           0x00017270 0x00028270 0x00028270 0x000009c0 0x00000a24 RW-   1000
Info (OR_OF) Target 'iss/cpu0' has object file read from 'application/exception.RISCV32.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type           Offset     VirtAddr   PhysAddr   FileSiz    MemSiz     Flags Align
Info (OR_PD) LOAD           0x00001000 0x00000000 0x00000000 0x0000000c 0x0000000c R-E   1000
Warning (RISCV_ABF) CPU 'iss/cpu0' 0x00010248 00b5050b custom1: Illegal instruction - extension X (non-standard extensions present) absent or inactive
Info
Info ----------------------------------------------------
Info CPU 'iss/cpu0' STATISTICS
Info   Type                : riscv (RV32IM)
Info   Nominal MIPS        : 100
Info   Final program counter : 0x102e4
Info   Simulated instructions: 1,340
Info   Simulated MIPS      : run too short for meaningful result
Info ----------------------------------------------------
Info
Info ----------------------------------------------------
Info SIMULATION TIME STATISTICS
Info   Simulated time      : 0.00 seconds
Info   User time           : 0.01 seconds
Info   System time         : 0.00 seconds
Info   Elapsed time        : 0.01 seconds
Info ----------------------------------------------------

CpuManagerMulti finished: Thu Aug 23 11:34:51 2018
```

Add custom instructions to model

- Use standard Open Virtual Platforms (OVP) instruction modeling APIs to add new instructions (and optional state) as ***new extension library***
  - Easy to extend decode table, add efficient behavioral JIT code
  - Optionally can call directly into user's provided C function of behavior
- Compile and link model extension library
- Simulate IA with ISS plus standard model extended with new library

➢ Instruction count and simulated time have reduced (IA)

```
//
// Create the RISCV decode table
//
static vmidDecodeTableP createDecodeTable(void) {

    vmidDecodeTableP table = vmidNewDecodeTable(RISCV_INSTR_BITS, RISCV_EIT_LAST);

    // R-Type instruction in custom-0 encoding space:
    // opcode [6:0]  = 00 010 11
    // funct3[14:12] = 0,1,2,3 (QR1-4)
    // funct7[31:25] = 0000000
    // rs1[19:15]
    // rs2[24:20]
    // rd[11:7]

    // handle custom instruction
    DECODE_ENTRY(0, CHACHA20QR1, "|0000000.........000.....0001011|");
    DECODE_ENTRY(0, CHACHA20QR2, "|0000000.........001.....0001011|");
    DECODE_ENTRY(0, CHACHA20QR3, "|0000000.........010.....0001011|");
    DECODE_ENTRY(0, CHACHA20QR4, "|0000000.........011.....0001011|");

    return table;
}
//
```

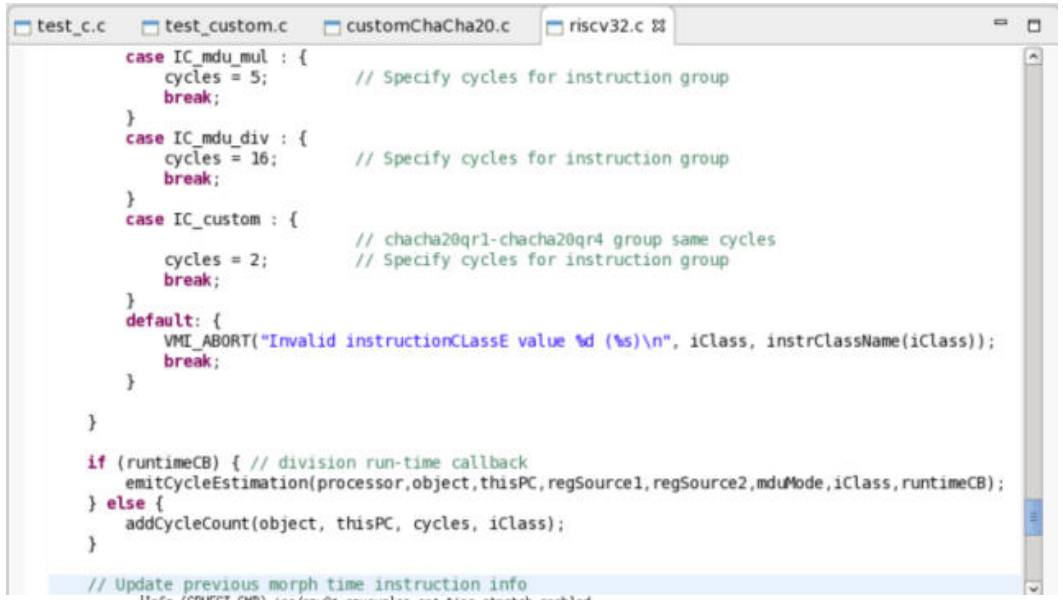
```
//
// Emit code implementing exchange instruction
//
static void emitChaCha20(
    vmiProcessorP processor,
    vmiosObjectP  object,
    Uns32         instruction,
    Uns32         rotl
) {

