SimCore/Alpha Functional Simulator
Version 1.0: Simple and Readable Alpha Processor Simulator

Kenji KISE  kis@is.uec.ac.jp

Graduate School of Information Systems,
University of Electro-Communications
Outline

- Introduction
- SimCore/Alpha overview
- SimCore/Alpha internals
- SimCore/Alpha practical use
- Summary
Introduction

- Various processor simulators are used for research and education activities.
- Famous **SimpleScalar Tool Set** is not a code that can easily be modified.

- **We have developed** a processor simulator **SimCore/Alpha Functional Simulator Version 1.0** (**SimCore/Alpha Version 1.0** in short).
  - Its design policy is to keep the source code readable and simple.
  - Software architecture of **SimCore/Alpha** is explained by referring to its source code.
  - As **SimCore/Alpha** practical use, we present the ideal instruction-level parallelism of SPEC benchmarks measured with a modified version of **SimCore/Alpha**.
SimCore/Alpha overview

This section overviews the SimCore/Alpha. We focus the design policy, simulation platfromes, execution image file format and simulation speed.
SimCore/Alpha Version 1.0 design policy

- **Target applications**
  - SPEC CINT95 and CINT2000.

- **Simple and readable (enjoyable and easy to read) code**
  - No global variable
  - No goto statement
  - No conditional compilation
  - The source code is only about 2,800 lines in C++.

- **It offers another choice**
  - A processor simulator is an important tool.
  - It is advantageous to choose a suitable tool from many choices.
  - SimCore/Alpha offers another choice.

- **Functional simulator, but**
  - Although only the capability of a functional simulator is given, the code is written considering the extension to the out-of-order processor simulator.
SimCore/Alpha Version 1.0 is a functional simulator.

- Functionally equivalent to the *sim-safe* or *sim-fast* of SimpleScalar Tool Set.
- Functional Simulator is available for
  - Simulator verification
  - Evaluation of instruction mix
  - Evaluation of branch predictors
  - Evaluation of memory systems
  - Evaluation of ideal instruction-level parallelism.

Figure 1. Typical flow in microprocessor design process. Interaction between the process steps refines the performance model throughout the process. (IEEE Computer, February 2002, p 30)
SimCore/Alpha Version 1.0 software components

<table>
<thead>
<tr>
<th>Line</th>
<th>Word</th>
<th>Byte</th>
<th>Filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>340</td>
<td>2968</td>
<td>17992</td>
<td>COPYING</td>
</tr>
<tr>
<td>73</td>
<td>193</td>
<td>1556</td>
<td>Makefile</td>
</tr>
<tr>
<td>356</td>
<td>1570</td>
<td>13174</td>
<td>README.txt</td>
</tr>
<tr>
<td>871</td>
<td>2948</td>
<td>21347</td>
<td>arithmetic.cc</td>
</tr>
<tr>
<td>73</td>
<td>199</td>
<td>1882</td>
<td>chip.cc</td>
</tr>
<tr>
<td>201</td>
<td>626</td>
<td>5286</td>
<td>debug.cc</td>
</tr>
<tr>
<td>368</td>
<td>1173</td>
<td>11142</td>
<td>define.h</td>
</tr>
<tr>
<td>271</td>
<td>811</td>
<td>7751</td>
<td>etc.cc</td>
</tr>
<tr>
<td>187</td>
<td>647</td>
<td>4505</td>
<td>instruction.cc</td>
</tr>
<tr>
<td>179</td>
<td>491</td>
<td>4412</td>
<td>memory.cc</td>
</tr>
<tr>
<td>15</td>
<td>49</td>
<td>417</td>
<td>sim.cc</td>
</tr>
<tr>
<td>562</td>
<td>1234</td>
<td>14871</td>
<td>syscall.cc</td>
</tr>
<tr>
<td>3496</td>
<td>12909</td>
<td>104335</td>
<td>total</td>
</tr>
<tr>
<td>2727</td>
<td>8178</td>
<td>71613</td>
<td>total *.cc *.h</td>
</tr>
</tbody>
</table>

- **Licensing**
  - GNU General Public License Version 2
- The source code and include file consists of only 2,727 lines.
- arithmetic.cc : 871 lines for arithmetic and logic etc.
- syscall.cc : 562 lines for system call implementation.
- Core source code is about 1,300 lines.
SimCore/Alpha plathomes

- SimCore/Alpha Version 1.0 plathomes:
  - Intel Pentium 4, RedHat Linux 7.3, gcc version 2.96
  - Intel Pentium III, RedHat Linux 6.2, gcc

