First Steps Simics/GEMS on Ubuntu 7.10

Author: Aleksandar Vitorovic, savitor@sbb.co.yu,
student of Faculty of Electrical Engineering,
University of Belgrade, Serbia

1 Introduction

This tutorial explains basic steps for users to get familiar with GEMS/Simics environment. It is idiot-proof tutorial. Basic think you should know to start simulator is showed. In every fork I decided to choose one option and explain it in detail. For further details and other options you should consult appropriate documentation.

In chapter 2 is detailed installation process for GEMS 2.1 and Simics 3.0.30 on Ubuntu 7.10 x86.

In chapter 3 is detailed installation process for Simics 2.2.19 on Ubuntu 7.10 x86.

In chapter 4 there is different manuals for Frequently used Commands.

In chapter 5 we explain how to add benchmark to your target machine and how to start that.

In chapter 6 there is list of used references.

2 GEMS 2.1/Simics 3.0.30 Installation

It is possible to install GEMS 2.1 based on Simics 3.0.30 on Ubuntu 7.10 x86 with GCC 4.1.3, although GEMS manufacturers internally work with GCC 3.4.6. This tutorial specifies whole installation process on this concrete case. It is supposed that Ubuntu didn't install any updates. Maybe this will work for updated Ubuntu, but there is no guarantee.

GEMS Supports SPARC-based targets out-of-the-box (different from Solaris, which is an OS that runs on SPARC AND x86-based systems). GEMS is intended to support the UltraSPARC III+'s ISA (Serengeti platform), which is a superset of SPARCv9. Opal is extremely tied to this ISA implementation. Ruby is less coupled to the SPARC ISA, and can be used with x86-based targets with some modification (there is a patch for GEMS 1.2 in the downloads directory). Sarek is Serengeti with Solaris installed. This is reason why you must start Simics with sarek target.

2.1 Obtaining software

First we will explain obtaining GEMS. GEMS is available under GPL licence. First you will register on:

https://www.cs.wisc.edu/gems/register.cgi

and then you can download GEMS 2.1 from:

https://www.cs.wisc.edu/gems/download.html

The procedure for obtaining Simics is little bit complicated. Here is descripted procedure for *Academic* licence, and it is free of charge. If you plan to use Simics in commercial projects, you should pay licence. You should firstly register on

https://www.simics.net/register/register.php

and wait for response from Simics in form on user name and password. Now you can download `Simics 3.0` for `Linux on x86` from

https://www.simics.net/evaluation/download.php

Now you should download licence. You can obtain licence directly from computer where you want to install Simics. Follow instructions from:

https://www.simics.net/evaluation/academic.php

After couple of minutes, you can download licence file from:

https://www.simics.net/evaluation/form_dl.php

2.2 Preparing

1. First we will install some prerequistics

   ```
   sudo apt-get install build-essential
   sudo apt-get install bison
   sudo apt-get install flex
   ```

   Download from

   `http://archive.ubuntu.com/ubuntu/pool/main/z/zlib/`

   packages

   `zlib1g and zlib1g-dev`

   and install them from command line:

   ```
   sudo dpkg --install zlib1g_1.2.3.dfsg-7ubuntu1_i386.deb
   sudo dpkg --install zlib1g-dev_1.2.3.dfsg-7ubuntu1_i386.deb
   ```

2. Then we add lines to $HOME`/.profile`

   ```
   export GEMS=$HOME/gems
   export SIMICS_INSTALL=$GEMS/simics-3.0.30
   export SIMICS_EXTRA_LIB=./modules
   export PYTHONPATH=./modules
   ```

   Now restart your machine.

3. Now install GEMS:

   ```
   mkdir $GEMS
   cd $GEMS
   ```

   Copy `gems-release21.tar.gz` into a chosen $GEMS directory and follow instructions:

```
tar xvzf gems-release21.tar.gz
cp -r $GEMS/gems-21/* $GEMS
rm -r $GEMS/gems-21
```

4. Copy `simics-pal-3.0.30-linux.tar` to $GEMS directory. Install Simics 3.0.30:

```
tar xvf $GEMS/simics-pal-3.0.30-linux.tar
cd $GEMS
cd simics-3.0-install
./install-simics.sh
```

Follow the installation instructions provided by Virtutech. When wizard ask you for install directory, write your $GEMS path.

You may need to copy your license file to $GEMS/simics-3.0.30/licenses.

5. Instantiate a Simics workspace:

```
cd $GEMS
mkdir simics_3_workspace
cd $SIMICS_INSTALL/bin
./workspace-setup $GEMS/simics_3_workspace
```

There will be directive how to build devices which you should follow.

6. Edit `$GEMS/scripts/makesymlinks.sh` to include the correct path to `$SIMICS_INSTALL`

```
cd $GEMS/scripts/
vi makesymlinks.sh
echo "Making symlink for import directory.."
ln -s $SIMICS_INSTALL/import import
```

7. Run `makesymlinks.sh` from your Simics workspace.

```
cd $GEMS/simics_3_workspace
../scripts/makesymlinks.sh
```

8. Link your Simics workspace to `$GEMS/simics`.

```
cd $GEMS/
ln -s simics_3_workspace simics
```

9. Edit GEMS Makefiles:

- `$GEMS/common/Makefile.simics_version`
  - set the version string to `3.0`
- `$GEMS/common/Makefile.common`
  - Modify the CC for your chosen compiler (in our case just in `x86-linux` department)
  - Set value to `HOST_TYPE` directly:
    `HOST_TYPE=x86-linux`
    because script `calc_host.sh` is not working properly
  - After line `ifeq ($(SIMICS_VERSION),3.0)` modify lines

```
SIMICS_ROOT := $(GEMS_ROOT)/simics
```

```
SIMICS_INCLUDE_ROOT := $(SIMICS_INSTALL)/src/include
```