    // extract instruction fields
    Uns32 rd  = RD(instruction);
    Uns32 rs1 = RS1(instruction);
    Uns32 rs2 = RS2(instruction);

    vmiReg reg_rs1 = vmimtGetExtReg(processor,  &object->rs1);
    vmiReg reg_rs2 = vmimtGetExtReg(processor,  &object->rs2);
    vmiReg reg_tmp = vmimtGetExtTemp(processor, &object->tmp);

    vmimtGetR(processor, RISCV_REG_BITS, reg_rs1, object->riscvRegs[rs1]);
    vmimtGetR(processor, RISCV_REG_BITS, reg_rs2, object->riscvRegs[rs2]);
    vmimtBinopRRR(32, vmi_XOR, reg_tmp, reg_rs1, reg_rs2, 0);
    vmimtBinopRC(32, vmi_ROL, reg_tmp, rotl, 0);

    vmimtSetR(processor, RISCV_REG_BITS, object->riscvRegs[rd], reg_tmp);
}
```

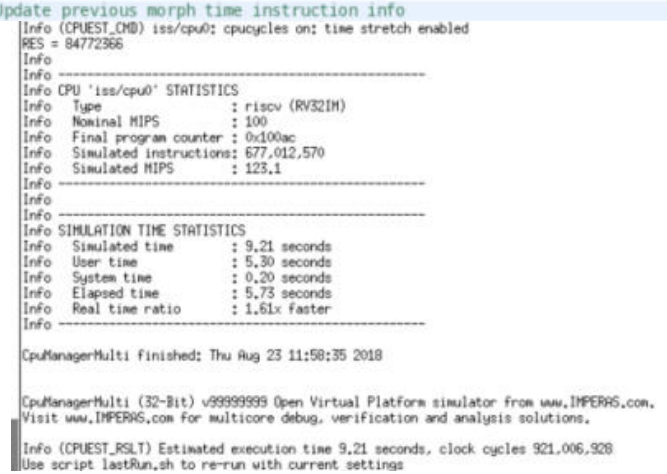
```
CpuManagerMulti (32-Bit) v99999999 Open Virtual Platform simulator from www.IMPERAS.com.
Copyright (c) 2005-2018 Imperas Software Ltd.  Contains Imperas Proprietary Information.
Licensed Software, All Rights Reserved.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

CpuManagerMulti started: Thu Aug 23 11:41:32 2018

Info (OP_LPR) Processor  iss/cpu0                              $IMPERAS_VLNV/riscv.ovpworld.or
Info (OR_DF) Target 'iss/cpu0' has object file read from 'application/test_custom.RISCV32.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type           Offset     VirtAddr   PhysAddr   FileSiz    MemSiz     Flags Align
Info (OR_PD) LOAD           0x00000000 0x00010000 0x00010000 0x00017270 0x00017270 R-E   1000
Info (OR_PD) LOAD           0x00017270 0x00028270 0x00028270 0x000009c0 0x00000a24 RW-   1000
Info (OR_DF) Target 'iss/cpu0' has object file read from 'application/exception.RISCV32.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type           Offset     VirtAddr   PhysAddr   FileSiz    MemSiz     Flags Align
Info (OR_PD) LOAD           0x00001000 0x00000000 0x00000000 0x0000000c 0x0000000c R-E   1000
Info (OP_PEX) Extension  iss/cpu0/riscv32Newlib                $IMPERAS_VLNV/riscv.ovpworld.org/semihosting/riscv32Newlib/1.0/model
Info (OP_PEX) Extension  iss/cpu0/exInst                       instructionExtensionLib
RES = 84772366
Info
Info ----------------------------------------------------
Info CPU 'iss/cpu0' STATISTICS
Info   Type                 : riscv (RV32IM)
Info   Nominal MIPS         : 100
Info   Final program counter : 0x100ac
Info   Simulated instructions: 677,012,570
Info   Simulated MIPS       : 1301.9
Info ----------------------------------------------------
Info
Info ----------------------------------------------------
Info SIMULATION TIME STATISTICS
Info   Simulated time       : 6.77 seconds
Info   User time            : 0.50 seconds
Info   System time          : 0.02 seconds
Info   Elapsed time         : 0.53 seconds
Info   Real time ratio      : 12.81x faster
Info ----------------------------------------------------

