

RHEiMS-ISS RHEiMS-Instruction Set Simulator

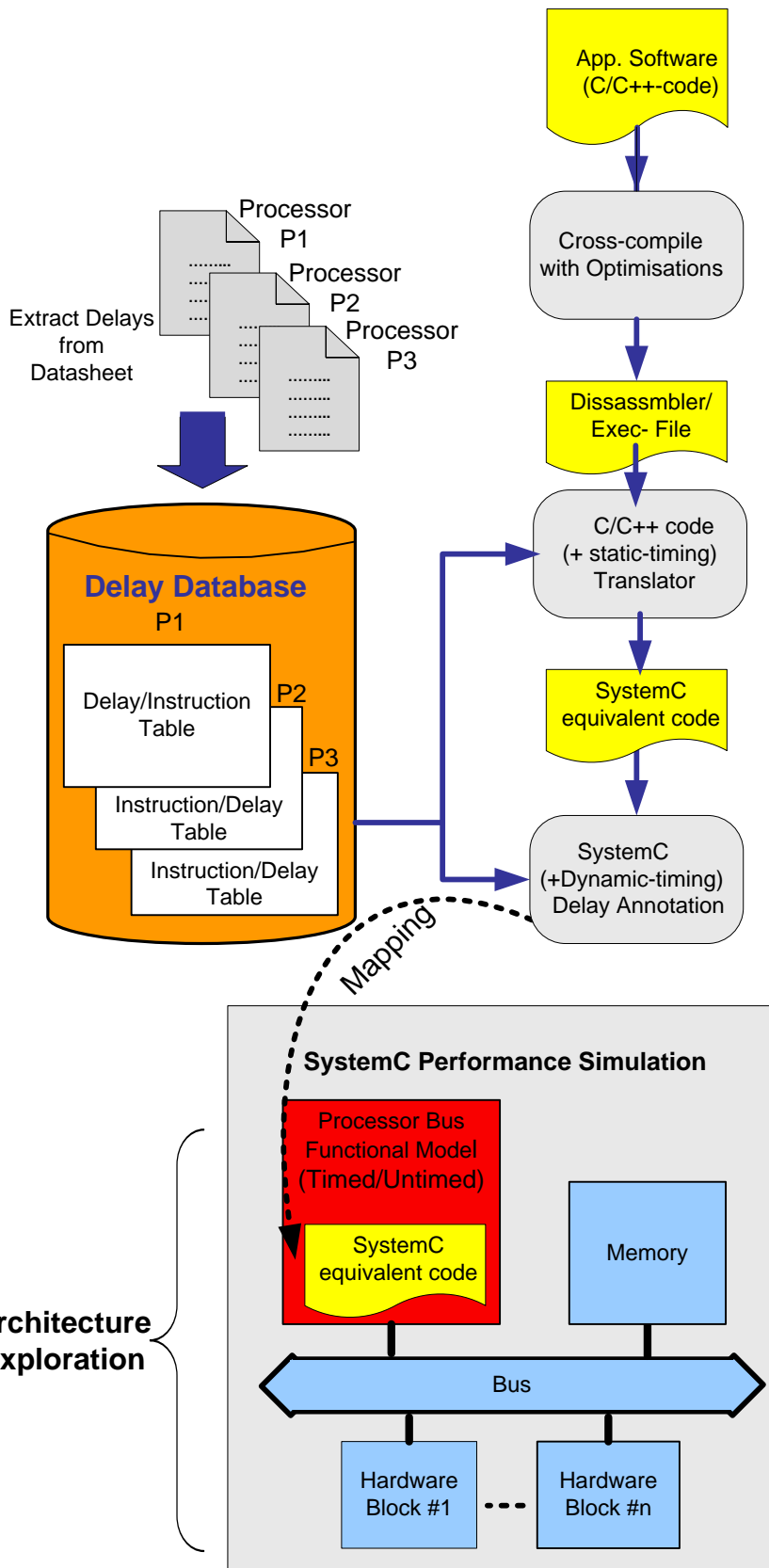
The RHEiMS-Instruction Set Simulator (ISS) solves one of the main challenges confronting programmers and designers of embedded systems--- **The rapid and accurate system-level determination and analysis of the code performance on a target processor.**

The RHEiMS-ISS simulator takes the application source-code (typically c-code), compiles the code taking into account any compiler optimisations, and then simulates the code on any one of a range of processors selected by the user. Pipelining and cache effects of the specific processor are incorporated into the simulation.