- `$GEMS/ruby/module/Makefile`
  - If you are not using x86-platform, modify `CC_version`
  - Set value to `HOST_TYPE` directly:
    `HOST_TYPE=x86-linux`
    because script `calc_host.sh` is not working properly
  - Use the Simics-3.0-appropriate definition of `GEMS_ROOT` – comment line below line `For Simics 2.X`, and decomment line below `For Simics 3.0`
- `$GEMS/opal/module/Makefile`
  - If you are not using x86-platform, modify `CC_version`
  - Set value to `HOST_TYPE` directly:
    `HOST_TYPE=x86-linux`
    because script `calc_host.sh` is not working properly
  - Use the Simics-3.0-appropriate definition of `GEMS_ROOT` – comment line below line `For Simics 2.X`, and decomment line below `For Simics 3.0`
- `$GEMS/tourmaline/module/Makefile`
  - If you are not using x86-platform, modify `CC_version`
  - Set value to `HOST_TYPE` directly:
    `HOST_TYPE=x86-linux`
    because script `calc_host.sh` is not working properly

2.3 Compilling

In this section, we will compile Ruby and Opal and run a simulation manually. This is meant to illustrate the steps necessary to get Ruby and Opal loaded into Simics. For actual research and result gathering, it is highly recommended to script the running of the simulation. The Multifacet group uses extensive scripting, as well as the use of the Condor batch-computing system, to automate the running and data collection of simulation. Some examples of our scripts are provided in the *$GEMS/gen-scripts* directory. See the README file located in that directory, or see this short write-up on running a simulation using ["gen-scripts"].

Compile Ruby

To compile Ruby, specify the SLICC protocol specification, and the destination directory which will be created in $GEMS/simics/home . This allows multiple protocols to co-exist in the Simics tree. This is accomplished by the *movemodule* target in the Ruby Makefile. If desired, you can edit this target such that the ruby module remains in $GEMS/simics/x86-linux/lib.

```
cd $GEMS/ruby
make PROTOCOL=MOSI_SMP_bcast DESTINATION=MOSI_SMP_bcast
```

There is high probablity to see some error about `extra qualification`. Go to specified .h file and delete class name in front of double colon. This is compiler issue, since there are GCC 4.x on Ubuntu 7.10, but GEMS was made for GCC 3.4.6.

When the above command is executed, SLICC is first compiled and run on the MOSI_SMP_bcast specification files. SLICC generates C++ code that is placed in the *ruby/generated/MOSI_SMP_bcast* directory.

MOSI_SMP_bcast is a snooping, broadcast-based cache coherence protocol for a SMP system. By default, Ruby is configured to work with this protocol. Other protocols may require you to change configuration options.

Compile Opal

Compiling Opal is optional Opal is independent of the chosen SLICC protocol, however the DESTINATION directory must be specified. Because of compiler issue you must modify files `$GEMS/opal/system/cache.h` and `$GEMS/opal/system/cache.C` so that all method definition from class `generic_cache_template` in `cache.C` should migrate to `cache.h`.

```
cd $GEMS/opal
make module DESTINATION=MOSI_SMP_bcast
```

2.4 Make Checkpoint

It is necessary to make Simics checkpoint for loading ruby, opal and tourmaline modules. It should be Sarek checkpoints. Also, checkpoints must be made in Simics 2.2.19, and they are forward compatible. Unfortunately, due to licensing issues, Multifacet cannot release their checkpoint files. Now you should install Simics 2.2.19:

```
cd $GEMS
tar xvzf simics-2.2.19-x86-linux.tar.gz
```

It is possible to install Solaris 8 and 9 directly on the simulated machine in Simics. We will choose Solaris 9, which can be obtained from Sun's web-site at
`http://www.sun.com/software/solaris/binaries/get.html`
in the form of CD ISO images. We will download

`sol-9-905hw-ga-sparc-v1-iso.zip` and
`sol-9-905hw-ga-sparc-v2-iso.zip`

You should now unzip these files, for example with commands:

```
unzip sol-9-905hw-ga-sparc-v1-iso.zip and
unzip sol-9-905hw-ga-sparc-v2-iso.zip
```

To simplify the installation process, several scripts are supplied with the Simics distribution for the sarek machine: `sol9-cd-install-1.simics` and `sol9-cd-install-2.simics`. This section describes how to install Solaris using the command-line version Simics.

Before we start, you should type in command prompt:

```
cd $GEMS
```

```
mkdir checkpoints
```

1. In directory `$GEMS/simics-2.2.19/home/sarek` edit files `sol9-cd-install-1.simics` and `sol9-cd-install-2.simics`. Set the path to the CD image in the simics script.

   ```
   @cdrom_path = "sol-8-u7-sparc-v1.iso"
   ```

2. Start the first installation script:

   ```
   cd $GEMS/simics-2.2.19/home/sarek/
   $ ./simics -x sol9-cd-install-1.simics
   ```

and wait for it to complete. This will take several hours, from 2 to 10 hours depending on the performance of the host machine.

3. When the script stops, installation from the first CD is finished, and Solaris has tried to reboot the system.
4. Now run the second script in the same way as the first, this will also take several hours to complete.

   ```
   $ ./simics -x sol9-cd-install-2.simics
   ```

5. When the second script has stopped the installation is ready. The newly created disk image has the following file name: `sarek-sol9-install.disk`.
6. To boot a machine with the newly installed operating system, use the scripts named `sol9-run.simics`.

   ```
   ./simics -x sol9-run.simics
   ```

7. The default `sol9-run.simics` scripts create a single-processor machine. If you want to run multiprocessors or modify the machine setup in some other way, you can have two options. One is to look in the other setup scripts (`sarek-1p.simics`, `sarek-2p.simics`, etc.) and copy the machine setup done before the call to `sarek-common.simics` to the `sol9-run.simics`. The other way is to modify the machine script files to call `sol9-run.simics` instead of `sarek-common.simics`.
8. Now you can make checkpoint in Simics prompt with

```
simics>continue
```

When system console of target machine finish work, and # appear, it's time to:

```
CTRL+C
```

In the following line change $GEMS with your directory because Simics does know nothing about environment variables:

```
simics>write-configuration $GEMS/checkpoints/first-sarek.check
simics>quit
```

2.5 Set Environment and Start Simics

The Makefile creates a link to the Simics executable in the$GEMS/simics/home/<DESTINATION> directory. This is what we typically use rather than the GUI Simics executable.