- SimCore/Alpha Version 1.4 plathomes:
  - Intel Pentium 4, RedHat Linux 7.3, gcc version 2.96
  - Intel Pentium 4, Windows XP Cygwin version 2.340, gcc
  - DEC Alpha 21264, Tru64 UNIX V5.1, gcc version 2.95.2
  - AMD Opteron, Turbo Linux 8, gcc version 3.2.2
  - Intel Pentium 4, RedHat Linux 7.3, Intel C++ Compiler 7.1

Note that SimCore/Alpha uses no conditional compilation.
SimCore/Alpha verification

- In the development phase of SimCore/Alpha, compatibility with SimpleScalar was carefully confirmed.

- When SimCore/Alpha executed one instruction, we confirmed that the two architecture states (a program counter, 32 integer registers, 32 floating point registers) were identical.

- SimCore/Alpha uses no global variable. Two or more simulation images can easily be generated in one process.
  - Any bug of simulators under development is discovered at an early stage.
Execution image file as SimCore/Alpha Input

- SimCore/Alpha reads not an Alpha binary but an execution image file in original format.
- This simple original format makes the knowledge of executables such as ELF and COFF unnecessary.

- **In the first part**, values are assigned to the registers.
- **In the second part**, values are assigned to the memory.

- This image file is created from an Alpha binary by SimCore-Loader.

Sample execution image file.

```plaintext
/* SimCore 1.0 Image File */
/*** Registers ***/
/@reg 16 0000000000000003
/@reg 17 000000011ff97008
/@reg 29 0000000140023e90
/@reg 30 000000011ff97000
/@pc 32 0000000120007d80

/*** Memory ***/
@11ff97000 00000003
@11ff97008 1ff97138
@11ff9700c 00000001
```
SimCore/Alpha simulation speed

- We ran the **8 SPEC CINT95** benchmark programs and calculated the average simulation speed.

- **Pentium 4 Xeon 2.8GHz dual processor PC** with 4GB memory running Red Hat Linux 7.3.

- **Configurations:**
  - **sim-fast** from SimplaScalar Version 3.0c
    - Compiled with GCC with –O2 optimization flag
  - **SimCore/Alpha Version 1.0**: first release of SimCore/Alpha.
    - Compiled with GCC with –O2 optimization flag
  - **SimCore/Alpha Version 1.4**: optimized version of SimCore/Alpha.
    - Compiled with GCC with –O3 optimization flag
    - Compiled with **Intel C++ Compiler** with –O2 –ipo optimization flag
SimCore/Alpha simulation speed

- SimCore/Alpha Version 1.0 is 6 times slower than sim-fast (SimpleScalar).
- SimCore/Alpha Version 1.4 is 30% faster than sim-fast.
- Now SimCore/Alpha is simple, readable and fast!
SimCore/Alpha internals

In this section, in order to show the high readability of the source code, the internal structure of SimCore/Alpha is explained showing actual C++ code (not pseudo code).
SimCore/Alpha main function

- Easy to understand and modify.
- No global variable

Sample command

SimCore/Alpha -e10000 -v100 aout.txt

option          program name
**Class simple_chip constructor**

```
TJNQMF@DIJQTJNQMF@DIJQ	DIBS QSPH
DIBSPQU

TDOFXTZTUFN@DPOGJH	QSPH
PQU

FOFXFWBMVBUJPO@SFTVMU
BTOFXBSDIJUFDUVSF@TUBUF	TD
F

NFNOFXNFNPSZ@TZTUFN	TD
F

EFC OFXEFCVH	BT
NFN
TD
F

TZTOFXTZTUFN@NBOBHFS	BT
NFN
TD
F

QOFXJOTUSVDUJPO	BT
NFN
TZT
TD
F

```

simple_chip creates seven objects.

```
instruction p

system_manager sys

architecture_state as memory_system mem

system_config sc evaluation_result e

debug deb
```
Class data_t definition and methods

- The calculation results are stored in register file or memory. These are defined as the collection of class data_t objects.

- Function **st** is used to store a data value into a data_t type object. Function **ld** is used to read a data value. Function **init** is used to generate a new object.
The architecture state of **Alpha-AXP** consists of a program counter, 32 integer registers and 32 floating point registers.

The **architecture state** is defined as a collection of the `data_t` type objects.
SimCore/Alpha main function again

Easy to understand and modify.
No global variable

Sample command

SimCore -e10000 -v100 aout.txt

option program name
One instruction is executed by calling eight functions.

