



# Methodology for Implementation of Custom Instructions in RISC-V Architecture

Kevin McDermott - VP Marketing, Imperas

26th-28th February 2019



# Agenda

- Who is adopting RISC-V and why...

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



- There are and will be many different RISC-V CPU developers:
  - Develop internal cores for SoCs
    - e.g. Nvidia, ...
  - Deliver RTL IP as a business (like Arm, MIPS, just for RISC-V)
    - e.g. Andes, Codaip, Syntacore, Incore, ...
  - 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, ...
  - Purchase processor IP and develop semiconductor products
    - e.g. Microsemi SoC FPGAs, ...
  - Download and use open source RTL in their products
    - e.g. 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

# Agenda



- Who is adopting RISC-V and why...
- Adding custom instructions to RISC-V processors

# 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



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



# Custom Instruction Optimization Flow



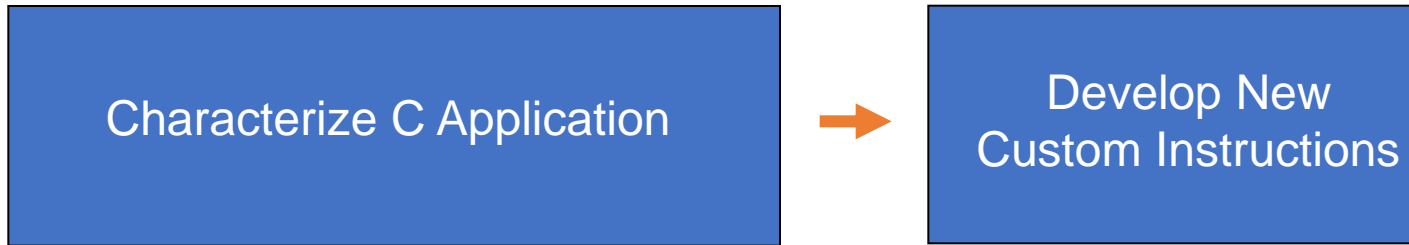
- Show the flow used when designing new instructions to improve performance of applications running on a RISC-V processor
    - Allows evaluation and firming up of the instructions
    - Gets to the specification of the instructions needed to be implemented in RTL
  - Introduce the technologies & tools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