Load Checkpoint

We require that you create a Simics checkpoint before loading Ruby or Opal. Booting Solaris, or your chosen operating system, is not tested and would take a long time. Start Simics from specified directory, because ruby and opal modules are installed here:

```
cd $GEMS/simics/home/MOSI_SMP_bcast
./simics -stall -c $GEMS/checkpoints/first-sarek.check
```

Objects in checkpoints are instantiated from classes that are defined by runtime-loadable modules.

Note that the `-stall` *option should be placed in the Simics command line arguments to ensure forward-compatibility. The -c option means that checkpoint file is loaded.*

Configure Simics for Ruby/Opal

These steps are **very important**. Omitting these steps will result in incorrect execution with little warning. These commands instruct Simics to deliver all instruction fetches to Ruby/Opal, and disable the internal cache. Setting `cpu-switch-time` to 1 is optional but highly recommended for accurate simulations.

```
simics> instruction-fetch-mode instruction-fetch-trace
simics> istc-disable
simics> dstc-disable
simics> cpu-switch-time 1
```

Instruction fetches are not sent to the memory hierarchy by default. This is reason why we set instruction-fetch-mode.

Note: The instructions-fetch-mode is a session attribute and Simics does not save session attributes in a checkpoint. Therefor it is necessary to reset this attribute or rerun the command each time a checkpoint is loaded. You can also add the attribute to each CPU object in the the checkpoint file by hand.

Load Ruby/Opal modules

At this point, Ruby can be loaded. Loading Opal is optional, and if omitted, will result in an in-order processor. Ruby and Opal automatically detect the presence of each other and cooperate accordingly.

```
simics> load-module ruby
simics> load-module opal
```

Configure Ruby

The default options are specified in `$GEMS/ruby/config/rubyconfig.defaults`, however any of these can be overridden at runtime. Further, the number of processors *must* be specified at runtime as shown below.

```
simics> ruby0.setparam g_NUM_PROCESSORS 1
```

The default memory size, set in rubyconfig.defaults, is 4GB. This can also be set at runtime to match your checkpoint.

```
simics> ruby0.setparam g_MEMORY_SIZE_BYTES 4294967296
```

Initialize Ruby and Opal

After setting runtime options, Ruby and Opal must be initialized.

```
simics> ruby0.init
simics> opal0.init
```

Opal also requires its output file be set

```
simics> opal0.sim-start "results.opal"
```

Start Simulation

If Opal has been loaded, Opal must start Simics.

```
simics> opal0.sim-step 10000000000
```

The argument to sim-step is merely a large number to run the simulation indefinitely until instructed otherwise.

If only Ruby has been loaded, we instruct Simics to start the simulation with the following command:

```
simics> c
```

Print Results

Press CTRL+C to interrupt the simulation

```
simics> ruby0.dump-stats
simics> opal0.listparam
simics> opal0.stats
```

3 Simics 2.2.19 Installation

It is possible to install GEMS 2.1 based on Simics 2.2.19 on Ubuntu 7.10 x86 with GCC 4.1.3, although GEMS manufacturers internally work with GCC 3.4.6. It is supposed that Ubuntu didn't install any updates. Maybe this will work for updated Ubuntu, but there is no guarantee.

It is assumed that you already installed GEMS 2.1, steps to do so is described in chapter 2.

3.1 Obtaining software

Here is descripted procedure for *Academic* licence, and it is free of charge. If you plan to use Simics in commercial projects, you should pay licence. You should firstly register on

https://www.simics.net/register/register.php

and wait for response from Simics in form on user name and password. Now you can download `Simics 2.2` for `Linux on x86` from

https://www.simics.net/evaluation/download.php

Now you should download licence. You can obtain licence directly from computer where you want to install Simics. Follow instructions from:

https://www.simics.net/evaluation/academic.php

After couple of minutes, you can download licence file from:
https://www.simics.net/evaluation/form_dl.php

Note: If you had already registered, than you can use your existing username and password. If you had already got licence to Simics 3.0.30, you can use same licence file.

3.2 Preparing

1. First we will install some prerequistics

   ```
   sudo apt-get install build-essential
   sudo apt-get install bison
   sudo apt-get install flex
   ```

   Download from

   `http://archive.ubuntu.com/ubuntu/pool/main/z/zlib/`
   packages

   `zlib1g and zlib1g-dev`
   and install them from command line:

   ```
   sudo dpkg --install zlib1g_1.2.3.3.dfsg-7ubuntu1_i386.deb
   sudo dpkg --install zlib1g-dev_1.2.3.3.dfsg-7ubuntu1_i386.deb
   ```

2. Then we add lines to $HOME`/.profile`

   ```
   export GEMS=$HOME/gems
   export SIMICS_INSTALL=$GEMS/simics-2.2.19
   export SIMICS_EXTRA_LIB=./modules
   export PYTHONPATH=./modules
   ```

   Note: You can use or Simics 2.2.19 or Simics 3.0.30, and choose between them by setting SIMICS_INSTALL environment variable to proper Simics directory.

   Now restart your machine.

3. Copy `simics-2.2.19-x86-linux.tar.gz` to $GEMS directory. Install Simics 2.2.19:

   ```
   tar xvzf $GEMS/simics-2.2.19-x86-linux.tar.gz
   ln -s $GEMS/simics-2.2.19 $GEMS/simics
   ```

   Note: You can use or Simics 2.2.19 or Simics 3.0.30, and choose between them by linking $GEMS/simics directory to proper Simics directory.

   You may need to copy your license file to $GEMS/simics-2.2.19/licenses.

4. Now we should prepare for configure:

   - Ubuntu uses '**dash**' instead of bash when sh is executed, which causes the configure script to fail. You can verify this by running
     `ls -l /bin/sh`.
     If it links to **dash**, that is problem. You can replace it with a link to bash and things should go better:
     `ln -sf bash /bin/sh`.

Note that you need root priviledges.