CpuManagerMulti finished: Thu Aug 23 11:41:33 2018
```

Cycle Approximate simulation including custom instructions

- IA simulation + timing annotation + custom instructions
  - Includes timing estimation for RV32IM processor
  - Need to add timing estimation for new custom instructions
- Simulate using C code application with inline assembler of custom extensions
- IA simulator + timing tool + custom extension instruction library

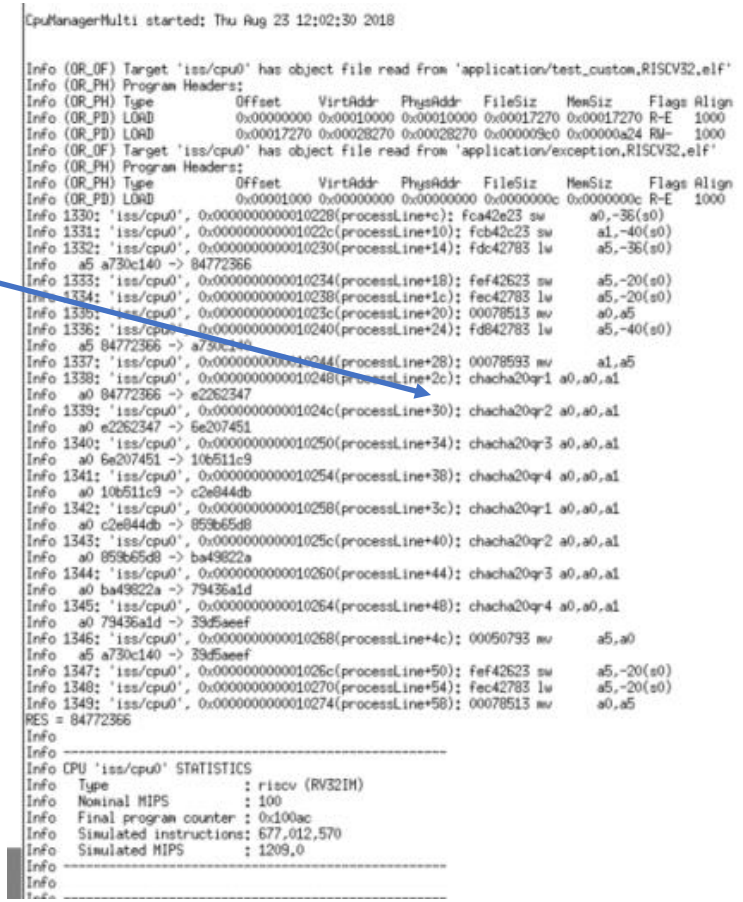
➢ See estimated improvement in throughput of application on new processor
  - ➢ Was 16.59 secs without custom instructions
  - ➢ ***Now 9.21 secs with custom instructions***





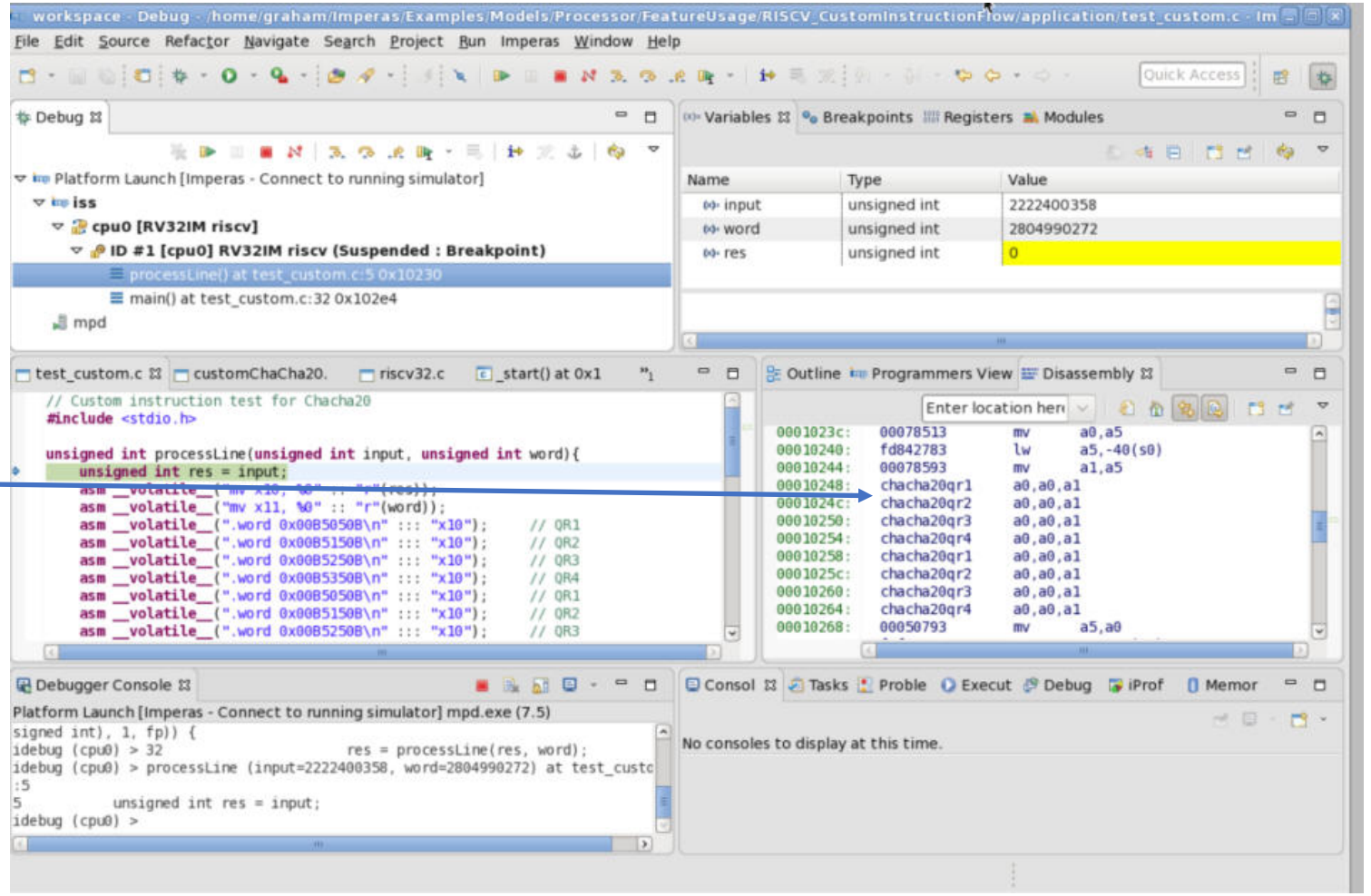
Trace custom instructions

- Simulator has many trace features built in
- See new custom instructions in trace disassembly
- Can select when/where to turn trace on/off
  - Very efficient tracing



Debug custom instructions