Characterize C Application

# Flow to add new custom instructions



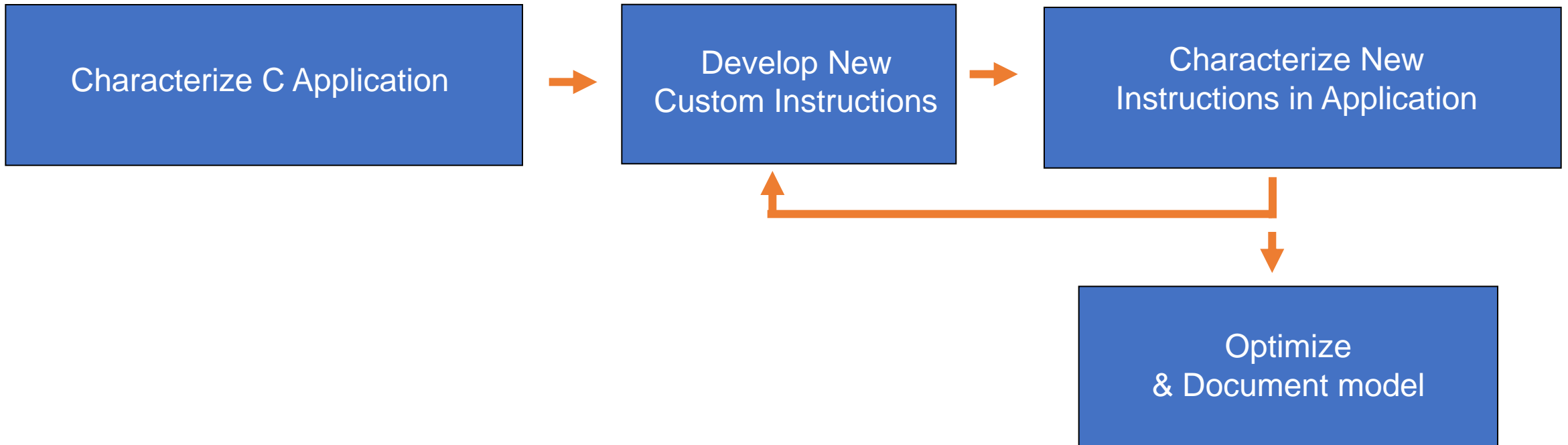
# Flow to add new custom instructions



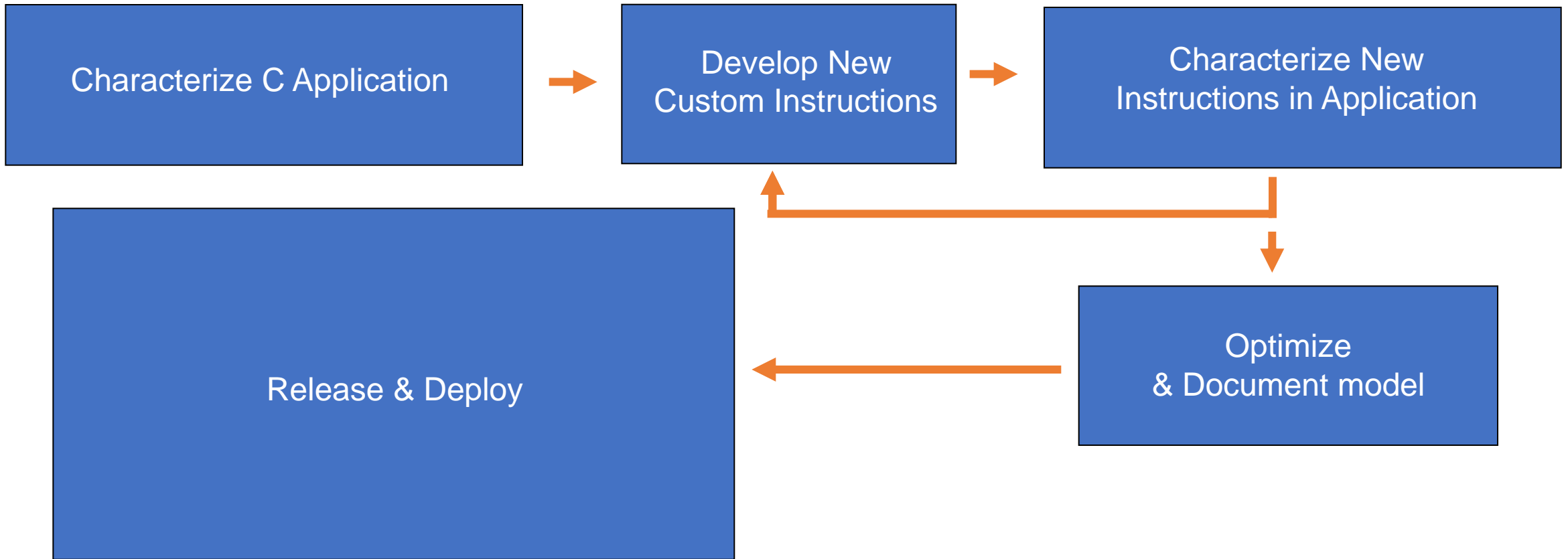
# Flow to add new custom instructions



# Flow to add new custom instructions



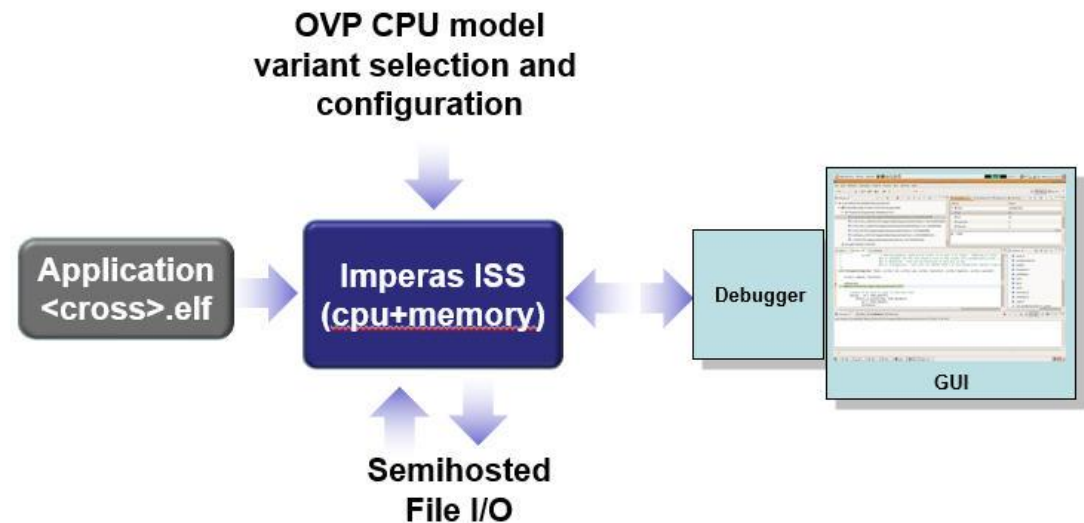
# Flow to add new custom instructions



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

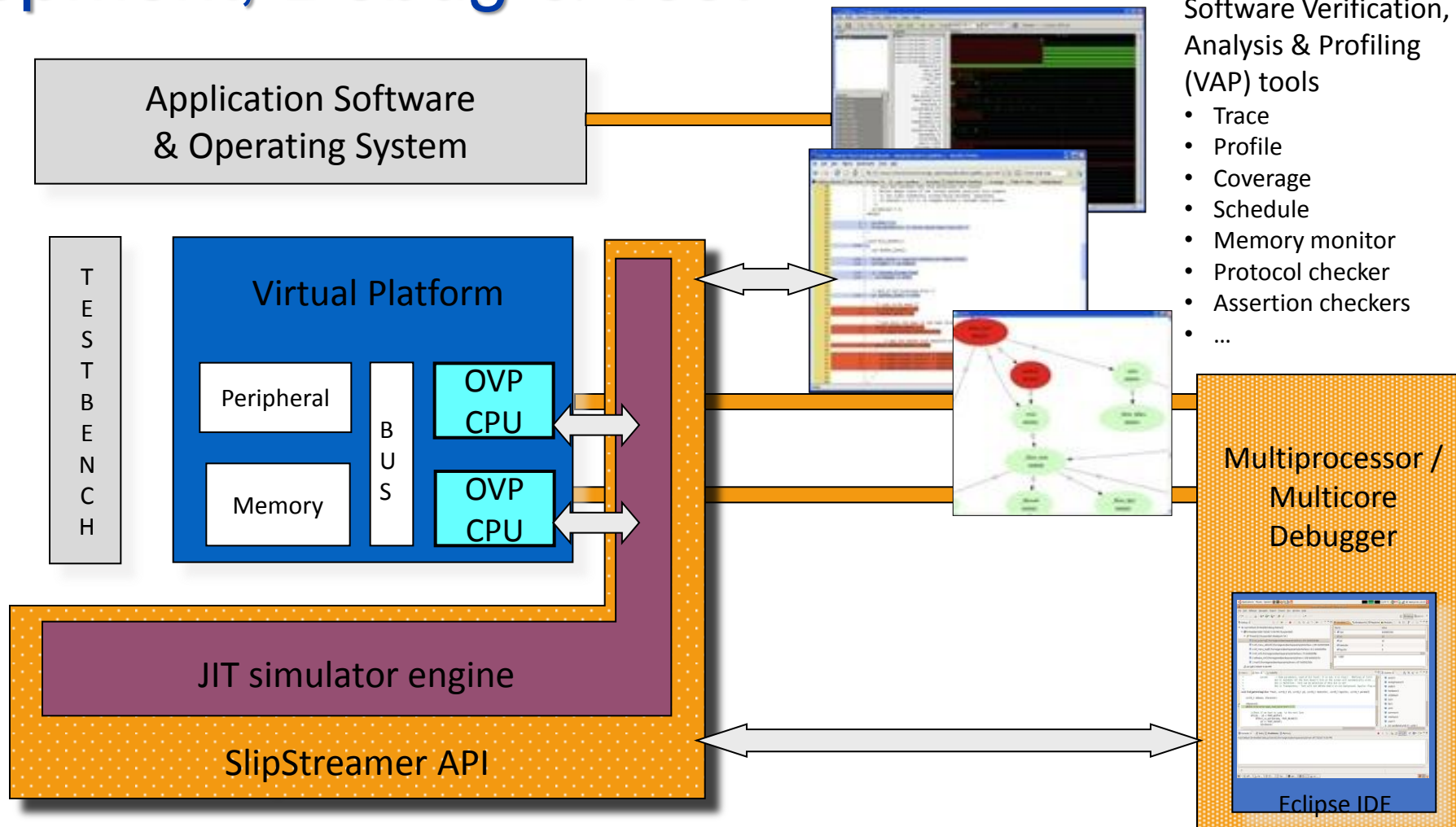
**imperas**

- 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 Tools for Embedded Software Development, Debug & Test



# 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

A screenshot of a development environment showing a C program named 'test\_c.c' and its execution output. The code defines a 'processLine' function that processes a word using a series of 'qr' functions and a 'main' function that reads data from a file and processes it in a loop. The output on the right shows the program's execution details, including target information, program headers, and performance statistics. A blue arrow points from the 'Simulated MIPS' value in the statistics to the 'runs fast' bullet point in the list on the left.

```
test_c.c
unsigned int processLine(unsigned int res, unsigned int word){
    res = qr1_c(res, word);
    res = qr2_c(res, word);
    res = qr3_c(res, word);
    res = qr4_c(res, word);
    res = qr1_c(res, word);
    res = qr2_c(res, word);
    res = qr3_c(res, word);
    res = qr4_c(res, word);
    return res;
}

int main(void) {
    const char *customData = "application/custom.data";
    FILE *fp = fopen(customData, "r");
    if (fp) {
        unsigned int res = 0x84772366;
        unsigned int word;
        unsigned int cnt=0;
        unsigned int iter=0;
        while (iter++ < 16) {
            while (fread(&word,sizeof(unsigned int), 1, fp)) {
                res = processLine(res, word);
            }
            rewind(fp);
        }
        fclose(fp);
        printf("RES = %08x", res);
    } else {
        printf("Failed to open file\n");
    }
    return 0;
}
```

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 11:19:21 2018

Info (OR\_OF) Target 'iss/cpu0' has object file read from 'application/test\_c.RISCV32.elf'

Info (OR_PH) Program Headers:	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flags	Align
Info (OR_PH) Type	0x00000000	0x00010000	0x00010000	0x000173c8	0x000173c8	R-E	1000
Info (OR_PD) LOB	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:	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flags	Align