- Add the following lines to $GEMS/simics/scripts/cctype.sh, right after the gcc3 check:

```
# treat gcc4 as gcc3 for now
$COMPILER -dumpversion 2>&1 |  egrep "^[A-Za-z-]*4\." >/dev/null
if [ "$?" = "0" ] ; then
 echo "gcc3" ; exit
fi
```

- Change the line in
/gems/simics-2.2.19/x86-linux/obj/include/simics/memory_api.h:

```
void VT_do_mem_op(generic_transaction_t *restrict mop, char
*restrict src, char *restrict dst);
```

with line:

```
void VT_do_mem_op(generic_transaction_t * mop, char * src,
char * dst);
```

This can be safely done, since restrict is used for loop optimizations.

5. Configure Simics for host (assume x86-linux). The output will be Makefile and some header files, necessary for ruby to work:

```
cd /gems/simics/x86-linux
../configure CC=gcc-4.1 CXX=g++-4.1
```

6. Create ruby and opal module directories, and create symlinks

```
cd $GEMS/simics/src/extensions/
mkdir ruby
cd ruby
ln -s ../../../../ruby/module/ruby.c
ln -s ../../../../ruby/interfaces/mf_api.h
ln -s ../../../../ruby/module/Makefile
ln -s ../../../../ruby/module/commands.py
ln -s ../../../../ruby/simics/commands.h
cd ..
mkdir opal
cd opal
ln -s ../../../../opal/module/opal.h
ln -s ../../../../opal/module/opal.c
ln -s ../../../../opal/module/Makefile
ln -s ../../../../opal/system/hfa_init.h
ln -s ../../../../opal/module/commands.py
mkdir tourmaline
cd tourmaline
ln -s ../../../../tourmaline/simics/commands.h
ln -s ../../../../tourmaline/module/commands.py
```

```
ln -s ../../../../tourmaline/module/Makefile
ln -s ../../../../tourmaline/module/tourmaline.c
```

7. Edit $GEMS/simics/config/modules.list-local, and add these lines

```
ruby       | API_2.0 | v9
opal       | API_2.0 | v9
tourmaline | API_2.0 | v9
```

8. Copy your compiler's libstdc++ and libgcc_s to $GEMS/simics/x86-linux/sys/lib

```
cp /usr/lib/libstdc++.so.5 $GEMS/simics/x86-linux/sys/lib
cp /lib/libgcc_s.so.1 $GEMS/simics/x86-linux/sys/lib/
```

9. Edit GEMS Makefiles:

- `$GEMS/common/Makefile.simics_version`
  - set the version string to `2.2.X`
- `$GEMS/common/Makefile.common`
  - Modify the CC for your chosen compiler (in our case just in `x86-linux` department)
  - Set value to `HOST_TYPE` directly:
    `HOST_TYPE=x86-linux`
    because script `calc_host.sh` is not working properly
  - After line `ifeq ($(SIMICS_VERSION),2.2.x)` modify lines

```
SIMICS_ROOT := $(GEMS_ROOT)/simics
SIMICS_INCLUDE_ROOT := $(SIMICS_INSTALL)/src/include
SIMICS_EXEC_ROOT := $(GEMS_ROOT)/simics
```

- `$GEMS/ruby/module/Makefile`
  - If you are not using x86-platform, modify `CC_version`
  - Set value to `HOST_TYPE` directly:
    `HOST_TYPE=x86-linux`
    because script `calc_host.sh` is not working properly
  - Use the Simics-2.2.x-appropriate definition of `GEMS_ROOT` – comment line below line `For Simics 3.0`, and decomment line below `For Simics 2.2.x`
- `$GEMS/opal/module/Makefile`
  - If you are not using x86-platform, modify `CC_version`
  - Set value to `HOST_TYPE` directly:
    `HOST_TYPE=x86-linux`
    because script `calc_host.sh` is not working properly
  - Use the Simics-2.2.x-appropriate definition of `GEMS_ROOT` – comment line below line `For Simics 3.0`, and decomment line below `For Simics 2.2.X`

- `$GEMS/tourmaline/module/Makefile`
  - If you are not using x86-platform, modify `CC_version`
  - Set value to `HOST_TYPE` directly:
    `HOST_TYPE=x86-linux`
    because script `calc_host.sh` is not working properly.

3.3 Compilling

In this section, we will compile Ruby and Opal and run a simulation manually. This is meant to illustrate the steps necessary to get Ruby and Opal loaded into Simics. For actual research and result gathering, it is highly recommended to script the running of the simulation. The Multifacet group uses extensive scripting, as well as the use of the Condor batch-computing system, to automate the running and data collection of simulation. Some examples of our scripts are provided in the *$GEMS/gen-scripts* directory. See the README file located in that directory, or see this short write-up on running a simulation using ["gen-scripts"].

Compile Ruby

To compile Ruby, specify the SLICC protocol specification, and the destination directory which will be created in $GEMS/simics/home . This allows multiple protocols to co-exist in the Simics tree. This is accomplished by the *movemodule* target in the Ruby Makefile. If desired, you can edit this target such that the ruby module remains in $GEMS/simics/x86-linux/lib.

```
cd $GEMS/ruby
make PROTOCOL= MESI_CMP_filter_directory DESTINATION=TMTest
```

There is high probablity to see some error about `extra qualification`. Go to specified .h file and delete class name in front of double colon. This is compiler issue, since there are GCC 4.x on Ubuntu 7.10, but GEMS was made for GCC 3.4.6.

When the above command is executed, SLICC is first compiled and run on the MOSI_SMP_bcast specification files. SLICC generates C++ code that is placed in the *ruby/generated/ MESI_CMP_filter_directory* directory.

MOSI_SMP_bcast is a snooping, broadcast-based cache coherence protocol for a SMP system. By default, Ruby is configured to work with this protocol. Other protocols may require you to change configuration options.

Compile Tourmaline

Tourmaline's compile syntax is very similar to that of Ruby and Opal. A required argument to "make" is the DESTINATION= definition, which determines where the Tourmaline module will be installed. Otherwise, compilation is identical to other modules.