- Imperas MPD is Eclipse based source code debug tool
- Can debug using source line or instruction level
- See new custom instructions and any new additional state registers

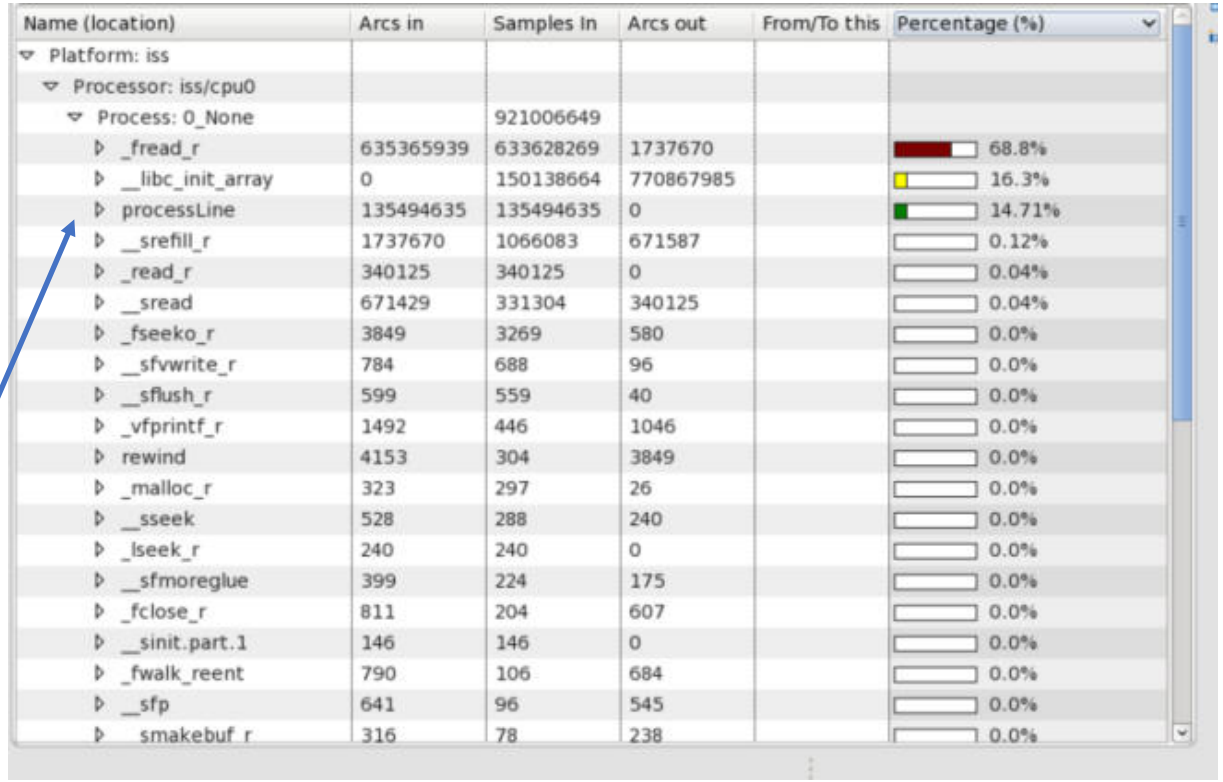


Function Profile custom instructions application

- IA simulation + timing annotation + custom instructions with sampled profiling

➢Shows where slowest function is
  - Now much faster…

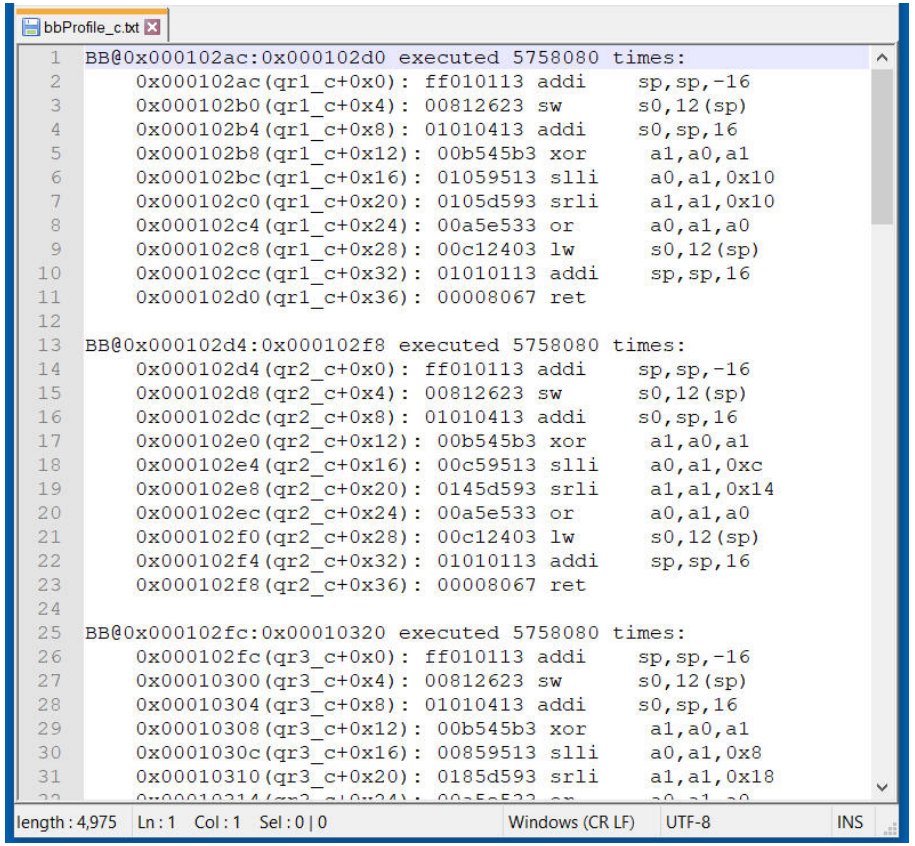
➢Shows benefits of using custom instructions
  ➢processLine was 21.35% now 16.3%



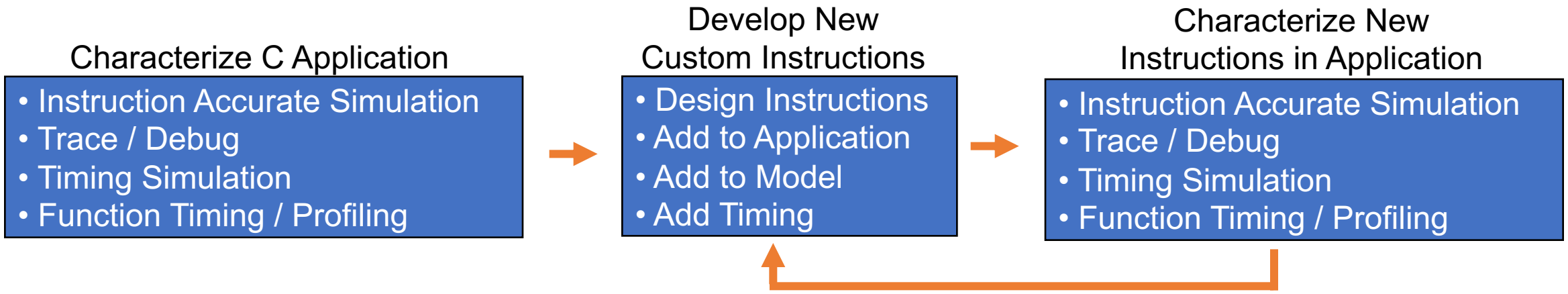
Name (location)	Arcs in	Samples in	Arcs out	From/To this	Percentage (%)
Platform: iss					
Processor: iss/cpu0					
Process: 0_None		921006649			
_fread_r	635365939	633628269	1737670		68.8%
__libc_init_array	0	150138664	770867985		16.3%
processLine	135494635	135494635	0		14.71%
__srefill_r	1737670	1066083	671587		0.12%
_read_r	340125	340125	0		0.04%
__sread	671429	331304	340125		0.04%
_fseeko_r	3849	3269	580		0.0%