Info (OR_PH) Type	0x00001000	0x00000000	0x00000000	0x0000000c	0x0000000c	R-E	1000

RES = 84772366

Info

Info CPU 'iss/cpu0' STATISTICS	
Info Type	: riscv (RV32IM)
Info Nominal MIPS	: 100
Info Final program counter	: 0x100ac
Info Simulated instructions	: 1,289,390,976
Info Simulated MIPS	: 1151.2

Info

Info SIMULATION TIME STATISTICS	
Info Simulated time	: 12.89 seconds
Info User time	: 1.10 seconds
Info System time	: 0.02 seconds
Info Elapsed time	: 1.14 seconds
Info Real time ratio	: 11.51x Faster

Info

CpuManagerMulti finished: Thu Aug 23 11:19:22 2018

# 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 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 0x00000000c 0x00000000c 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 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 --print lastRun.sh to review 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	1824985	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%
▸ _flush_r	117	117	40		0.0%

# Flow to add new custom instructions

## Characterize C Application

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



## Develop New Custom Instructions

- Design Instructions
- Add to Application
- Add to Model
- Add Timing



# 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 0x00850508\n" :: "x10"); // QR1
    asm __volatile__(".word 0x00851508\n" :: "x10"); // QR2
    asm __volatile__(".word 0x00852508\n" :: "x10"); // QR3
    asm __volatile__(".word 0x00853508\n" :: "x10"); // QR4
    asm __volatile__(".word 0x00850508\n" :: "x10"); // QR1
    asm __volatile__(".word 0x00851508\n" :: "x10"); // QR2
    asm __volatile__(".word 0x00852508\n" :: "x10"); // QR3
    asm __volatile__(".word 0x00853508\n" :: "x10"); // QR4
    asm __volatile__("mv %0,x10" :: "r"(res));
    return res;
}

int main() {
    // ...
}

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 0x00000000 0x00000000 R-E 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 0x00000000 0x00000000 R-E 1000
Warning (RISCV_RIF) 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 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] = 00000000
    // 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 = vmimGetExtReg(processor, &object->rs1);
    vmiReg reg_rs2 = vmimGetExtReg(processor, &object->rs2);
    vmiReg reg_tmp = vmimGetExtTemp(processor, &object->tmp);

    vmimGetR(processor, RISCV_REG_BITS, reg_rs1, object->riscvRegs[rs1]);
    vmimGetR(processor, RISCV_REG_BITS, reg_rs2, object->riscvRegs[rs2]);
    vmimBinopRRR(32, vmi_XOR, reg_tmp, reg_rs1, reg_rs2, 0);
    vmimBinopRC(32, vmi_ROL, reg_tmp, reg_rs1, 0);