There are two modes of operation:

1. Algorithmic assessment mode : The RHEiMS-ISS executes and determines the cycle performance of the application code, but does not operate in the cycle mode of the processor. Even in this mode the simulator still takes into account all cache optimisations, pipelining and cache effects. The major benefits of this mode of operation is a simulation performance **3 orders of magnitude faster** than current commercial instruction set simulators with an accuracy approximately within 5% of a cycle-based instruction set simulation. **This mode is ideal for rapid software performance analysis requiring details on the number of cycles executed, register and memory references made and branches taken.**

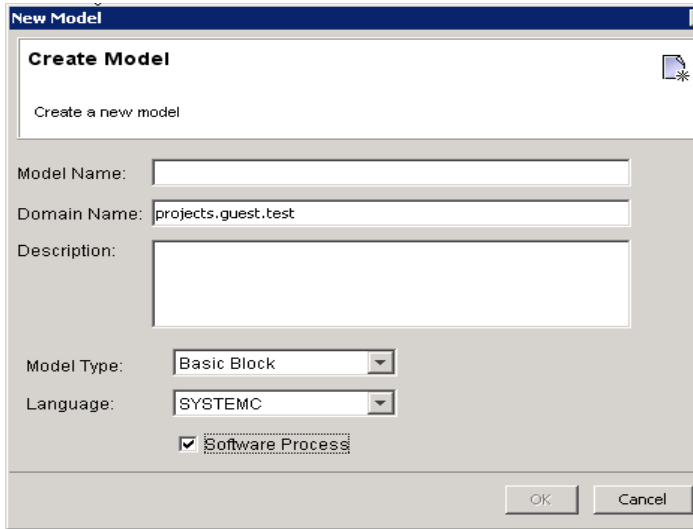
2. Hardware Co-simulation mode: The RHEiMS-ISS executes in a cycle regulated mode for use in system-level applications involving **hardware models with timing**. Simulation performance is approximately **1 order of magnitude faster** than conventional instruction set simulators.



A cross-compiler specific to the target processor is used to generate a binary file. Any level of optimisation provided by the compiler can be used. The Dissassembler/ Executable file identifies the original source instructions and associated assembly code. By accessing the appropriate processor instruction database, a Translator parses the latter file to produce a SystemC equivalent file, which is functionally identical but which has been annotated with static timing information. This file is further annotated with dynamic timing information which models the pipelining and cache effects of the target architecture. Finally, the SystemC file is compiled and executed in isolation or within the context of the SoC SystemC environment. Several processor architectures can be rapidly evaluated by simply retargeting a new processor each time.

The RHEiMS-ISS GUI

1. A software process is created to hold the source-files of the application code. A distinction between software and hardware processes is made for integration purposes with other design tools in the Neosera portfolio



New Model

Create a new model

Model Name:

Domain Name: projects.guest.test

Description:

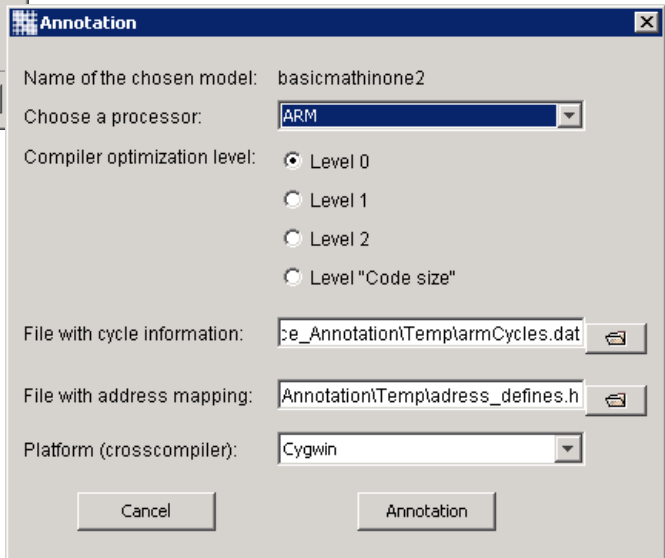
Model Type: Basic Block

Language: SYSTEMC

Software Process

OK Cancel

2. The Annotation menu is used to select the target processor, the cross-compiler, the optimisation level, and the source files containing the structure of the memory architecture and the instruction delay information.