__sfvwrite_r	784	688	96		0.0%
__sflush_r	599	559	40		0.0%
_vfprintf_r	1492	446	1046		0.0%
rewind	4153	304	3849		0.0%
_malloc_r	323	297	26		0.0%
__sseek	528	288	240		0.0%
_lseek_r	240	240	0		0.0%
__sfmoreglue	399	224	175		0.0%
_fclose_r	811	204	607		0.0%
__sinit.part.1	146	146	0		0.0%
_fwalk_reent	790	106	684		0.0%
__sfp	641	96	545		0.0%
smakebuf_r	316	78	238		0.0%

Basic Block (BB) Profile custom instructions application

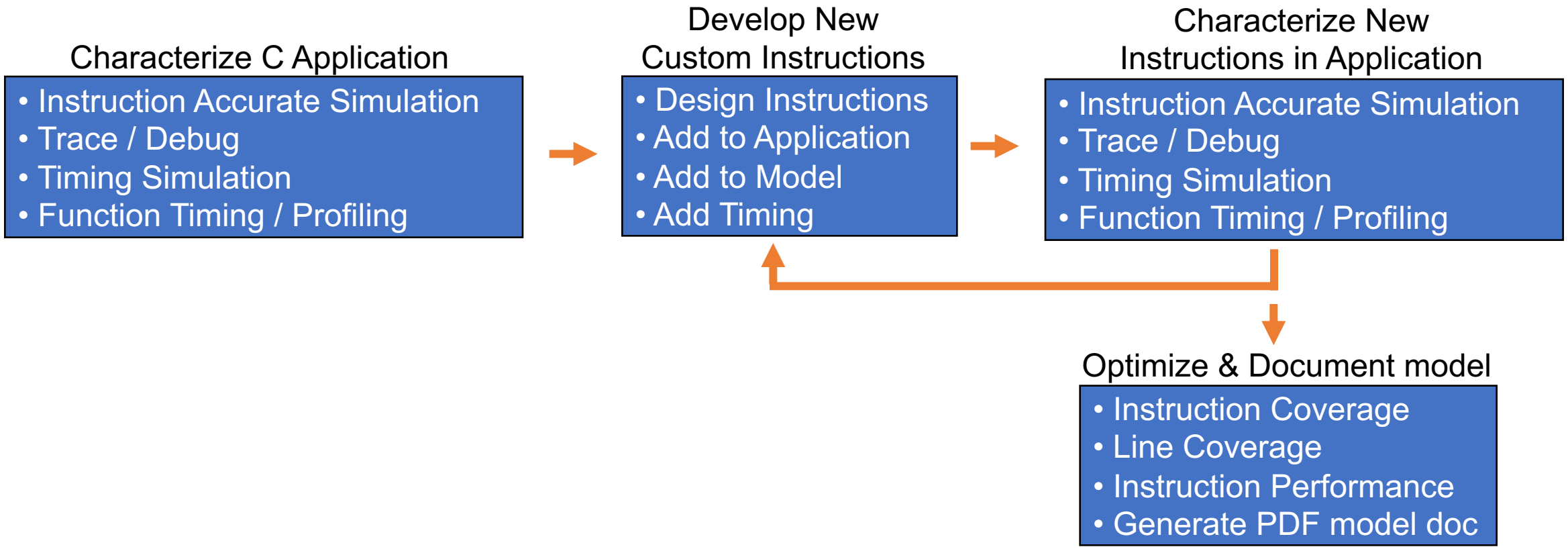
- IA simulation + timing annotation + custom instructions with detailed BB profiling
- Shows where expensive instruction sequences are
- Allows understanding of instruction performance
  - Useful for Compiler teams
  - Useful for Hardware teams



Flow to add new custom instructions



Flow to add new custom instructions

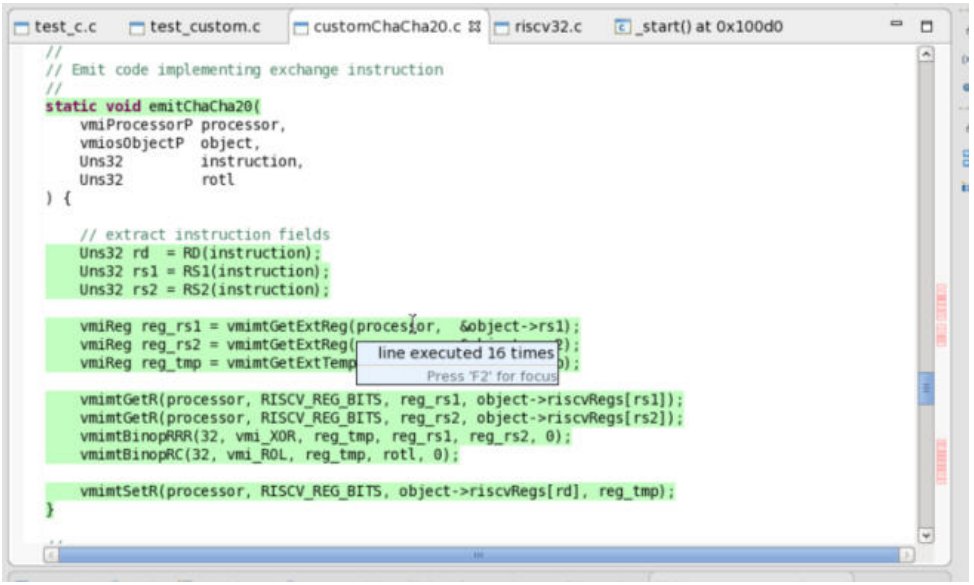


Further tools for model developers

- Model source line coverage
  - To see how completely the tests exercise the model

