Agate User's Guide

System Technology and Architecture Research (STAR) Lab
Oregon State University, OR

SMART Interconnects Group
University of Southern California, CA

System Power Optimization and Regulation Technology (SPORT) Lab
University of Southern California, CA

July 2016

Table of Contents

1. Introduction 3
2. Getting Started 3
   2.1 Download and Unpack 3
   2.2 Install Dependent Tools 4
      2.2.1 The gem5 simulator 4
      2.2.2 DSENT 4
   2.3 Running Simulation 5
      2.3.1 Network test 5
      2.3.2 Full system simulation 5
   2.4 Simulation Output 6
3. Configuration and Functionality 6
   3.1 Power gating options 6
   3.2 Simulation options introduced by *Agate* 6
4. Instrumentation 7
   4.1 Timing 7
   4.2 Power and Energy 7
5. Extensions 8
   5.1 Basic Components 8
   5.2 Power Gating Strategies 8
   5.3 Compatibility with more recent versions of the gem5 simulator 9
   5.4 A list of new or changed files to implement *Agate* in gem5 9
6. Contributors and Contact Infomation 11
7. Acknowledgment 11

References 11

1. Introduction

Agate is a simulator tool that meets the key requirements for simulating NoC power-gating techniques. To enable full-system simulation, *Agate* is implemented as a closed-loop, cycle-accurate module in the widely used gem5 simulator [3] and the included Garnet [1]. In addition to behavioral modeling of power-gating, *Agate* is able to use inputs from other simulators that provide detailed power/energy modeling of NoC components, allowing flexibility at different modeling levels and accuracies. Various statistics relating to power-gating are collected by *Agate*, such as the number of cycles each NoC router is power-gated, the distribution of the length of router idle intervals, the total number of stalls and the total length of stalls of packets due to power-gating, and many others.

If you use *Agate* in your research that leads to publications, we would appreciate a citation to our paper "Simulation of NoC Power-Gating: Requirements, Optimizations, and the Agate Simulator" in Journal of Parallel and Distributed Computing (JPDC) [4]. It is highly recommended to read that paper before (or at least in addition to) this manual in order to understand and utilize *Agate* better. This manual also assumes that the readers are familiar with the basic usage of Linux. There are many good tutorials about Linux available online.

2. Getting Started

2.1 Download and Unpack

The official releases of *Agate* are available at the STAR Lab's website under the "Software" section (the current link is http://web.engr.oregonstate.edu/~chenliz/software.html) or the SPORT Lab's website under the "Downloads -> Packages" tab (http://sportlab.usc.edu/downloads/packages/). *Agate* is developed under a specific version of gem5 and DSENT. To facilitate setup and installation process, the released package has already included the compatible version of the gem5 simulator and DSENT. Details on how to integrate *Agate* on newer versions of gem5 is discussed in Section 5.3.

After downloading *Agate* from the above website, unpack it in your selected working directory. **The remaining of this manual assumes that the current directory is the working directory after unpacking the *Agate* package.** You can see the following files and directories in the current directory:

```
Build_ALPHA_MESI_Two_Level  build_opts  COPYING  ext      parser.out   parsetab.pyc  SConstruct  src      tests
Build_ALPHA_Network_test    configs     DSENT    LICENSE  parsetab.py  README        scripts     system   util
```

2.2 Install Dependent Tools

Agate does not add new requirements for software dependency, and the released package includes gem5 and DSENT. However, additional tools and libraries may be needed on your machine to run gem5 and DSENT themselves.