*Note: Tourmaline was developed and tested with Simics 2.2.X. It is **not** compatible with Simics 3.0.*

```
make -j 4 DESTINATION=TMTest
```

Note: Simics may not detect that its list of modules has changed, so it may be required to delete the `modules.cache` file in `$GEMS/simics/x86-linux/lib/` if compilation of Tourmaline fails when linking the Tourmaline module. This problem is characterized by the error message:

```
No rule to make target: tourmaline
```

4 Frequently used commands on Simics 3.0.30

For all commands in this chapter it is assumed that we use Simics 3.0.30:

```
cd $GEMS/
ln -s simics_3_workspace simics
```

4.1 Tracing

This section describes how to use the tracing facility provided by the trace module. It enables the user to get all instructions, memory accesses, and exceptions printed out in the order in which they occurred.
N*ote: Simics is heavily optimizing execution. To get the trace output we want here, Simics has to be told to send all memory transactions to the trace module. This is done by adding the flag -stall when starting.*

Start a new Simics:

```
cd $GEMS/simics/home/MOSI_SMP_bcast
./simics -stall -c $GEMS/checkpoints/first-sarek.check
```

Now let us create a tracer by typing:

```
simics> new-tracer
Trace object 'trace0' created. Enable tracing with 'trace0.start'
simics>
```

The trace module tells us that it has created an object named `trace0` that will handle the tracing. This object supports two commands, `trace0.start` and `trace0.stop`; what they do should be obvious. Let us try to trace the first 10 instructions executed.

```
simics>    trace0.start traceFile.txt
Tracing    enabled
simics>    continue 10
[cpu0] v:0x0000000001000a488 p:0x00002c0a488  sethi %hi(0x1000000), %g4
```

To write lines to file type:

```
simics> trace0.stop
```

There is file `output.txt` in directory `$GEMS/simics/home/MOSI_SMP_bcast`, and in our example its contents is:

```
inst: [        1] CPU  0 <v:0x0000000001004a070> <p:0x00002c4a070>
81c3e008 jmpl [%o7 + 8], %g0
exce: [        1] CPU  0 exception  96 (Interrupt_Vector)
```

```
inst: [        2] CPU  0 <v:0x0000000001000c00> <p:0x00002c00c00>
c2d80920 ldxa [%g0 + %g0] 0x49, %g1  # ASI_INTR_RECEIVE
inst: [        3] CPU  0 <v:0x0000000001000c04> <p:0x00002c00c04>
80886020 andcc %g1, 32, %g0
inst: [        4] CPU  0 <v:0x0000000001000c08> <p:0x00002c00c08>
126825ff bne,pt %xcc, 0x100a404
inst: [        5] CPU  0 <v:0x0000000001000c0c> <p:0x00002c00c0c>
01000000 nop
inst: [        6] CPU  0 <v:0x000000000100a404> <p:0x00002c0a404> 84102040
mov 64, %g2
inst: [        7] CPU  0 <v:0x000000000100a408> <p:0x00002c0a408>
cad88fe0 ldxa [%g2 + %g0] 0x7f, %g5  # ASI_INTR_R
inst: [        8] CPU  0 <v:0x000000000100a40c> <p:0x00002c0a40c>
0ec9401f brgez,pt %g5, 0x100a488
inst: [        9] CPU  0 <v:0x000000000100a410> <p:0x00002c0a410> 893f703d
srlx %g5, 61, %g4
```

The lines starting with inst: indicates that an instruction is executed. The rest of the lines describes the address of the instruction and the instruction itself, in binary and human-readable form.

The lines starting with data: indicates that some instructions are performing memory operations. The rest of the line show the memory operation address, its type (Vani stands for Vanilla, i.e., standard type), whether it's a read or a write, its size and finally its value.

You can obtain some statistics through commands:

```
simics> pstats
simics> ptime
```

4.2 Tracing just user programs

There is two simics modules for tracing specific issues. It is `cpu-mode-tracker` and `linux-process-tracker`.

cpu-mode-tracker

`cpu-mode-tracker` only distinguishes between user and supervisor mode (but works on all targets and operating systems). Supervisor mode implies kernel code, and user mode implies user code.

Note: User mode tracks ALL programs in user mode, not just program started from target console.In Sarek platform in user space reside only command prompt user program, which you should not kill.

linux-process-tracker

`linux-process-tracker` tracks processes on Linux on PowerPC, UltraSPARC and x86 targets.

Note that a tracker is watching over a specific set of processors. This set should typically contain all processors that run an operating system together, and no other processors. That way, processes that migrate between processors will not get lost, and processes running in different operating system

instances will not be mixed up. If there is more than one operating system running in the simulation, they will need separate trackers.

We will show the CMP example for x86-440bx target, so we need SMP version of Linux. This can be download from download section of Simics, and we will use `tango1-fedora5.craff`. Then we will put it in the directory `/gems/simics-3.0.30/targets/x86-440bx/images`.

Now, it's time to add line

```
if not defined num_cpus        {$num_cpus = 4}
```

in `/gems/simics-3.0.30/targets/x86-440bx`.

You should know that target documentation says “that most Linux versions are limited to 8 processors”.

We will start this target:

```
cd $GEMS/simics
./simics targets/x86-440bx/tango-common.simics
simics>continue
```

We will wait until system is booted, login with 'root', password is 'simics', and then:
CTRL+C

```
simics> new-linux-process-tracker
New process tracker tracker0 created.
```

The tracker’s set of processors will initially be empty. You can add the processors you want it to watch with its add-processors command.

```
simics> tracker0.add-processor cpu0
simics> tracker0.add-processor cpu1
simics> tracker0.add-processor cpu2
simics> tracker0.add-processor cpu3
```

Since we don't provide a named set of parameters by giving the kernel argument in command `new-linux-process-tracker`, we must use the `autodetect-parameters` command.

```
simics> tracker0.autodetect-parameters
```

This makes the process tracker examine memory to figure out what operating system the target machine is actually running. This is why the target system must be booted before this command is issued.

```
simics>tracker0.activate
```

It's time to make python file named `trace-ls.py` in $GEMS with contents:

```
cli.run_command("new-tracer")
def exec_hap(user_arg, tracker, tid, cpu, binary):
     if binary.endswith("ls"):
          def active_hap(user_arg, tracker, tid, cpu, active):
            if active:
              cli.run_command("trace0.start")
            else:
              cli.run_command("trace0.stop")
          SIM_hap_add_callback_obj_index("Core_Trackee_Active",
tracker,0, active_hap, None, tid)
SIM_hap_add_callback_obj("Core_Trackee_Exec", conf.tracker0, 0,
exec_hap, None)
```

Now, we will start this script (You should change $GEMS in next command with your $GEMS directory):

```
simics>run-python-file  $GEMS/trace-ls.py
```

Now, type

```
simics>continue
```

and then we can run `ls` command on target. We will see race only for `ls` command, and editing the string “ls” in python file we can trace our benchmark program.