type filter text

Name	Total Lines	Instrumente	Executed Lir	Coverage %
Summary	16,900	4,411	2,834	64.25%
customChaCha20.c	374	87	42	48.28%
riscvBus.c	175	46	1	2.17%
riscvCSR.c	2,573	758	549	72.43%
riscvConfigList.c	70	2	0	0.0%
riscvDebug.c	527	167	72	43.11%
riscvDecode.c	1,599	360	164	45.56%
riscvDisassemble.c	514	185	185	100.0%
riscvDoc.c	725	143	30	20.98%
riscvExceptions.c	1,408	420	317	75.48%
riscvInfo.c	83	3	0	0.0%
riscvMain.c	383	147	25	17.01%
riscvMorph.c	2,887	1,032	776	75.19%
riscvParameters.c	589	145	9	6.21%
riscvRegisterTypes.h	115	10	6	60.0%
riscvSemiHost.c	44	6	3	50.0%
riscvStructure.h	204	4	2	50.0%
riscvUtils.c	493	122	45	36.89%
riscvVM.c	2,695	770	608	78.96%
vmiMt.h	1,442	4	0	0.0%



Further tools for model developers (2)

- Custom instruction coverage
  - To see that there are tests for new instructions

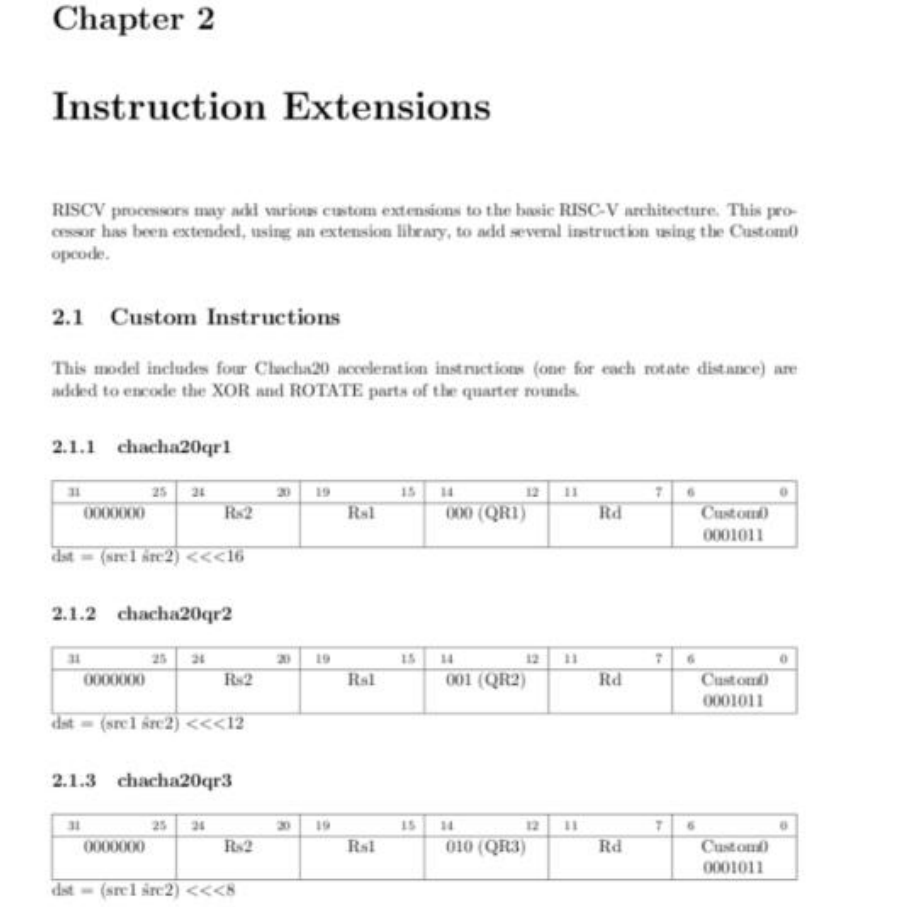
```
CpuManagerMulti finished: Thu Aug 23 12:09:15 2018