Annotation

Name of the chosen model: basicmathnone2

Choose a processor: ARM

Compiler optimization level: Level 0
 Level 1
 Level 2
 Level "Code size"

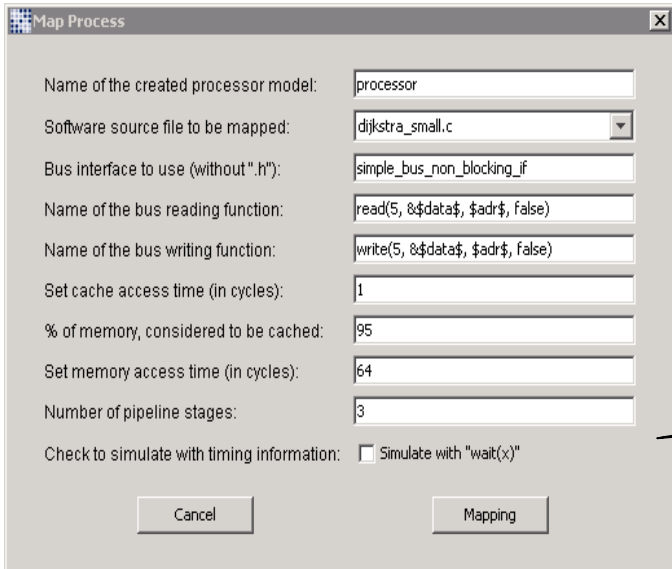
File with cycle information: ce_Annotation\TemplarmCycles.dat

File with address mapping: Annotation\Templaddress_defines.h

Platform (crosscompiler): Cygwin

Cancel Annotation

3. The map menu selects the application code that will compiled onto the target processor, the bus interface to memory and the invocation calls that read and write to it, and all pertinent information on the processor's pipelining and caching characteristics.



Map Process

Name of the created processor model: processor

Software source file to be mapped: dijksra_small.c

Bus interface to use (without ".h"): simple_bus_non_blocking_if

Name of the bus reading function: read(5, &data\$, \$adr\$, false)

Name of the bus writing function: write(5, &data\$, \$adr\$, false)

Set cache access time (in cycles): 1

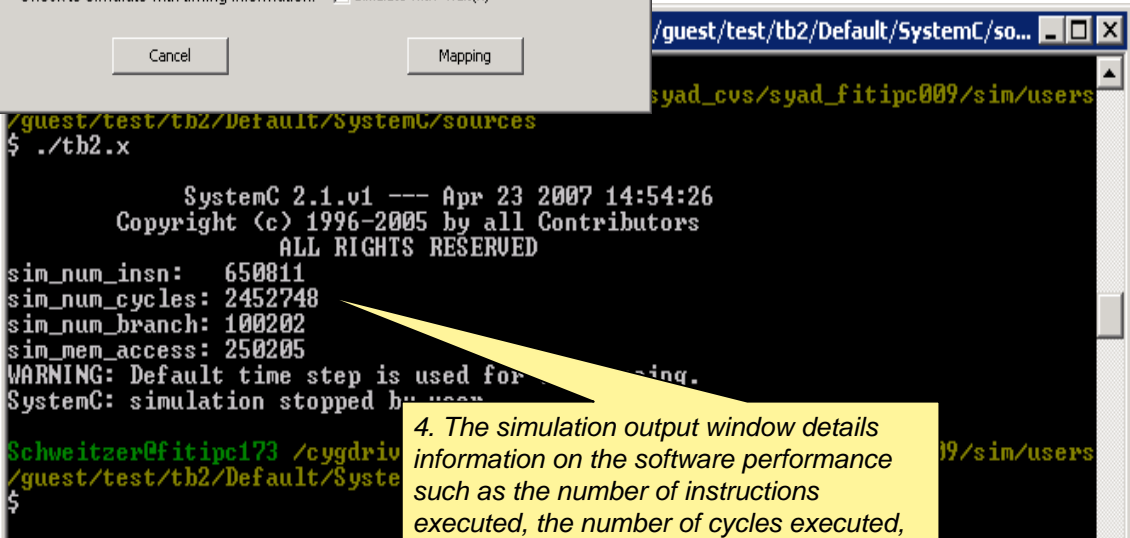
% of memory, considered to be cached: 95

Set memory access time (in cycles): 64

Number of pipeline stages: 3

Check to simulate with timing information: Simulate with "wait(x)"

