NoCTweak: a Highly Parameterizable Simulator for Early Exploration of Performance and Energy of Networks On-Chip

Anh T. Tran and Bevan M. Baas

Department of Electrical and Computer Engineering
University of California - Davis, USA
{*anhtr, bbaas*}@*ucdavis.edu*

Abstract

As the number of processing elements (PE) on a single chip increases with each generation of CMOS technology, network on-chip (NoC) has become a de-facto communication fabric for these PEs. Due to high design and test costs for real many-core chips, simulators which allow exploring the best design options for a system before actually building it have been becoming highly necessary in system design and optimization flows. This paper presents *NoCTweak*, a highly parameterizable NoC simulator used for early exploration of performance and energy efficiency of on-chip networks. The simulator has been developed in SystemC, a C++ plugin, which allows fast modeling of concurrent hardware modules at the cycle-level accuracy. The statistic output results provided by the simulator are the average network latency, throughput, router power and energy per transferred data packet corresponding to a given network configuration, a certain traffic pattern and load. Area, timing and power of router components are post-layout data based on standard-cell libraries.

1 Introduction

Due to high design and test costs for real many-core chips, simulators, which allow exploring the best design options for a system before actually building it, have been becoming highly necessary in system design and optimization flows. Simulators are normally developed using high-level languages such as C/C++ and Java which run much faster than RTL modeling languages such as Verilog and VHDL. Besides that, high-level languages allow programmers to build highly flexible simulators which are easy to tweak their parameters for fast exploration of design trade-offs. In this paper, we present *NoCTweak*, an open-source NoC simulator for early exploration of performance and energy efficiency of on-chip networks. The simulator has been developed using SystemC [1], a C++ plugin, which allows fast modeling of concurrent hardware modules at the cycle-level accuracy.

This paper is organized as follows: Section 2 presents the network architecture and configurable parameters for the routers supported by our NoCTweak simulator. Statistic output results reported by the simulators are described in Section 3. A few common network configuration examples run by the simulator is shown in Section 4. Section 5 reviews related work and, finally, Section 6 concludes this paper.

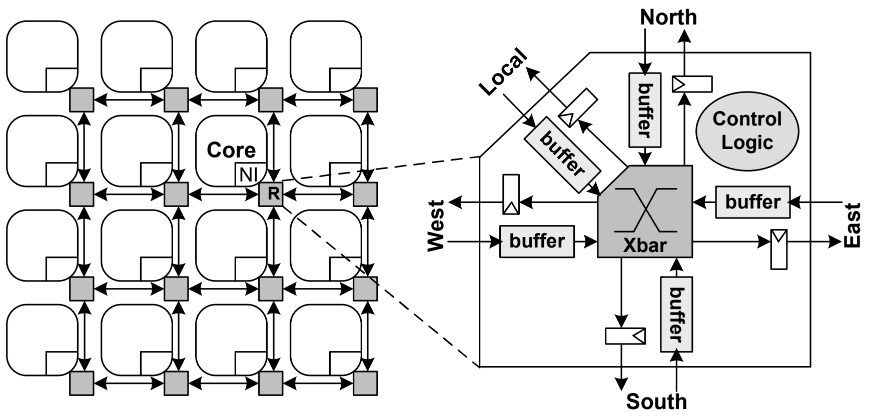

Figure 1: A simulated platform includes multiple cores interconnected by a 2-D mesh network of routers

2 Configurable Simulation Parameters

Fig. 1 depicts a platform including multiple cores interconnected by a 2-D mesh network of routers which is simulated by NoCTweak version 1.0 (current version). Each node consists of a processor (core + NI) and an associated router. Each router connects with four nearest neighboring routers forming a 2-D mesh network [2]. Each processor core generates data packets and injects into the network through its router. Packets are routed on the network of routers by a selected routing algorithm to their destinations at which the packets are immediately consumed.