2.2.1 The gem5 simulator

Agate is built based on the gem5 simulator version 2.0. To build gem5 2.0, you will need the following software:

- g++ version 4.3 or newer.
- Python, version 2.4 - 2.7 (Python 3.X is not supported). gem5 links in the Python interpreter, so you need the Python header files and shared library (e.g., /usr/lib/libpython2.4.so) in addition to the interpreter executable. These may or may not be installed by default. For example, on Debian/Ubuntu, you need the "python-dev" package in addition to the "python" package. If you need a newer or different Python installation but can't or don't want to upgrade the default Python on your system, see http://www.gem5.org/Using_a_non-default_Python_installation
- SCons, version 0.98.1 or newer. SCons is a powerful replacement for make. If you don't have administrator privileges on your machine, you can use the "scons-local" package to install scons in your m5 directory, or install SCons in your home directory using the '--prefix=' option.
- SWIG, version 1.3.34 or newer
- zlib, any recent version. For Debian/Ubuntu, you will need the "zlib-dev" or
- "zlib1g-dev" package to get the zlib.h header file as well as the library itself.
- m4, the macro processor.

For detailed information about building the gem5 simulator and getting started please refer to http://www.gem5.org .

2.2.2 DSENT

The current version of *Agate* integrates the electrical component of the DSENT NoC power model [5]. No additional library is needed in our tests, but if there is any further issues, please check out the publications and manuals on the DSENT website at https://sites.google.com/site/mitdsent/. The power statistics in the *Agate* simulation results are also partly based on the DSENT model. Users can adopt their own models to get different power statistics.

2.3 Running Simulation

Before running the simulation, make sure of no compilation error after compiling the gem5 simulator. For example, if you are using network test, run the following compilation command:

```
scons build/ALPHA_Network_test/gem5.opt
```

If you are running Alpha with two-level MESI, run the following compilation command:

```
scons build/ALPHA_MESI_Two_Level/gem5.opt
```

2.3.1 Network test

Here is an example of running a network test for an 8x8 mesh NoC with 0.01 injection rate, under uniform random traffic (`--synthetic=0`), for 10,000 cycles. Make sure to include `--garnet-network=fixed` to simulate the NoC with power gating functions.

```
./build/ALPHA_Network_test/gem5.opt configs/example/ruby_network_test.py --num-cpus=64 --num-dirs=64 --topology=Mesh --mesh-rows=8 --garnet-network=fixed --synthetic=0 --sim-cycles=10000 --injectionrate=0.01
```

2.3.2 Full system simulation

As an example to run full-system simulation, we use the PARSEC benchmarks [2] augmented by the University of Texas at Austin, which can be found at http://www.cs.utexas.edu/~cart/parsec_m5/ with installation instructions.

After the installation of PARSEC benchmarks, we create checkpoints for the gem5 simulator in order to focus on the region of interest (ROI). Details on how to create and restore checkpoints can be found at http://www.m5sim.org/Checkpoints. An example command is as follows

```
./build/ALPHA_MESI_Two_Level/gem5.opt configs/example/fs.py --num-cpus=64 --cpu-type=atomic --mem-size=1GB --checkpoint-dir=<CHECKPOPINT_DIR> --script=./scripts/blackscholes_64c_simsmall_ckpts.rcS
```

In the full system simulation, be sure to include `–ruby` and `--garnet-network=fixed` options to activate the ruby memory system (http://www.m5sim.org/Ruby) and the fixed pipeline of routers. An example of execution command to restore checkpoint is as follows, where `<CHECKPOPINT_DIR>` is the path to the checkpoint directory.

```
./build/ALPHA_MESI_Two_Level/gem5.opt configs/example/fs.py --num-cpus=64 --restore-with-cpu=timing --ruby --l1i_size=32kB --l1d_size=32kB --l2_size=16MB
```

```
--l2_assoc=16 --num-l2caches=64 --num-dirs=4 --mem-size=1GB --garnet-net-
work=fixed --topology=MeshDirCorners --mesh-rows=8 --checkpoint-
dir=<CHECKPOPINT_DIR> --checkpoint-restore=1
```

2.4 Simulation Output

The output of *Agate* are integrated into the standard output of the gem5 simulator. The output directory and files can be directed using the following options (provided by the gem5 simulator).

```
--outdir=<OUTPUT_DIR>
--stats-file=<STATS_FILE>
--stdout-file=<STDOUT_FILE>
```

3. Configuration and Functionality

3.1 Power gating options

TABLE I lists some parameters that can be changed directly in *Agate*.

TABLE I. PARAMETERS IN *AGATE*.