Cancel Mapping

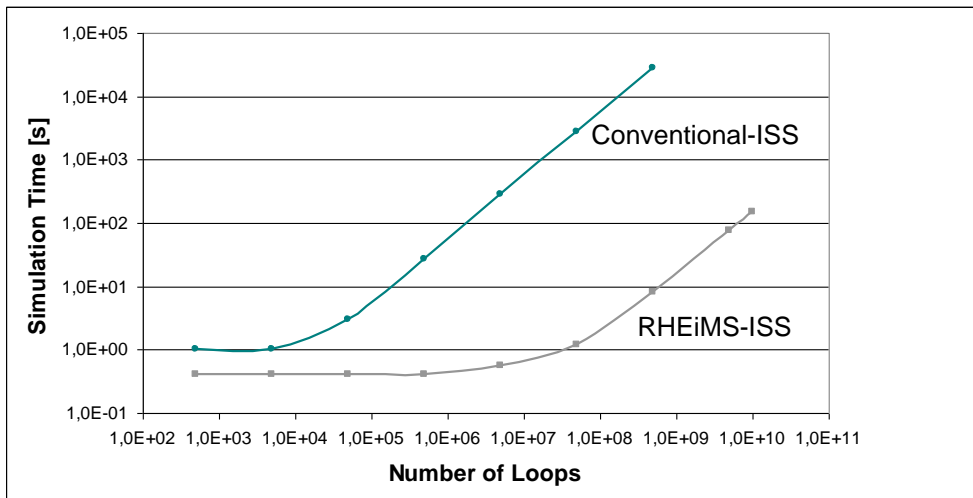


```
/guest/test/tb2/Default/SystemC/so...
syad_cvs/syad_fitipc009/sim/users
/guest/test/tb2/Default/SystemC/sources
$ ./tb2.x

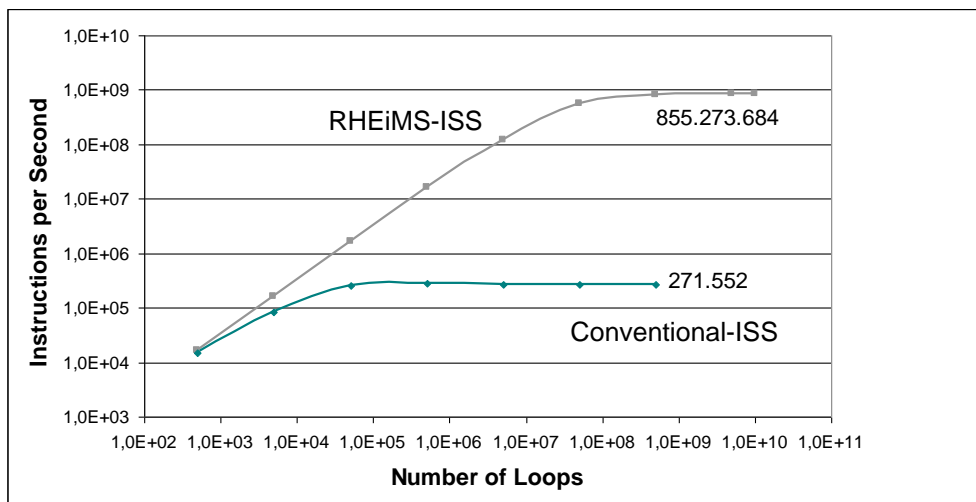
SystemC 2.1.v1 --- Apr 23 2007 14:54:26
Copyright (c) 1996-2005 by all Contributors
ALL RIGHTS RESERVED
sim_num_insn: 650811
sim_num_cycles: 2452748
sim_num_branch: 100202
sim_mem_access: 250205
WARNING: Default time step is used for timing.
SystemC: simulation stopped by user
Schweitzer@fitipc173 /cygdriv
/guest/test/tb2/Default/Syste
$
```

4. The simulation output window details information on the software performance such as the number of instructions executed, the number of cycles executed, the number of branches and memory accesses.

A Performance Comparison between RHEiMS-ISS and a conventional ISS (SimpleScalar/ ARM ISS: Univ of Wisconsin-Madison)