The users can choose network parameters through changing their default values in the source code before compiling or through a terminal (command line window) when running the simulator. Below are lists of network and router parameters which can be set for the simulator (these options are also displayed in the command line window with the command "`./noctweak -help`"):

Listing 1: Platform Options

```
-platform [option]  application traffic simulated on this platform.
                      option = synthetic: a synthetic traffic pattern (default)
                      option = embedded: an embedded application trace
-seed [value]       random seed for the simulation.
                      (the same random seed will drive the same output results for
                      the same network configuration. It's used for easier debugging.
                      Default value = system time.)
-log [filename]     log file for simulation outputs
-vcd [filename]     VCD file for signal waveform traces
-simmode [option]   simulation mode (packet or cycle)
-simtime [value]    simulation running time
                      value = N. Default = 100,000.
                      if simmode option = packet: stop simulation after transferring N packets
                      if simmode option = cycle: stop simulation after running N clock cycles
-warmtime [value]   warmup time for the network to become stable
                      value = M (M < N). Default = 10,000.
                      if simmode option = packet: do not consider the first M received packets
                      if simmode option = cycle: warmup time is M clock cycles
```

Listing 2: Synthetic Traffic Patterns

```
-dimx [value]       X dimension length of the 2-D mesh network. Default value = 8.
```

```
-dimy [value]        Y dimension length of the 2-D mesh network. Default value = 8.
-traffic [option]    synthetic traffic patterns used for the simulation.
                       option = random: uniform random (default)
                       option = transpose: transpose
                       option = bitc: bit-complement
                       option = bitr: bit-reverse
                       option = tornado: tornado
                       option = shuffle: bit-shuffle
                       option = rotate: bit-rotate
                       option = neighbor: nearest neighbor traffic
                       option = regional: communication distance <= 3
                       option = hotspot: central or corner hot spots
-nhs [value]         the number of hot spots. Default = 4.
-hstype [option]     hot-spot type
                       option = central: hot spots at the central cores
                       option = corner: hot spots at the corners (default)
-percent [value]     percentage of traffics going to neighboring or regional or hotspot cores
```

Listing 3: Embedded Application Traces

```
-appfile [option]    application's task communication graph used for the simulation.
                       option = vopd.app: video object plan decoder with 16 tasks (default)
                       option = mms.app: multimedia system with 25 tasks
                       option = mwd.app: multi-window display with 12 tasks
                       option = wifirx.app: WiFi baseband receiver with 25 tasks
                       option = cavlc.app: H.24 CAVLC encoder with 16 tasks
                       option = mpeg4.app: MPEG4 decoder with 12 tasks
                       option = vce.app: video conference encoder with 25 tasks
                       option = autoindust.app: E3S auto-indust benchmark with 24 tasks
                       option = consumer.app: E3S consumer benchmark with 12 tasks
                       option = telecom.app: E3S telecom benchmark with 30 tasks
-mapping [option]    mapping algorithm used to map the task graph to the processor array
                       option = random: random mapping
                       option = nmap: near-optimal mapping using the NMAP algorithm
```

Listing 4: Traffic Options

```
-fir [value]            flit injection rate (number of flits injected by each core per cycle)
                          0 < fir <= 1. Default = 0.2
-dist [option]          probability distribution of the period between two injected packets
                          option = exponential: exponential distribution (default)
                          option = identical: identical distribution
-plengthtype [option]   packet length is fixed or variable
                          option = fixed: fixed packet length (default)
                          option = variable: variable packet length
-plength [value]        the number of flits per packet.
                          (only for the fixed packet length option. Default = 5.)
-plengthmin [value]     the minimum number of flits per packet
                          (only for the variable packet length option. Default = 2.)
-plengthmax [value]     the maximum number of flits per packet
                          (only for the variable packet length option. Default = 10.)
```

Listing 5: Router Settings

```
-router [option]     the simulated router
                       option = wh: wormhole router (default)
```

```
                          option = vc: virtual-channel router
                          option = roshaq: RoShaQ share-queues router
                          option = bufferless: bufferless router
                          option = cs: circuit-switched router
-pptype [value]         pipeline type and the number of pipeline stages. Default = 3 stages
-bsize [value]          buffer depth (2, 4, 8, 16, 32 flits). Default = 4 flits.
-sbsize [value]         shared-buffer queue depth (2, 4, 8, 16, 32 flits). Default = 4 flits.
-nvc [value]            the number of virtual-channel buffers per input port. Default = 2 queues.
-nsb [value]            the number of shared-buffer queues in RoShaQ routers. Default = 5 queues.
-routing [option]       routing algorithm
                          option = xy: XY dimension-ordered routing (default)
                          option = nfminimal: Negative-First minimal adaptive routing
                          option = wfminimal: West-First minimal adaptive routing
                          option = nlminimal: North-Last minimal adaptive routing
                          option = oeminimal: Odd-Even minimal adaptive routing
                          option = table: lookup table based routing
-outsel [option]        choose an output port among multiple ones returned by an adaptive routing
                          option = xyordered: the X dimension first (default)
                          option = nearestdim: the dimension nearest to the destination first
                          option = farthestdim: the dimension farthest to the destination first
                          option = roundrobin: round-robin among output ports
                          option = credit: the output port having the highest credit first
-sa [option]            switch arbitration policy
                          option = rr: round-robin (default)
                          option = oldest: oldest first
                          option = takeall: winner takes all (only for virtual-channel routers)
                          option = islip: iSLIP based algorithm (only for virtual-channel routers)
-vca [option]           virtual-channel allocation policy (only for virtual-channel routers)
                          option = rr: round-robin (default)
                          option = oldest: oldest first
                          option = islip: iSLIP based algorithm
-llength [value]        inter-router link length (in um). Default = 1000 um.
```