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

```
CpuManagerHwlti (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.

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

Info (OP_LPR) Processor iss/cpu0 $IMPERAS_VLNV/riscv.ovpworld.
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 FileSize MemSize Flags Align
Info (OR_PD) LOAD 0x00000000 0x00010000 0x00010000 0x00017270 R-E 1000
Info (OR_PD) LOAD 0x00017270 0x00028270 0x00028270 0x00000000 R-W 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 FileSize MemSize Flags Align
Info (OR_PD) LOAD 0x00000000 0x00000000 0x00000000 0x00000000 R-E 1000
Info (OR_PEX) Extension iss/cpu0/riscv32Newlib $IMPERAS_VLNV/riscv.ovpworld.org/semihosting/riscv32Newlib/1.0/model
Info (OR_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 : 1304.3
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
CpuManagerHwlti 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 (IA) 6.77 secs now (CA) 9.21 secs

A screenshot of the Imperas IDE interface. The top pane shows a C file named 'test\_custom.c' with code defining instruction cycles for 'IC\_mdu\_mul', 'IC\_mdu\_div', and 'IC\_custom'. The bottom pane shows the simulation output, including statistics for the RV32IM processor and a comparison of simulated vs. real time.

```
case IC_mdu_mul : {
    cycles = 5;      // Specify cycles for instruction group
    break;
}
case IC_mdu_div : {
    cycles = 16;     // Specify cycles for instruction group
    break;
}
case IC_custom : {
    // chacha20qr1-chacha20qr4 group same cycles
    cycles = 2;      // Specify cycles for instruction group
    break;
}
default: {
    VMI_ABORT("Invalid instructionClassE value %d (%s)\n", iClass, instrClassName(iClass));
    break;
}
}
```

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 : 0xd00ac  
Info Simulated instructions: 677,012,570  
Info Simulated MIPS : 123.1  
Info  
Info  
Info SIMULATION TIME STATISTICS  
Info Simulated time : 9.21 seconds  
Info User time : 5.30 seconds  
Info System time : 0.20 seconds  
Info Elapsed time : 5.73 seconds  
Info Real time ratio : 1.61x faster  
Info  
CpuManagerMulti finished: Thu Aug 23 11:58:35 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 9.21 seconds, clock cycles 921,006,928  
Use script lastRun.sh to re-run with current settings

# 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