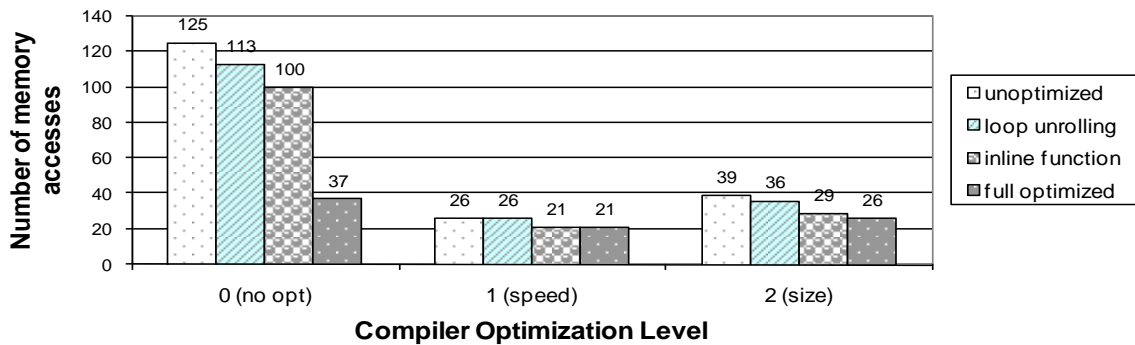
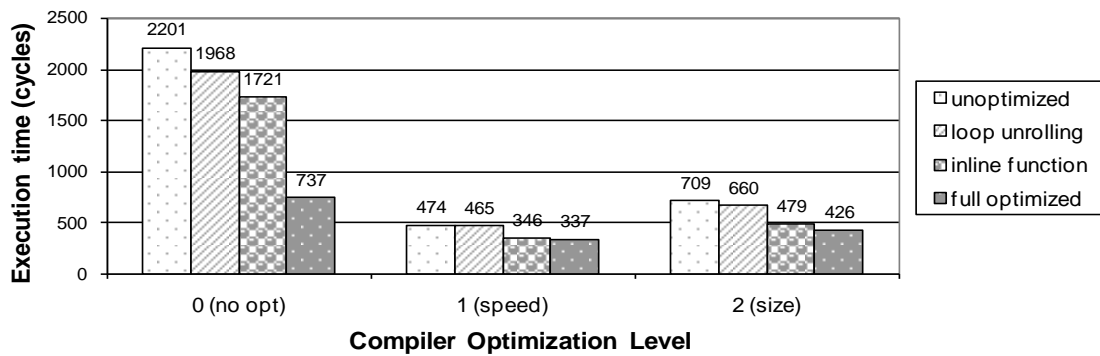
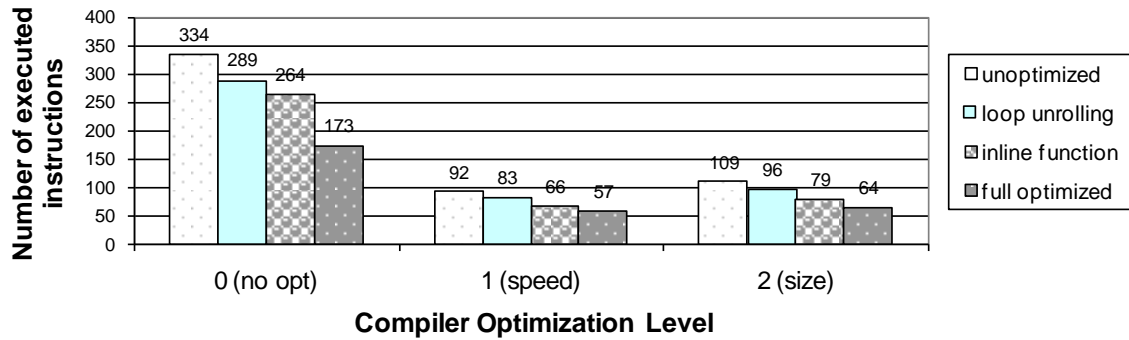
The above diagram shows the simulation time on the two instruction set simulators of a software benchmark with an internal execution loop that was varied. The number of loop iterations was varied between 10^2 to 10^{10} and RHEiMS operated in **algorithmic assessment** mode. RHEiMS-ISS operates approximately 3 orders of magnitude faster than a conventional-ISS and the difference between the results of the two simulators were within 5% (Deviation in no. of instructions 3.8%, Deviation no of Cycles 5.2%).



The above diagram illustrates the **Simulated instruction through-put** (No. Instructions simulated/Simulation time) of RHEiMS-ISS compared to a conventional ISS. RHEiMS consistently outperforms and saturates at a much higher level- 3.10^3 higher.

RHEiMS-ISS Value Proposition

The following diagrams illustrate some of the typical analysis regarding code optimisation that can be performed with RHEiMS-ISS. A section of code was optimised subject to loop unrolling, inline function insertion and full optimisation(all combined). Furthermore, each generated code segment was subjected to compiler optimisation with respect to level 1 optimisation(speed) and level 2 (memory size) respectively.



The results confirm that through the assistance of RHEiMS-ISS analysis, it was possible to determine directly the effects of various optimisation strategies on the execution code. It was possible to reduce the no. of instructions executed and execution time by 81% and the number of memory references by 80%.

Conclusion, RHEiMS-ISS outperforms existing instruction set simulators by delivering the following features:

- Execution speed is 3 orders of magnitude faster than conventional ISS.
- All compiler optimisations, pipeline and cache memory effects are accommodated in the analysis.
- The GUI is simple and straightforward.