Term	Defined in	Default value	Meaning
TIMEOUT	`PGController_d.hh`	4	After becoming idle, the waiting time before sleep. Must be greater than SLEEP_MARGIN
SLEEP_MARGIN	`PGController_d.hh`	4	Minimum waiting time before sleep after sending sleep notification *S_SLEEP_NTF* to ensure delivery of in-flight flits, or the maximum time that can send active notification *S_ACTIVE_NTF* in advance before this router becoming fully active
WAKEUP_LAT	`PGController_d.hh`	8	Router wakeup latency
TBE	`PGController_d.hh`	10	Break-even time, the number of cycles the router needs to sleep in order to compensate the wakeup energy overhead

The TIMEOUT must be greater than or equal to SLEEP_MARGIN to ensure correct behaviors of power-gating, as explained in details in [4]. Note that SLEEP_MARGIN is actually both the minimum waiting time before sleep and the maximum time that can send active notification in advance, because they both include the number of router cycles that a flit cannot be stalled (three cycles SA, ST, LT in a five-stage pipeline), and one cycle spent on the router link, as elaborated in [4].

3.2 Simulation options introduced by *Agate*

1) Display debug information. To activate the debugging, add the following option in the command line

```
--pg-debug
```

If the option `pg-debug` is enabled (discussed in Section 3.1), it is strongly recommended to also include the redirecting option `--stdout-file=<STDOUT_FILE>`, so that the debugging output is redirected to

TABLE II. NEW STATISTICAL DATA INTRODUCED BY AGATE.

Name	Meaning
system.ruby.network.routers00.pg_percentage	Power gating percentage in the number of cycles for each router
system.ruby.network.router_dynamic_power	Overall router dynamic power
system.ruby.network.router_static_power	Overall router static power
system.ruby.network.pg_wakeup_overhead	Overall router wakeup overhead amortized to power
system.ruby.network.pg_times	Average number of times a router gets power gated
system.ruby.network.pg_percentage	Average power gating percentage
system.ruby.network.pg_stuck_times	Total number of times a packet is stuck due to power gating
system.ruby.network.waiting_for_pg_cycles	Total number of cycles a packet is waiting for sleep router

STDOUT_FILE. To add or modify the debugging outputs, use codes started by PG_PRINT in the source code. Example:

```
PG_PRINT("Router %02d [%05d]: Router put to sleep.\n", get_id(), int(curCycle()));
```

Output in STDOUT_FILE:

```
Router 03 [00041]: Router put to sleep.
```

2) Two new synthetic traffic patterns added in network test:

```
--synthetic=3
```

Bit Reverse traffic pattern

```
--synthetic=4
```

Transpose traffic pattern.

4. Instrumentation

Table II shows the new statistical data introduced by *Agate*.

4.1 Timing

Agate simulates power-gating in a cycle-accurate manner and changes the timing of NoC router components accordingly in Garnet to enable closed-loop simulation. The final packet latency and execution time metrics are reported through the gem5 simulator, which are in the STATS_FILE.

4.2 Power and Energy

As mentioned above, for the ease of *Agate* users, we have integrated the latest DSENT [5] NoC power model in providing the aforesaid energy metrics. We modified the source codes of the DSENT statistics in ./src/mem/ruby/network/dsent in the following way: Reported static power with power gating P_R^S

(`system.ruby.network.router_static_power`) is equal to the static power without considering power gating $P^S_{R,orig}$ multiplied by the percentage of router active time, i.e.,.

$$P^S_R = P^S_{R,orig} \cdot (1 - p_{PG}). \quad (1)$$

where p_{PG} is `system.ruby.network.pg_percentage`.

Currently we consider zero static power consumption when a router is power gated, but users can modify the code at `./src/mem/ruby/network/dsent` if they consider non-zero static power in power gating. If users want to integrate other models than DSENT, they can remove the DSENT folders in the gem5 simulator and use provided stats data listed Table II to calculate the new static power.