```
CpuManagerMulti started: Thu Aug 23 12:02:30 2018

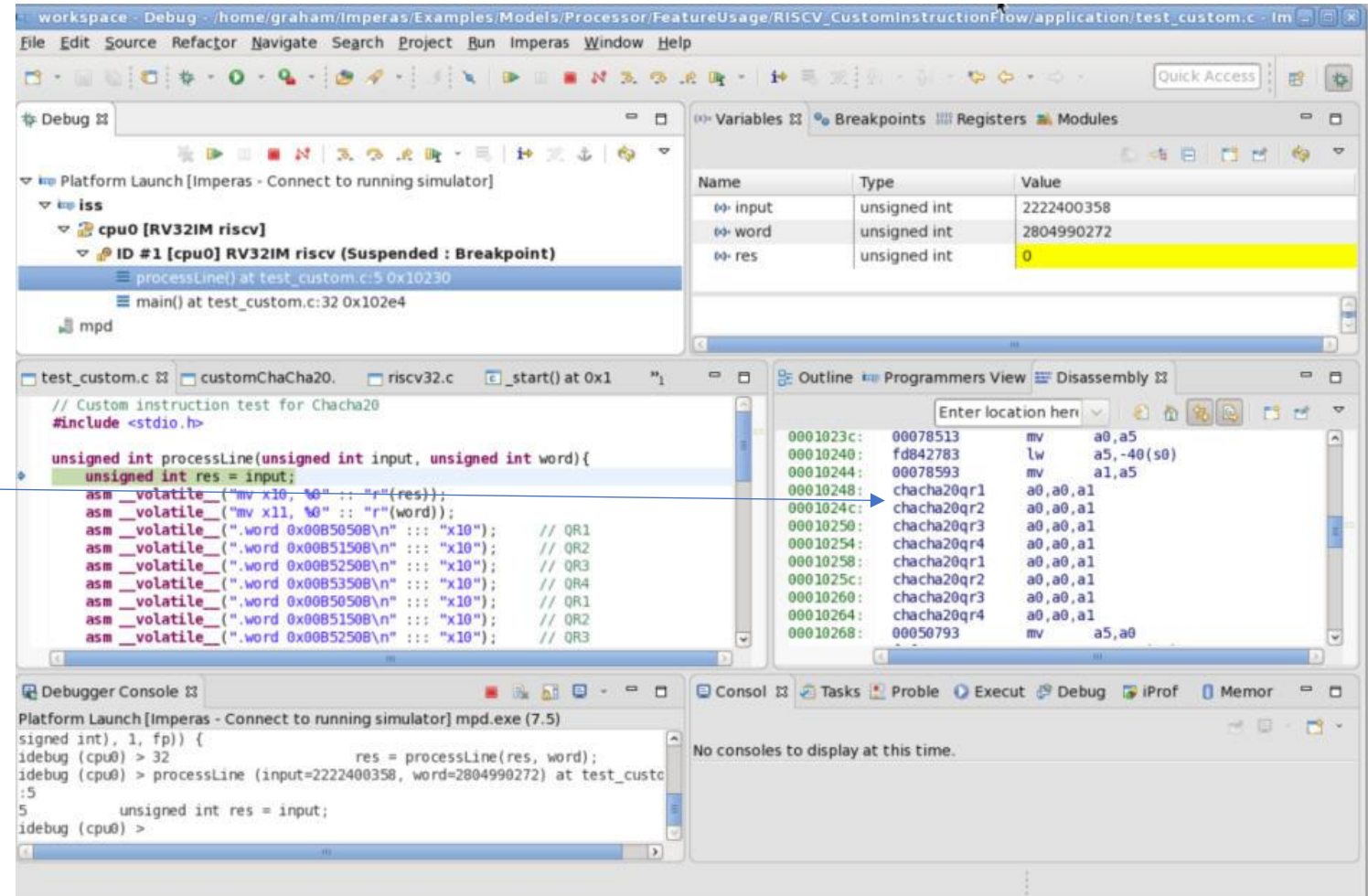
Info (OR_OF) Target 'iss/cpu0' has object file read from 'application/test_custom,RISCV32.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type      Offset      VirtAddr  PhysAddr  FileSiz  MemSiz  Flags Align
Info (OR_PD) LOAD      0x00000000 0x00010000 0x00010000 0x00017270 0x00017270 R-E 1000
Info (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 0x00000000 0x00000000 R-E 1000
Info 1330: 'iss/cpu0', 0x0000000000001028(processLine+): fca42e23 sw    a0,-36(a0)
Info 1331: 'iss/cpu0', 0x0000000000001022c(processLine+10): fcb42c23 sw    a1,-40(a0)
Info 1332: 'iss/cpu0', 0x00000000000010230(processLine+14): fdc42783 lw    a5,-36(a0)
Info    a5 a730c140 -> 84772366
Info 1333: 'iss/cpu0', 0x00000000000010234(processLine+18): feF42623 sw    a5,-20(a0)
Info 1334: 'iss/cpu0', 0x00000000000010238(processLine+1c): fec42783 lw    a5,-20(a0)
Info 1335: 'iss/cpu0', 0x0000000000001023c(processLine+20): 00078513 mv    a0,a5
Info 1336: 'iss/cpu0', 0x00000000000010240(processLine+24): fd842783 lw    a5,-40(a0)
Info    a5 84772366 -> a730c140
Info 1337: 'iss/cpu0', 0x00000000000010244(processLine+28): 00078593 mv    a1,a5
Info 1338: 'iss/cpu0', 0x00000000000010248(processLine+2c): chacha20qr1 a0,a0,a1
Info    a0 84772366 -> e2262347
Info 1339: 'iss/cpu0', 0x0000000000001024c(processLine+30): chacha20qr2 a0,a0,a1
Info    a0 e2262347 -> 6e207451
Info 1340: 'iss/cpu0', 0x00000000000010250(processLine+34): chacha20qr3 a0,a0,a1
Info    a0 6e207451 -> 10b511c9
Info 1341: 'iss/cpu0', 0x00000000000010254(processLine+38): chacha20qr4 a0,a0,a1
Info    a0 10b511c9 -> c2e844db
Info 1342: 'iss/cpu0', 0x00000000000010258(processLine+3c): chacha20qr1 a0,a0,a1
Info    a0 c2e844db -> 853b65d8
Info 1343: 'iss/cpu0', 0x0000000000001025c(processLine+40): chacha20qr2 a0,a0,a1
Info    a0 853b65d8 -> ba49822a
Info 1344: 'iss/cpu0', 0x00000000000010260(processLine+44): chacha20qr3 a0,a0,a1
Info    a0 ba49822a -> 79436a1d
Info 1345: 'iss/cpu0', 0x00000000000010264(processLine+48): chacha20qr4 a0,a0,a1
Info    a0 79436a1d -> 39d5aeef
Info 1346: 'iss/cpu0', 0x00000000000010268(processLine+4c): 00050793 mv    a5,a0
Info    a5 a730c140 -> 39d5aeef
Info 1347: 'iss/cpu0', 0x0000000000001026c(processLine+50): feF42623 sw    a5,-20(a0)
Info 1348: 'iss/cpu0', 0x00000000000010270(processLine+54): fec42783 lw    a5,-20(a0)
Info 1349: 'iss/cpu0', 0x00000000000010274(processLine+58): 00078513 mv    a0,a5
RES = 84772366