They are corresponding to seven pipeline stages and then calling the eighth function of WriteBack in order.
The code of an instruction fetch stage is shown.

This function **loads 4 bytes instruction** from the memory specified by the program counter, and stores it in the **variable ir**.

Then, the address of the next instruction is stored in variable Npc.
Instruction fetch stage sets some variables
Slot (or decode) stage

The code of a decode stage is shown.

The variables are decoded using the fetched instruction code.

The description of Verilog-HDL is similar to the code.

Therefore, part of the C++ code can be reused for Verilog-HDL.
Slot (or decode) stage sets some variables
The code of an issue stage is shown.

Here, an immediate Imm is generated according to the type of instruction.
Issue stage sets variable \textit{Imm}
The code of a register read stage is shown.

- The values of Rav and Rbv are each selected from an immediate value, a floating point register and an integer register.
Register read stage sets some variables
The code of an execution stage is shown.

Three data values are updated in the execution stage.

The arithmetic and logic instruction calculates the value of $Rcv$ by considering $Rav$ and $Rbv$ as input.

A load/store instruction calculates the memory reference $address \text{ Adr}$.

A branch instruction calculates the branch $target \text{ address Tpc}$. 
Memory access stage

The code of a memory access stage is shown.

- In the store instruction, the value of $Rav$ is stored in memory.
- In the load instruction, the loaded value is saved at $Rcv$. 
The code of a writeback stage is shown.

In the instruction which generates a result, **Rcv is stored in a register file.**

The instruction completes execution.

An execute_pal function is called when the instruction currently executed is PAL(Privileged Architecture Library) code.
SimCore/Alpha Version 1.0 design policy again

- Target applications
  - SPEC CINT95 and CINT2000.
- Simple and readable (enjoyable and easy to read) code
  - No global variable
  - No goto statement
  - No conditional compilation
  - The source code is only about 2,800 lines in C++.
- It offers another choice
  - A processor simulator is an important tool.
  - It is advantageous to choose a suitable tool from many choices.
  - SimCore/Alpha offers another choice.
- Functional simulator, but
  - Although only the capability of a functional simulator is given, the code is written considering the extension to the out-of-order processor simulator.
SimCore/Alpha practical use

This section gives an example of the SimCore/Alpha practical use. SimCore/Alpha is modified to measure ideal instruction-level parallelism. The parallelism is acquired only after considering data dependency as a restriction.
To measure ideal instruction-level parallelism

- Ideal instruction-level parallelism or **Oracle IPC** is the parallelism which is acquired only after considering data dependency as a restriction.
  - **No control dependency**
  - **No resource conflict**
- In order to measure Oracle IPC, the value (this is called the **rank**) equivalent to the height of the data flow graph is calculated.

**Oracle IPC** = $\frac{6}{4} = 1.5$
Calculation of the rank for each instruction type

- Class data_t is modified to store the rank.
- **Arithmetic and logic instruction**, the rank of Rcv is obtained by adding the operation latency to the maximum of the rank of Rav and Rbv.
- **Load instruction**, the rank is calculated by adding the memory reference latency and the address computation latency to the rank of Rbv.
- **Store instruction**, the maximum of the Rav written in memory and the rank obtained by address computation is the rank of the data.

Modified class data_t
Oracle IPC measured by the modified SimCore/Alpha
SimCore/Alpha development plan

SimCore/Alpha Version 1.0 is a functional simulator. We have the plan to construct cycle-accurate performance simulator SimCore/Alpha Real SuperScalar modeling various out-of-order superscalar processors.
Summary

- Processor simulator **SimCore/Alpha Version 1.0** was developed for processor architecture research and processor education.
- To show the high **readability** of the code, the software architecture of SimCore/Alpha was explained using the actual C++ code.
- As an example of the practical use of SimCore/Alpha, the evaluation method of **Oracle-IPC** was explained.

- We have the plan to construct **cycle-accurate performance simulators** modeling various out-of-order superscalar processors.
- We are implementing SimCore/Alpha of the **Verilog-HDL version**, which works on an FPGA board.
SimCore/Alpha is available!

- **SimCore Homepage**
  - The source code of SimCore/Alpha is **downloadable** from the following URL.
  - [http://www.yuba.is.uec.ac.jp/~kis/SimCore/](http://www.yuba.is.uec.ac.jp/~kis/SimCore/)

- **SimCore/Alpha document**

Note that SimCore/Alpha was called SimAlpha until 2003-09.