Note that the wake-up overhead P^{wk}_R is reported separately in the stats report (`system.ruby.network.pg_wakeup_overhead`), not included in the dynamic power P^d_R and static power P^s_R. The overall wakeup energy overhead is amortized into each cycle and reported as the wakeup power overhead to provide a direct comparison with in the dynamic power P^d_R and static power P^s_R. P^{wk}_R is calculated by

$$P^{wk}_R = N_{PG} \cdot \frac{(e^s_R \cdot t_{BE})}{T} \quad (2)$$

where T is the overall runtime, N_{PG} is the total number of times the router gets power-gated (`system.ruby.network.routers00.pg_times`). e^s_R denotes the NoC router static energy per cycle and t_{BE} is the break-even time, i.e., the number of cycles the router needs to sleep in order to compensate the wakeup energy overhead. The breakeven time t_{BE} can vary a lot for different router designs. Its default value is set to 10 cycles in *Agate* which is derived from post-synthesis simulation of a Verilog implementation of a NoC router. However, users can change this value through the parameter TBE as shown in TABLE I.

5. Extensions

One of the key features of *Agate* is its extensibility. A new power gating scheme can be easily implemented by modifying the logic in the PG controller. This includes defining events that trigger the PG controller to make decisions and adding new signals between PG controllers if needed.

5.1 Basic Components

Fig. 1 shows the main components in *Agate* and how *Agate* is interfaced with Garnet. There are two main components in *Agate*, namely PG link (including the "PG inward" link and "PG outward" link in the figure) and PG controller. Detailed descriptions can be found in [4].

5.2 Power Gating Strategies

The key control logic of a power-gating scheme is extracted and placed in the power gating controller logic defined by `PGController_d.hh` and `PGController_d.cc` where state transition rules are defined.

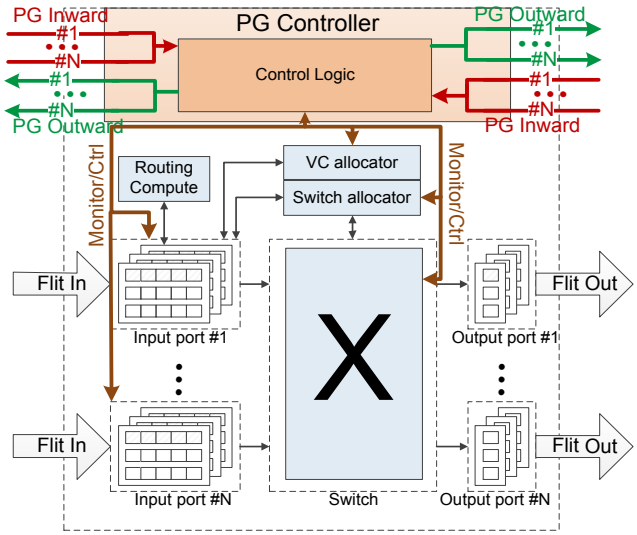

Fig. 1. *Agate* components and interface with Garnet router.

The PG controller of each NoC router is a decision block for controlling the power-gating of that router. It monitors the status of router components and various signals, and takes power-gating actions when certain events occur. While one can modify the processing logic in the controller, the essential functions of a PG controller must include the following:

- Update the power-gating state of neighboring routers (which is stored locally) upon receiving the sleep/active notifications from PG links;
- Make power gating decisions based on the status of the associated router and neighboring routers, as well as the signals from neighbor routers; and
- Send wakeup requests or sleep/active notifications through PG links to neighboring PG controllers.

5.3 Compatibility with more recent versions of the gem5 simulator

Agate is based on the version of the gem5 simulator which has a canonical 4-stage pipelined router. In more recent gem5 simulator versions, a two-stage speculative router is implemented (as of April 2016), along with some other non-NoC features. We might make *Agate* be compatible with (or integrate with) more recent gem5 in a future version. Before then, users who would like to use *Agate* in newer versions of gem5 can do so with the help of the list of new or changed files in Section 5.4 and the *Agate* component description in [4].