Info
Info -----
Info CPU 'iss/cpu0' STATISTICS
Info Type      : riscv (RV32IH)
Info Nominal MIPS : 100
Info Final program counter : 0x100ac
Info Simulated instructions: 677,012,570
Info Simulated MIPS : 1209,0
Info -----
Info
```

# Debug custom instructions

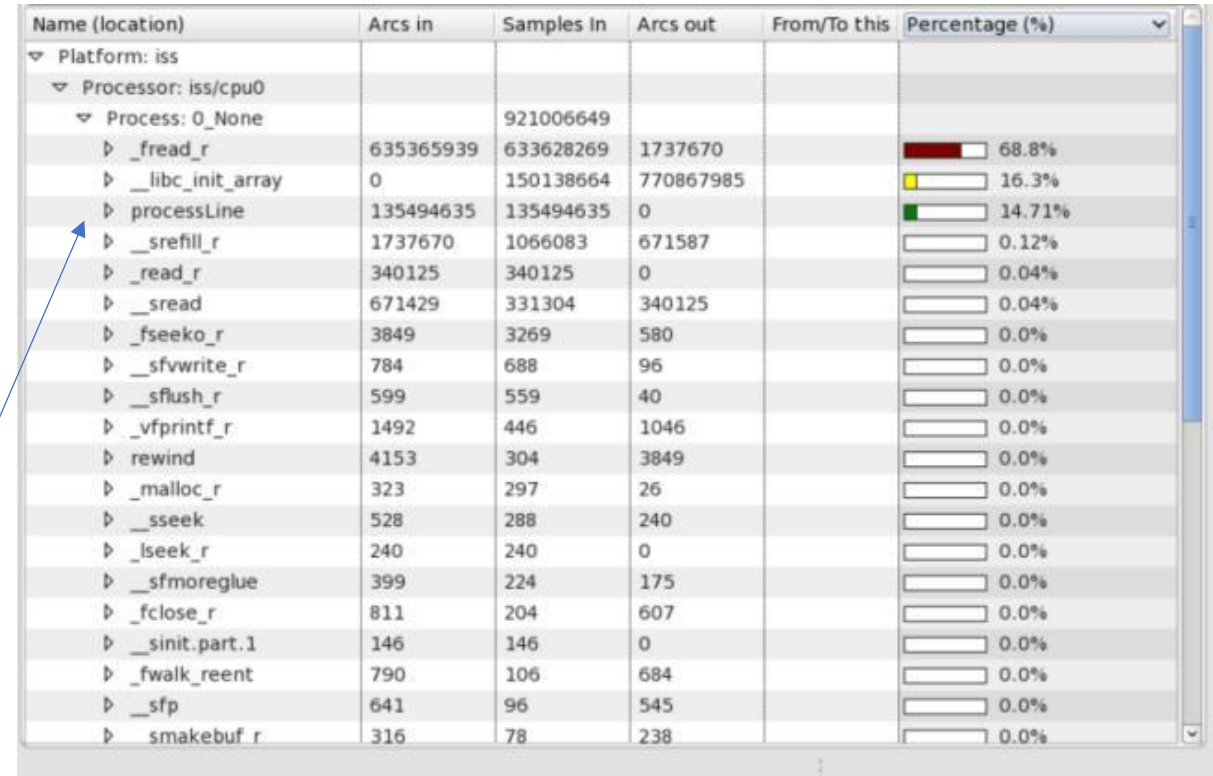


- Imperas MPD is Eclipse based source code debug tool
- Can debug using source line or instruction level
- See new custom instructions and any new additional state registers



# 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%
▸ _sfwrite_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%
▸ _seek	528	288	240		0.0%
▸ _lseek_r	240	240	0		0.0%
▸ _sfmorglue	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

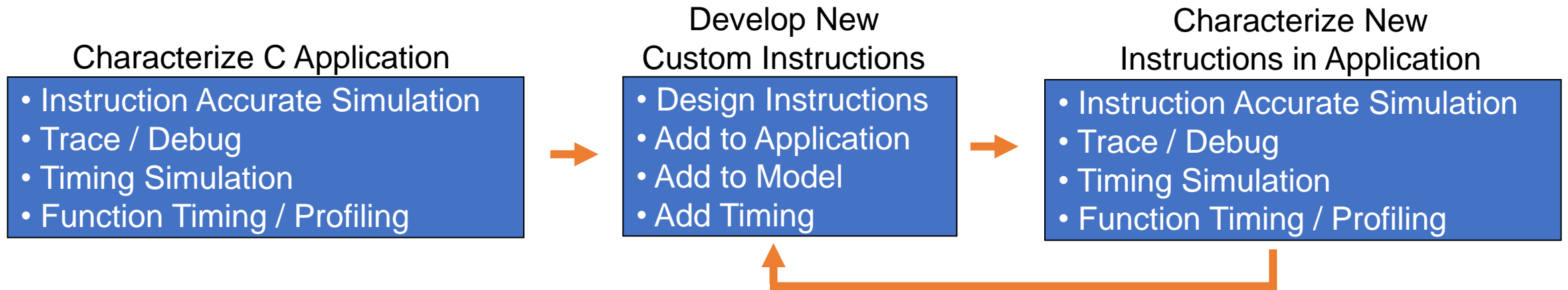


- IA simulation + timing annotation + custom instructions with detailed BB profiling
- Shows where expensive instruction sequences are
- Allows understanding of instruction performance
  - Useful for Compiler teams
  - Useful for Hardware teams

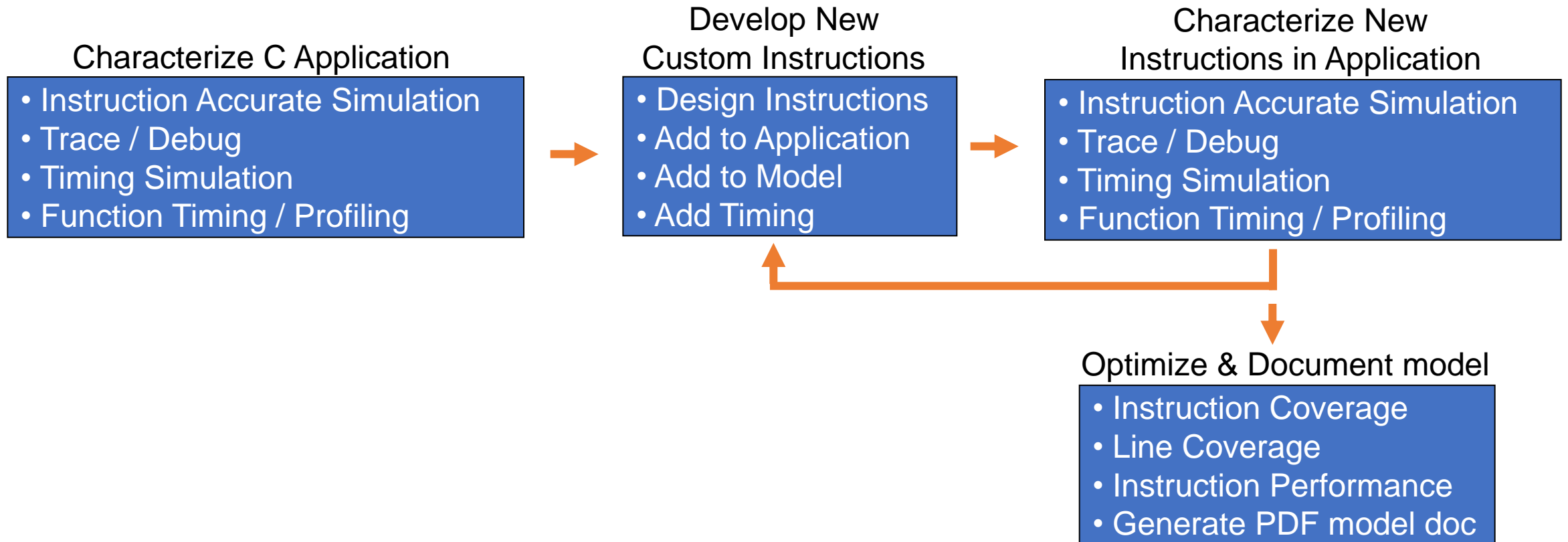
A screenshot of a text editor window titled "bbProfile\_c.txt". It displays three Basic Block (BB) profiles, each with a list of instructions and their execution counts. The first BB starts at address 0x000102ac and is executed 5758080 times. The second BB starts at 0x000102d4 and is also executed 5758080 times. The third BB starts at 0x000102fc and is executed 5758080 times. Each instruction is shown with its address, a comment in parentheses, and the instruction itself. The instructions are mostly RISC-V style, including addi, sw, add, xor, slli, srli, or, lw, and ret. The status bar at the bottom shows "length: 4,975", "Ln: 1", "Col: 1", "Sel: 0 | 0", "Windows (CR LF)", "UTF-8", and "INS".