CpuManagerMulti (32-Bit) v99999999 Open Virtual Platform simulator from www.IMPERAS.com.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

* iss/cpu0: Instruction Profile Totals:
*         OpCode :    Pct : Count (Total instrs=      677012570)
* ---------------- : ------ : ----------
*             sw : 19.87% :      134497608
*             lw : 16.78% :      113575358
*             sw : 13.05% :       88327451
*           addi :  8.70% :       58868837
*           andi :  4.97% :       33620233
*           beqz :  4.35% :       29426103
*           bnez :  3.73% :       25248055
*            add :  3.73% :       25231546
*           slli :  3.10% :       21020892
*            ret :  2.49% :       16843028
*            jal :  2.49% :       16843026
*           srli :  2.48% :       16810160
*           bltu :  2.48% :       16793686
*           bgeu :  1.24% :        8421494
*           bltz :  1.24% :        8421470
*              j :  1.24% :        8388742
*       chacha20qr1 :  1.24% :        8388608
*       chacha20qr2 :  1.24% :        8388608
*       chacha20qr3 :  1.24% :        8388608
*       chacha20qr4 :  1.24% :        8388608
*             lh :  0.63% :        4243601
*          auipc :  0.62% :        4210858
*            sub :  0.62% :        4210835
*            xor :  0.62% :        4210707
*            not :  0.62% :        4194307
*            beq :  0.00% :          16546
*           jalr :  0.00% :          16430
*           blez :  0.00% :          16407
*             sb :  0.00% :            206
*            lhu :  0.00% :             97
*             sh :  0.00% :             82
*            lui :  0.00% :             67
*             or :  0.00% :             50
*            and :  0.00% :             49
*            lbu :  0.00% :             47
*            bne :  0.00% :             34
*            ori :  0.00% :             29
*             jr :  0.00% :             25
*            bge :  0.00% :             17
*            neg :  0.00% :             16
*            blt :  0.00% :             12
*           srai :  0.00% :              8
*           bgtz :  0.00% :              7
*           bgez :  0.00% :              6
*            sll :  0.00% :              4
```

- Custom instruction profile
  - See how long the simulator takes to execute each instruction
  - Use to focus speed up simulation of instructions
  - Enables improvement of speed of simulation runs