5.4 A list of new or changed files to implement *Agate* in gem5

Below is a list of new or changed files to implement *Agate* in gem5. We try to be as complete as we can so as to help implementing *Agate* in newer versions of gem5 by incorporating these changes. However, we have no control on how gem5 might evolve in the future. Thus, additional efforts may be needed if the targeted version of gem5 is much newer than the version which *Agate* is initially developed on.

1) Changed files in the fixed-pipeline folder (This folder contains the new source codes to support power gating in on-chip network):

```
./src/mem/ruby/network/garnet/fixed-pipeline/flit_d.*
./src/mem/ruby/network/garnet/fixed-pipeline/GarnetLink_d.*
./src/mem/ruby/network/garnet/fixed-pipeline/GarnetNetwork_d.*
./src/mem/ruby/network/garnet/fixed-pipeline/InputUnit_d.*
./src/mem/ruby/network/garnet/fixed-pipeline/NetworkInterface_d.*
./src/mem/ruby/network/garnet/fixed-pipeline/NetworkLink_d.*
./src/mem/ruby/network/garnet/fixed-pipeline/OutputUnit_d.*
./src/mem/ruby/network/garnet/fixed-pipeline/Router_d.*
./src/mem/ruby/network/garnet/fixed-pipeline/SWallocator_d.cc
./src/mem/ruby/network/garnet/fixed-pipeline/Switch_d.*
./src/mem/ruby/network/garnet/fixed-pipeline/VCallocator_d.cc
```

2) New files in the fixed-pipeline folder:

```
./src/mem/ruby/network/garnet/fixed-pipeline/flitPG_d.*
./src/mem/ruby/network/garnet/fixed-pipeline/PGController_d.*
./src/mem/ruby/network/garnet/fixed-pipeline/PGLink_d.*
```

3) Changed file:

```
./src/mem/ruby/network/garnet/NetworkHeader.hh
```

4) Changed file:

```
./configs/example/ruby_network_test.py
```

New codes are added to this python script to support the debugging output option.

5) Add the following two folders for DSENT:

```
./src/mem/ruby/network/dsent/
./DSENT/
```

These two directories are added to integrate the DSENT power model.

6) Changed files:

```
./src/cpu/testers/networktest/networktest.cc
./src/cpu/testers/networktest/networktest.hh
./src/cpu/testers/networktest/NetworkTest.py
```

New codes are added to this python script to support the debugging output option and new synthetic traffic patterns.

6. Contributors and Contact Infomation

Prof. Lizhong Chen (chenliz@oregonstate.edu) from the STAR Lab at the Oregon State University and Dr. Di Zhu (dizhu@usc.edu) from the SPORT Lab at the University of Southern California (USC) contributed significantly to the development of the *Agate* simulator. This work is also supported, in part, by USC SPORT Lab led by Prof. Massoud Pedram and USC SMART Interconnects Group led by Prof. Timothy M. Pinkston.

If you have questions regarding this simulator, please send us emails with the tag "[Agate]" in the email title (including the brackets). Note that general questions on gem5, Garnet or DSENT should be directed to the corresponding developers, manuals and online forums of these simulators rather than to us.

7. Acknowledgment

This work is supported, in part, by the National Science Foundation (NSF) grants CCF-1619456 and CCF-1321131, and the Software and Hardware Foundation of NSF.

References

[1] N. Agarwal, T. Krishna, L. Peh, N. K. Jha, "Garnet: A detailed on-chip network model inside a full-system simulator," International Symposium on Performance Analysis of Systems and Software (ISPASS), 2009.

[2] C. Bienia and K. Li, "Parsec 2.0: A new benchmark suite for chipmultiprocessors," in 5th Annual Workshop on Modeling, Benchmarking and Simulation, 2009.

[3] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basil, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 Simulator," Computer Architecture News, vol. 39, pp. 1-7, 2011.

[4] L. Chen, D. Zhu, M. Pedram, and T. M. Pinkston, "Simulation of NoC Power-Gating: Requirements, Optimizations, and the Agate Simulator", in Journal of Parallel and Distributed Computing (JPDC), 2016.

[5] C. Sun, C.-H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic, "DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling," In International Symposium on Networks-on-Chip, 2012.