```
bbProfile_c.txt
1 BB@0x000102ac:0x000102d0 executed 5758080 times:
2   0x000102ac(qr1_c+0x0): ff010113 addi    sp,sp,-16
3   0x000102b0(qr1_c+0x4): 00812623 sw      s0,12(sp)
4   0x000102b4(qr1_c+0x8): 01010413 addi    s0,sp,16
5   0x000102b8(qr1_c+0x12): 00b545b3 xor    a1,a0,a1
6   0x000102bc(qr1_c+0x16): 01059513 slli    a0,a1,0x10
7   0x000102c0(qr1_c+0x20): 0105d593 srli    a1,a1,0x10
8   0x000102c4(qr1_c+0x24): 00a5e533 or     a0,a1,a0
9   0x000102c8(qr1_c+0x28): 00c12403 lw      s0,12(sp)
10  0x000102cc(qr1_c+0x32): 01010113 addi    sp,sp,16
11  0x000102d0(qr1_c+0x36): 00008067 ret
12
13 BB@0x000102d4:0x000102f8 executed 5758080 times:
14  0x000102d4(qr2_c+0x0): ff010113 addi    sp,sp,-16
15  0x000102d8(qr2_c+0x4): 00812623 sw      s0,12(sp)
16  0x000102dc(qr2_c+0x8): 01010413 addi    s0,sp,16
17  0x000102e0(qr2_c+0x12): 00b545b3 xor    a1,a0,a1
18  0x000102e4(qr2_c+0x16): 00c59513 slli    a0,a1,0xc
19  0x000102e8(qr2_c+0x20): 0145d593 srli    a1,a1,0x14
20  0x000102ec(qr2_c+0x24): 00a5e533 or     a0,a1,a0
21  0x000102f0(qr2_c+0x28): 00c12403 lw      s0,12(sp)
22  0x000102f4(qr2_c+0x32): 01010113 addi    sp,sp,16
23  0x000102f8(qr2_c+0x36): 00008067 ret
24
25 BB@0x000102fc:0x00010320 executed 5758080 times:
26  0x000102fc(qr3_c+0x0): ff010113 addi    sp,sp,-16
27  0x00010300(qr3_c+0x4): 00812623 sw      s0,12(sp)
28  0x00010304(qr3_c+0x8): 01010413 addi    s0,sp,16
29  0x00010308(qr3_c+0x12): 00b545b3 xor    a1,a0,a1
30  0x0001030c(qr3_c+0x16): 00859513 slli    a0,a1,0x8
31  0x00010310(qr3_c+0x20): 0185d593 srli    a1,a1,0x18
32  0x00010314(qr3_c+0x24): 00a5e533 or     a0,a1,a0
33  0x00010318(qr3_c+0x28): 00c12403 lw      s0,12(sp)
34  0x0001031c(qr3_c+0x32): 01010113 addi    sp,sp,16
35  0x00010320(qr3_c+0x36): 00008067 ret
length: 4,975  Ln: 1  Col: 1  Sel: 0 | 0  Windows (CR LF)  UTF-8  INS
```

# 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

Name	Total Lines	Instrumente	Executed Lin	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%

```

test_c.c  test_custom.c  customChaCha20.c  riscv32.c  _start() at 0x100d0
// Emit code implementing exchange instruction
//
static void emitChaCha20(
    vmiProcessorP processor,
    vmiObjectP object,
    Uns32 instruction,
    Uns32 rotl
) {
    // extract instruction fields
    Uns32 rd = RD(instruction);
    Uns32 rs1 = RS1(instruction);
    Uns32 rs2 = RS2(instruction);

    vmiReg reg_rs1 = vmiGetExtReg(processor, &object->rs1);
    vmiReg reg_rs2 = vmiGetExtReg(processor, &object->rs2);
    vmiReg reg_tmp = vmiGetExtReg(processor, &object->tmp);
    // line executed 16 times
    vmiGetR(processor, RISC_V_REG_BITS, reg_rs1, object->riscvRegs[rs1]);
    vmiGetR(processor, RISC_V_REG_BITS, reg_rs2, object->riscvRegs[rs2]);
    vmiBinopRRR(32, vmi_XOR, reg_tmp, reg_rs1, reg_rs2, 0);
    vmiBinopRC(32, vmi_ROL, reg_tmp, reg_tmp, 0);