5 Starting benchmark on Simics 2.2.19

Our goal is to make visible host files to target machine. First we can load ISO image in target machine, then using it we can load whole host filesystem to target.

For all commands in this chapter it is assumed that we use Simics 2.2.19:

```
cd $GEMS/
ln -s simics-2.2.19 simics
```

5.1 Accessing a CD-ROM Image File

A file containing an ISO-9660 image can be used as medium in the simulated CD-ROM. This image file can be created from real CD-ROM disks, or from collections of files on any disk.

An image can be created from a set of files with the *mkisofs* program. For example:

```
mkisofs -l -allow-leading-dots -o $HOME/testImage -r $GEMS /ruby
```

Start Simics:

```
cd $GEMS/simics/home/MOSI_SMP_bcast
./simics -stall -c $GEMS/checkpoints/first-sarek.check
simics> continue
CTRL+C
```

Once you have an image file, a `file-cdrom` object can be created, and then inserted into a simulated CD-ROM device. In the first following command change $HOME with your home directory.

```
simics> new-file-cdrom $HOME/testImage
cdrom 'testImage' created
simics> cd25B_2_6.insert testImage
Inserting media 'testImage' into CDROM drive
```

Note that `cd25B_2_6` above refers to the Simics object name of the CD-ROM drive. This may, or may not be called `cd25B_2_6`. To see which object name to use, try the `list-objects` command and look for an object of class `scsi-cdrom` or `ide-cdrom`.

Now type:

```
simics> continue
```

You can now access cdrom on target console by typing:

```
# cd /cdrom/cdrom
# ls -l
```

5.2 Accesing host filesystem through SimicsFS

SimicsFS gives you access to the file system of your real computer inside the simulated machine. This greatly simplifies the process of importing files into the simulated machine.

Start Simics:

```
cd $GEMS/simics-2.2.19/home/sarek
./simics -stall -c $GEMS/checkpoints/first-sarek.check
```

In some other Linux console type commands:

```
mkdir $GEMS/targetTemp
cp $GEMS/simics-2.2.19/import/sun4u/mount_hostfs $GEMS/targetTemp/mount
cp $GEMS/simics-2.2.19/import/sun4u/hostfs-sol9 $GEMS/targetTemp/hostfs
mkisofs -l -allow-leading-dots -o $HOME/targetTemp -r $GEMS/targetTemp
```

In Simics console type (change $HOME with your home directory path):

```
simics> new-file-cdrom $HOME/targetTemp
cdrom 'targetTemp' created
simics> cd25B_2_6.insert targetTemp
Inserting media 'targetTemp' into CDROM drive
simics>continue
```

In target console type:

```
# mkdir /usr/lib/fs/hostfs
```

Sometimes next command will not work until some time expired:

```
# cd /cdrom/cdrom
# cp mount /usr/lib/fs/hostfs/mount
# cp hostfs /usr/kernel/fs/sparcv9/hostfs
```

Add the following line to /etc/vfstab on the simulated disk, for example using Vi editor (good tutorial for Vi can be found on ViTutorial):

```
hostfs   -   /host   hostfs   -   no   -
```

Create the mount point and mount on it on the simulated machine with:

```
# mkdir /host
# mount /host
```

You should now be able to do `ls /host` on the simulated system to get a list of the files on the host. It's time to save checkpoint with SimicsFS (In the first command change $GEMS with your $GEMS directory):

```
simics>CTRL+C
simics>write-configuration $GEMS/checkpoints/sarekSimicsFS.check
simics>quit
```

5.3 Starting TM benchmark

5.3.1 Making multiprocessor target checkpoint

First we will make sarek checkpoint with four processors by editing $GEMS/simics-2.2.19/home/sarek/sol9-run.simics. Insead of lines:

```
# set up 1 processor with 256MB
@if not "boards" in dir():
     boards = { 0 : [[0, 1, 256]]}
```

we will put lines:

```
# set up 4 processor with 512MB of shared memory
@if not "boards" in dir():
     boards = { 0 : [[0, 4, 512]]}
```

It's time to:

```
cd $GEMS/simics-2.2.19/home/sarek
./simics -x sol9-run.simics
simics>continue
```

When system is booted is time to make checkpoint (change $GEMS with your $GEMS directory path):

```
CTRL+C
simics>write-configuration $GEMS/checkpoints/sarek4P.simics
simics>quit
```

5.3.2 Target accessing host

```
cd $GEMS/simics-2.2.19/home/sarek
./simics -stall -c $GEMS/checkpoints/sarek4P.simics
```

In some other Linux console type commands:

```
mkdir $GEMS/targetTemp
cp $GEMS/simics-2.2.19/import/sun4u/mount_hostfs $GEMS/targetTemp/mount
cp $GEMS/simics-2.2.19/import/sun4u/hostfs-sol9 $GEMS/targetTemp/hostfs
mkisofs -l -allow-leading-dots -o $HOME/targetTemp -r $GEMS/targetTemp
```

In Simics console type (change $HOME with your home directory path):

```
simics> new-file-cdrom $HOME/targetTemp
```

```
cdrom 'targetTemp' created
simics> cd25B_2_6.insert targetTemp
Inserting media 'targetTemp' into CDROM drive
simics>continue
```

In target console type:

```
# mkdir /usr/lib/fs/hostfs
```

Sometimes next command will not work until some time expired:

```
# cd /cdrom/cdrom
# cp mount /usr/lib/fs/hostfs/mount
# cp hostfs /usr/kernel/fs/sparcv9/hostfs
```

Add the following line to /etc/vfstab on the simulated disk, for example using Vi editor (good tutorial for Vi can be found on ViTutorial):

```
hostfs   -   /host   hostfs   -   no   -
```

Create the mount point and mount on it on the simulated machine with:

```
# mkdir /host
# mount /host
```