```
-------------------------------------------------------------------------------
PROFILE REPORT
-------------------------------------------------------------------------------
    sw                0,27s      88,327,451   3.1ps/instruction (iss/cpu0)
    lw                0,20s     113,575,358   1.8ps/instruction (iss/cpu0)
    sw                0,16s     134,497,608   1.2ps/instruction (iss/cpu0)
    ret               0,07s      16,843,028   4.2ps/instruction (iss/cpu0)
    jal               0,06s      16,843,026   3.6ps/instruction (iss/cpu0)
    addi              0,05s      58,868,837   0.8ps/instruction (iss/cpu0)
    andi              0,05s      33,620,233   1.5ps/instruction (iss/cpu0)
    add               0,05s      25,231,546   2.0ps/instruction (iss/cpu0)
    auipc             0,04s       4,210,858   9.5ps/instruction (iss/cpu0)
    beqz              0,03s      29,426,103   1.0ps/instruction (iss/cpu0)
    bnez              0,03s      25,248,055   1.2ps/instruction (iss/cpu0)
    chacha20qr3       0,03s       8,388,608   3.6ps/instruction (iss/cpu0)
    (JIT translation) 0,03s           2,500   12.0ns/instruction
    bltu              0,02s      16,793,686   1.2ps/instruction (iss/cpu0)
    chacha20qr1       0,02s       8,388,608   2.4ps/instruction (iss/cpu0)
    lh                0,02s       4,243,601   4.7ps/instruction (iss/cpu0)
    xor               0,02s       4,210,707   4.7ps/instruction (iss/cpu0)
    slli              0,01s      21,020,892   0.5ps/instruction (iss/cpu0)
    srli              0,01s      16,810,160   0.6ps/instruction (iss/cpu0)
    bltz              0,01s       8,421,470   1.2ps/instruction (iss/cpu0)
    j                 0,01s       8,388,742   1.2ps/instruction (iss/cpu0)
    not               0,01s       4,194,307   2.4ps/instruction (iss/cpu0)
    (unallocated)     0,01s
    bgeu              0,00s       8,421,494   0.0ps/instruction (iss/cpu0)
    chacha20qr2       0,00s       8,388,608   0.0ps/instruction (iss/cpu0)
    chacha20qr4       0,00s       8,388,608   0.0ps/instruction (iss/cpu0)
    sub               0,00s       4,210,835   0.0ps/instruction (iss/cpu0)
    beq               0,00s          16,546   0.0ps/instruction (iss/cpu0)
    jalr              0,00s          16,430   0.0ps/instruction (iss/cpu0)
    blez              0,00s          16,407   0.0ps/instruction (iss/cpu0)
    sb                0,00s             206   0.0ps/instruction (iss/cpu0)
    lhu               0,00s              97   0.0ps/instruction (iss/cpu0)
    sh                0,00s              82   0.0ps/instruction (iss/cpu0)
    lui               0,00s              67   0.0ps/instruction (iss/cpu0)
    or                0,00s              50   0.0ps/instruction (iss/cpu0)
    and               0,00s              49   0.0ps/instruction (iss/cpu0)
    lbu               0,00s              47   0.0ps/instruction (iss/cpu0)
    bne               0,00s              34   0.0ps/instruction (iss/cpu0)
    ori               0,00s              29   0.0ps/instruction (iss/cpu0)
    jr                0,00s              25   0.0ps/instruction (iss/cpu0)
    TOTAL             1,21s     677,012,570
-------------------------------------------------------------------------------
Info
```

Document custom instructions

- Imperas tools automatically generate a processor model document PDF
- Includes all base model registers and any new registers
- Provides detailed documentation of new custom instructions



Chapter 2

Instruction Extensions

RISCV processors may add various custom extensions to the basic RISC-V architecture. This processor has been extended, using an extension library, to add several instruction using the Custom0 opcode.

2.1 Custom Instructions

This model includes four Chacha20 acceleration instructions (one for each rotate distance) are added to encode the XOR and ROTATE parts of the quarter rounds.

2.1.1 chacha20qr1

31 25	24 20	19 15	14 12	11 7	6 0
0000000	Rs2	Rs1	000 (QR1)	Rd	Custom0 0001011

dst = (src1 ŝrc2) <<<16

2.1.2 chacha20qr2

31 25	24 20	19 15	14 12	11 7	6 0
0000000	Rs2	Rs1	001 (QR2)	Rd	Custom0 0001011

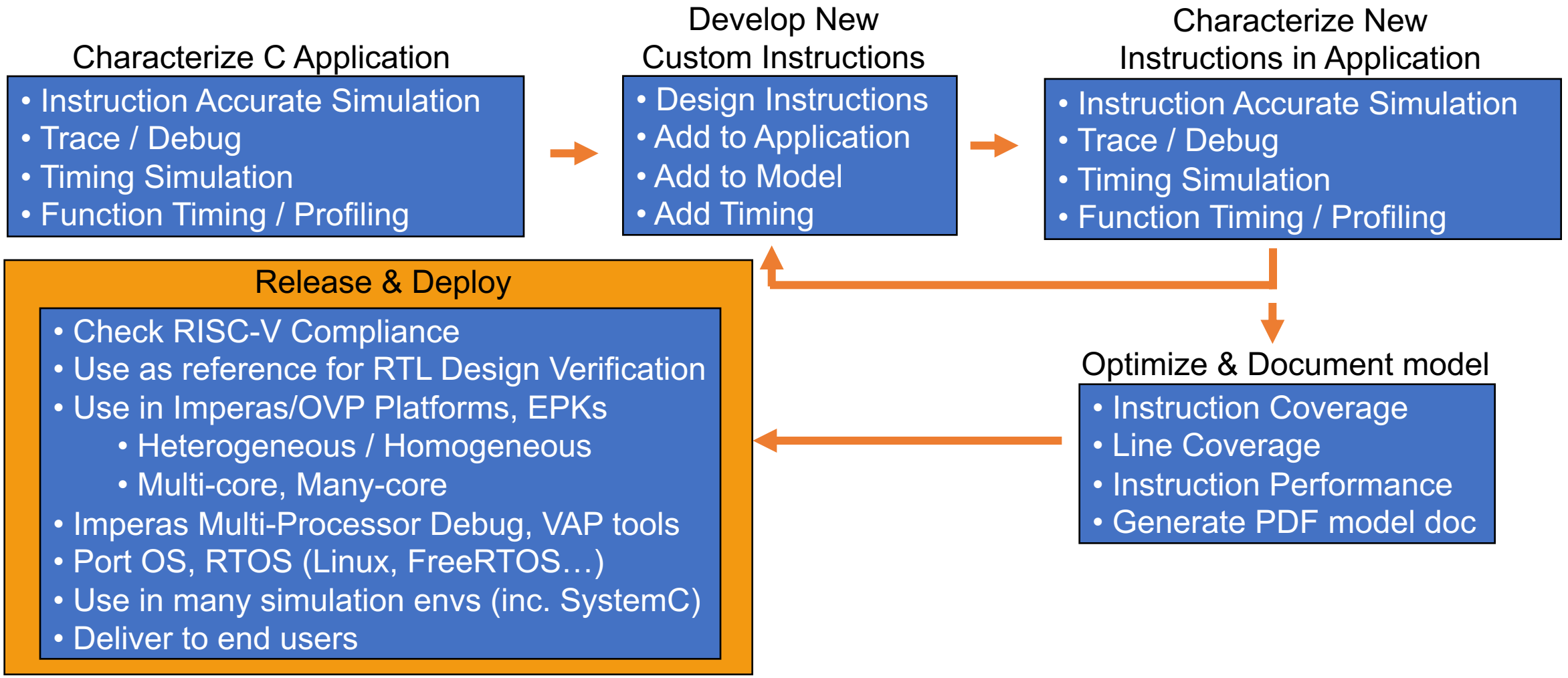
dst = (src1 ŝrc2) <<<12

2.1.3 chacha20qr3

31 25	24 20	19 15	14 12	11 7	6 0
0000000	Rs2	Rs1	010 (QR3)	Rd	Custom0 0001011

dst = (src1 ŝrc2) <<<8

Flow to add new custom instructions



Summary

- Compliance is a new challenge for open ISAs
- Additional verification tasks related to the open ISA processor
- Custom instructions are a key value proposition of RISC-V
  - Adding custom instructions requires solving the key challenge of how to optimize those instructions
- Instruction accurate (IA) simulation environment using IA models can be extended to enable custom instruction analysis and optimization
- Flow for optimizing custom instructions in RISC-V processors is being used in real designs

Thank You

- Visit www.imperas.com/riscv and www.OVPworld.org/riscv for more information

- RISC-V Foundation Compliance Suite, includes riscvOVPsim available:
  - https://github.com/riscv/riscv-compliance

Larry Lapides
VP Sales, Imperas Software
larryl@imperas.com