    vmiSetR(processor, RISC_V_REG_BITS, object->riscvRegs[rd], reg_tmp);
}

```



# Further tools for model developers (2)



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

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

CpuFManagerMulti (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 Instr= 677012570)
#-----
sw      : 19.87% : 134497608
lw      : 16.78% : 113575358
sw      : 13.95% : 98327451
addi    : 6.70% : 58668837
andi    : 4.37% : 33620233
bneq    : 4.25% : 29426103
bnez    : 3.73% : 25248055
add     : 3.73% : 25231546
slli    : 3.10% : 21020892
ret     : 2.49% : 16843028
jal     : 2.49% : 16843026
orli    : 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% : 4210850
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
arai    : 0.00% : 9
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	58,668,837	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	4,210,850	9.5ps/instruction (iss/cpu0)
bneq	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)
ret	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)
bge	0.00s	17	0.0ps/instruction (iss/cpu0)
neg	0.00s	16	0.0ps/instruction (iss/cpu0)
blt	0.00s	12	0.0ps/instruction (iss/cpu0)
arai	0.00s	9	0.0ps/instruction (iss/cpu0)
bgtz	0.00s	7	0.0ps/instruction (iss/cpu0)
bgez	0.00s	6	0.0ps/instruction (iss/cpu0)
sll	0.00s	4	0.0ps/instruction (iss/cpu0)
TOTAL	1.21s	677,012,570	

# 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 instructions 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

$dst = (src1 \oplus src2) \lll 16$

##### 2.1.2 chacha20qr2

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

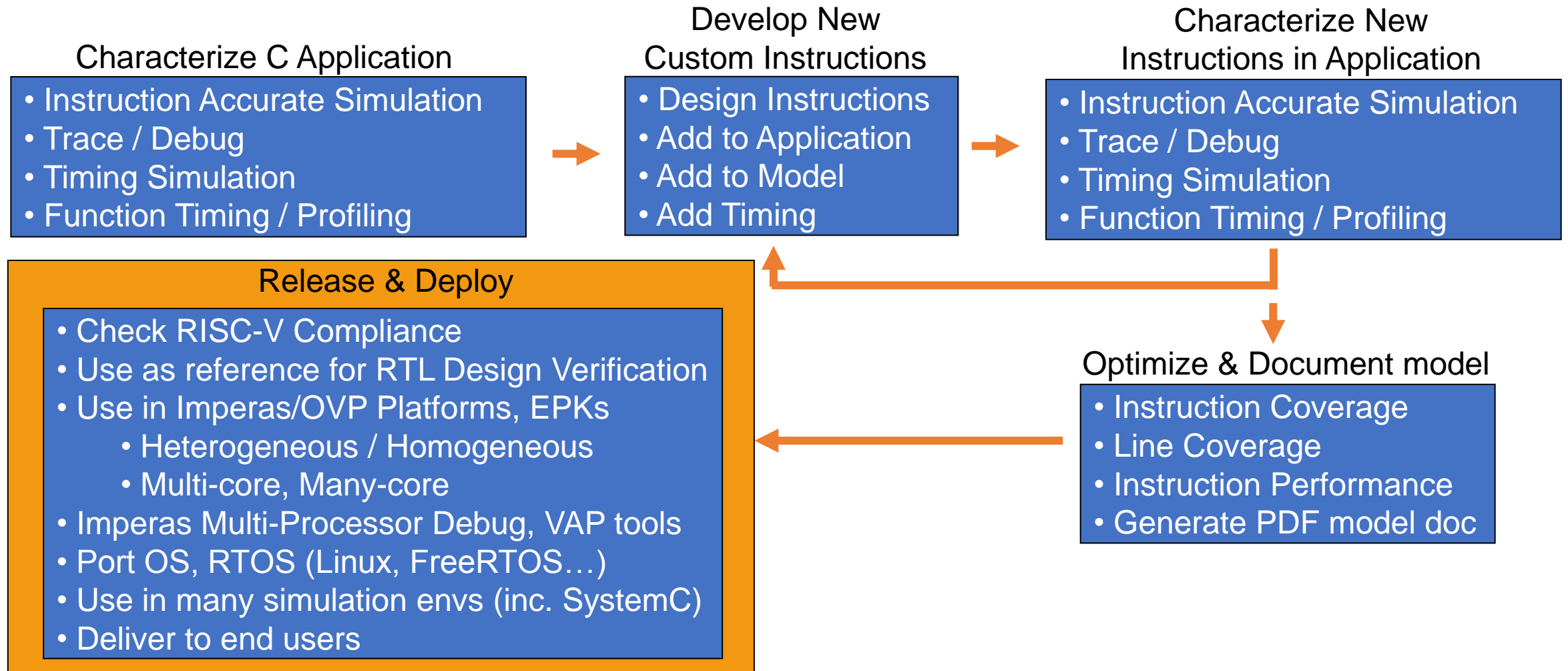
$dst = (src1 \oplus src2) \lll 12$

##### 2.1.3 chacha20qr3

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

$dst = (src1 \oplus src2) \lll 8$

# Flow to add new custom instructions



# Summary



- 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
- Flow for optimizing custom instructions in RISC-V processors is being used in real designs

# Thank You



- Visit [www.imperas.com/riscv](http://www.imperas.com/riscv) and [www.OVPworld.org/riscv](http://www.OVPworld.org/riscv) for more information
- RISC-V Foundation Compliance Suite, includes riscvOVPsim available:
  - <https://github.com/riscv/riscv-compliance>

Kevin McDermott  
VP Marketing, Imperas Software  
[kevinm@imperas.com](mailto:kevinm@imperas.com)