You should now be able to do `ls /host` on the simulated system to get a list of the files on the host. It's time to save checkpoint with SimicsFS (In the first command change $GEMS with your $GEMS directory):

```
simics>CTRL+C
simics>write-configuration $GEMS/checkpoints/sarek4PSimicsFS.check
simics>quit
```

5.3.3 Installing GCC on Solaris target

Go to http://www.sunfreeware.com/indexsparc9.html and download:

`gcc-3.4.6-sol9-sparc-local.gz` and `libinconv-1.11-sol9-sparc-local.gz`. Place it on Desktop and run:

```
cd $HOME/Desktop
gunzip -d gcc*
gunzip -d lib*
```

Start checkpoint:

```
cd $GEMS/simics-2.2.19/home/sarek
./simics -stall -c $GEMS/checkpoints/sarek4PSimicsFS.simics
simics>continue
```

In target console run commands:

```
#mkdir /usr/local
#mkdir /usr/local/bin
#PATH=$PATH:/usr/local/bin:/usr/ccs/bin
#export PATH
#bash
#mkdir /priv
#cd /priv
#cp /host/root/Desktop/libiconv-1.11-sol9-sparc-local /priv
#cp /host/root/Desktop/gcc-3.4.6-sol9-sparc-local /priv
```

Now we will install packages:

```
#head /priv/gcc*
#/usr/sbin/pkgadd -d gcc*
#/usr/sbin/pkgchk -d gcc* SMCgcc
#head /priv/lib*
#/usr/sbin/pkgadd -d lib*
```

Go back to simics console (change $GEMS with your GEMS directory):

```
CTRL+C
simics>write-configuration $GEMS/checkpoints/sar4PsimFSgcc.check
simics>quit
```

5.3.4 Setting TM benchmark deque parameters

OS events

If we decide to abort in the middle of an OS event, we wait till the event is completed before unrolling execution. Only user state is rolled back.

Procedure

Tourmaline emulates the same binary interface as the LogTM implementation in Ruby. However, Tourmaline is intended to run stand-alone, without either Ruby or Opal loaded. It will not function correctly in the presense of these modules. The Tourmaline module will perform functional simulation of transactional memory. Tourmaline is a tool intended to enable warm-up of transactional applications, as well as to facilitate debugging of transactional applications seperate from the debugging of the timing simulator. However, tourmaline is a viable tool for (much) longer simulations at the cost of some timing fidelity.

We show only procedure for Ruby LogTM, because benchmarks in GEMS release are not binary-compatible with Tourmaline. GEMS authors recently made same changes on ruby, which are not backward-compatible with tourmaline. However, Ruby also has a Tourmaline-like functionallity which simulates a perfect zero-latency memory system by bypassing caches. It's possible by using XACT_ENABLE_TOURMALINE flag in *$GEMS/ruby/config/rubyconfig.defauls.*

1. First, we need to change some parameters in $GEMS/ruby/config/rubyconfig.defaults:

- REMOVE_SINGLE_CYCLE_DCACHE_FAST_PATH: Have to set to true in order to use MESI_CMP_filter_directory protocol
- NUMBER_OF_VIRTUAL_NETWORKS: Set to **5** in order to get the proper number of virtual channels for the on-chip interconnect used for MESI_CMP_filter_directory.
- XACT_MEMORY: Set to true for TM protocols. Must be set to true during protocol compilation when simulating a lazy system (e.g. any system with Lazy version management)

2. In $GEMS/gen-scripts/multifacet.py in function start_ruby change lines:

- `mfacet.run_sim_command("ruby0.setparam g_MEMORY_SIZE_BYTES 8589934592")`

  with

  `mfacet.run_sim_command("ruby0.setparam g_MEMORY_SIZE_BYTES 536870912")`

  because we made checkpoint with 512MB shared memory.

- `mfacet.run_sim_command("ruby0.setparam g_PROCS_PER_CHIP %d" % processors_per_chip)`

  with

  `mfacet.run_sim_command("ruby0.setparam g_PROCS_PER_CHIP 4")`

  because we made checkpoint with 4 processors.

- `mfacet.run_sim_command("ruby0.setparam_str g_NETWORK_TOPOLOGY FILE_SPECIFIED")`

  with

  `mfacet.run_sim_command("ruby0.setparam_str g_NETWORK_TOPOLOGY PT_TO_PT")`

  because of CMP topology.

3. In $GEMS/gen-scripts/multifacet.py in function start_ruby_small_cache change lines:

- `mfacet.run_sim_command("ruby0.setparam g_MEMORY_SIZE_BYTES 8589934592")`
  with

  `mfacet.run_sim_command("ruby0.setparam g_MEMORY_SIZE_BYTES 536870912")`

  because we made checkpoint with 512MB shared memory.

4. We will write file $GEMS/microbenchmarks/transactional/deque:

```
#############################################################
@sys.path.append("../../../gen-scripts")
@import microbench, workloads, mfacet

###### Read simulation parameters
@env_dict = workloads.prepare_env_dictionary(simics = 1)
```

```
#@processors = int(workloads.get_var(env_dict, "PROCESSORS"))
#we will set it manually, this workload file can be used for other
workloads
@processors = int(4)


#                 0  1  2  3  4  5  6  7  8   9  10  11  12  13  14
#15
@ruby_proc_map = (1, 2, 2, 4, 4, 8, 8, 8, 8, 16, 16, 16, 16, 16, 16,
16, 16)
@if(processors <= 16):
    ruby_procs = ruby_proc_map[processors]
    #proc_no = [0, 1, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
#19]
    proc_no = [0, 0, 1, 2, 3, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
19]
    #for processors' sets number 1 to 4, proc_no going 0 to 3
    #if something goes wrong, first check the output of command
#psrinfo on target console -already typed below, you should just
#check output
    #if the number of processors greater than 16, change elif
     #(processors<=32)
elif (processors <= 32):
    ruby_procs = 32
    proc_no = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47]
else:
    processors = -1

@print "running simics with %d processors" % ruby_procs

#we can see workload parameters for deque in $GEMS/gen-
scripts/workload.py
#@lock_type = workloads.get_var(env_dict, "LOCK_TYPE")
@lock_type = "TM"
#@transactions = int(workloads.get_var(env_dict, "TRANSACTIONS"))
@transactions = 2000
#@dump_interval = int(workloads.get_var(env_dict, "DUMP_INTERVAL"))
@dump_interval = 1
#@arg_str = workloads.get_var(env_dict, "MBENCH_ARG_STRING")
@arg_str = "2000 32"

@env_dict = workloads.prepare_env_dictionary(simics = 1)
@visualizer = int(workloads.get_var(env_dict, "XACT_VISUALIZER"))
@results_dir = workloads.get_var(env_dict, "RESULTS_DIR")

#@filename_prefix = "%s/%s" % (results_dir,
#workloads.get_microbench_output_file_name_prefix(env_dict, 0))
@filename_prefix = "dequeIzlaz"
#output will be put in $GEMS/simics/home/TMTest

#@mfacet.run_sim_command('read-configuration "../../checkpoints-
```

```
#u3/microbenchmark/microbenchmark-%dp.check"' % ruby_procs)
@mfacet.run_sim_command('read-configuration
"/gems/checkpoints/sar4PsimFSgcc.check"')

@mfacet.run_sim_command('magic-break-enable')

@hostpath = "/host" + os.getcwd() +
"/../../../microbenchmarks/transactional/"

@command_lines = [
     "psrinfo\n",
     "PATH=$PATH:/usr/local/bin:/usr/ccs/bin\n",
     "export PATH\n",
     "bash\n",
     "mkdir /transactional\n",
     "cp -r " + hostpath + " /\n",
     "umount /host\n",
     "cd /transactional/deque\n",
     "make\n"
    ]

#copied from btree.simics commented example with some modifications
## Create Processor Sets
#@for j in range(processors + 1):
#Solaris allows you to bind upto n-1 processors to various processor
#sets, where n is the total number of processors in #the system.
@for j in range(processors):
    if(j != 0):
        print "j=%d, num_p=%d" % (j, processors)
        command_lines.append("psrset -c\n")
        print "psrset -c\n"
        command_lines.append("psrset -a %d %d\n" % (j, proc_no[j]))
        print "psrset -a %d %d\n" % (j, proc_no[j])

@command_lines.append("./deque_TM %d %s\n" % (processors-1, arg_str))
#starting benchmark

@mfacet.console_commands(command_lines,"#")
c

@microbench.start_ruby(debug=0)

@if visualizer == 1:
  mfacet.run_sim_command('ruby0.xact-visualizer-file %s.visual' %
filename_prefix)
@mfacet.setup_run_for_n_transactions(transactions, dump_interval)
@mfacet.run_sim_command('c')

@mfacet.run_sim_command('ruby0.dump-stats %s.stats' %
filename_prefix)

quit
```

```
----------------
************************************************************
```

We bind threads on processors. First, we create processor sets using pset_create and then bind the threads belonging to tm-deque using pset_bind (later is done in benchmark .c file). There is no context switching during a transaction. This will prevent other processes from interfering with your trace.

5.3.5 Starting TM benchmark deque

Communication between Simics and our modules going through MAGIC calls. This is instructions in benchmarks which results in nops in Simics, but can be used for Simics to signal our modules, for example, to stop simulation. Magic call have codes which differentiate its meaning, for example, begin_transaction from commit_transaction.

```
cd $GEMS/simics/home/TMTest
./simics -stall -x /gems/microbenchmarks/transactional/deque/deque.simics
```

Reason why we must go to TMTest directory is because we previously installed ruby and tourmaline there.

Your host console output at same point should look like:

```
...
end_transaction_magic_call: transaction started: 1, transaction completed: 0,
transaction_limit: 20, mem_res: 155.44921875 mem_total: 184.8828125 mem_ratio: 0.840799

end_transaction_magic_call: transaction started: 1, transaction completed: 1,
transaction_limit: 20, mem_res: 155.51171875 mem_total: 185.01171875 mem_ratio: 0.840551

end_transaction_magic_call: transaction started: 2, transaction completed: 1,
transaction_limit: 20, mem_res: 155.51171875 mem_total: 185.01171875 mem_ratio: 0.840551

end_transaction_magic_call: transaction started: 2, transaction completed: 2,
transaction_limit: 20, mem_res: 155.51171875 mem_total: 185.01171875 mem_ratio: 0.840551

...
```

Note that the 'transaction' you see on the screen output refers to a workload transaction which is different from LogTM transactions. Workload transactions record the number of times the workload has executed a benchmark-specific unit of work. This guarantees the same amount of 'work' has been done, even when different execution paths have been followed.

All LogTM statistics in the stats files start with the string "xact." For example, `xact_size_dist` is number of commited transactions. `xact_aborts` is number of aborts. No way to find out which transactions began and not commited yet by the stop simulation. If you want to turn off LogTM statistics, turn off the transactional memory profiling by setting the parameter "PROFILE_XACT" in $GEMS/ruby/rubyconfig.defaults to false. Also, for each processor you can see statistics like in this example:

```
xact_trans_cycles_processor_0: 5744

xact_aborting_cycles_processor_0: 4832
```

```
xact_barrier_cycles_processor_0: 0
xact_backoff_cycles_processor_0: 1059
xact_stall_trans_cycles_processor_0: 5347
xact_nontrans_cycles_processor_0: 59578
xact_stall_nontrans_cycles_processor_0: 0
timed_cycles_processor_0: 0
```

6 References

[1] Milo M. K. Martin1, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill and David A. Wood, *Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset* , Computer Architecture News (CAN), September 2005

[2] Mike Marty, Brad Beckmann, Luke Yen, Alaa Alameldeen, Min Xu, Kevin Moore, *Multifacet GEMS General Execution Driven Simulator*, ISCA Tutorial, June 5th, 2005

[3] http://www.cs.wisc.edu/gems/doc/gems-wiki/moin.cgi, USA, 2005

[4] http://lists.cs.wisc.edu/mailman/listinfo/gems-users, USA, 2008

[5] http://lists.cs.wisc.edu/mailman/listinfo/gems-users, USA, 2008

[6] https://www.simics.net/, USA, 2008