Listing 6: Environmental Settings

```
-technode [value]       CMOS technology process (90, 65, 45, 32, 22 nm). Default = 65 nm.
-freqmode [option]      clock frequency setting
                          option = fixed: fixed clock frequency (in MHz)
                          option = max: the maximum clock frequency supported by the router
-freq [value]           for fixed clock frequency (in MHz). Default = 1000 MHz.
-volt [value]           supply voltage (in V). Default = 1.0 V.
```

3 Statistic Outputs

Simulation's statistic results are displayed in the command line window and also be written into a log file for later use. Activities of circuit components of all routers in the network are tracked for router power and energy evaluation [3]. These activities are also recorded into another log file for later check.

3.1 Network Latency

Latency of a packet is measured from the time its head flit is generated by the source to the time its tail flit is consumed by the destination. Clearly, packet latency also includes the time when packet waits at the source queue due to network congestion. When a processor receives a packet, it subtracts the packet's generating time (in the packet's head flit) from the current simulation time to get the packet latency. Network latency is the mean of latency of all packets transferred by the network. For more accuracy, we only consider packets received after the warmup time.

Let L_{ij} be the packet latency of packet j and N_i be the number of packets received by processor i (after the warmup time), then the average network latency is given by:

$$L_{avg} = \frac{1}{N} \sum_{i=1..N} \left(\frac{1}{N_i} \sum_{\forall j} L_{ij} \right) \tag{1}$$

where N is the number of processors in the platform.

3.2 Network Throughput

Network throughput is defined as the rate at which the network can successfully accept and deliver the injected packets. Let T_{sim} and T_{warm} be the simulation and warmup time, then the average network throughput (in packets per unit time per node) is given by:

$$T_{avg} = \frac{1}{N(T_{sim} - T_{warm})} \sum_{i=1..N} N_i \tag{2}$$

Given a clock frequency and a packet length, we easily drive the network latency in terms of cycles or seconds and the network throughput in terms of packets per cycle or packets per second or flits per cycle or flits per second. All these terms are shown in the output results, hence the user can choose any terms suitable for her needs.

3.3 Power Consumption

RTL designs in Verilog of all router components were synthesized with Synopsys Design Compiler and placed & routed with Cadence SoC Encounter using a 65 nm CMOS standard cell library. Post-layout power data of these components are fed to the simulator for power and energy estimation based on the activities of components while running a certain traffic pattern. Let $P_{act,j}$ and $P_{inact,j}$ be post-layout active power and inactive power of component j at 1.0 V and 1.0 GHz; let α_{ij} be active percentage of component j in router i (after the warmup time), then the average power of router i is:

$$P_i = \sum_{\forall j} [\alpha_{ij} P_{act,j} + (1 - \alpha_{ij}) P_{inact,j}] \tag{3}$$

Hence, the average router power at 1.0 V and 1.0 GHz is given by:

$$P_{avg} = \frac{1}{N} \sum_{i=1..N} P_i = \frac{1}{N} \sum_{i=1..N} \sum_{\forall j} [\alpha_{ij} P_{act,j} + (1 - \alpha_{ij}) P_{inact,j}] \tag{4}$$

Router power at a certain supply voltage and a given clock frequency is scaled from the power calculated above.

3.4 Energy Consumption

Average energy dissipated by each router after warming up is:

$$E_{avg} = P_{avg}(T_{sim} - T_{warm}) \tag{5}$$

Hence, the average energy dissipated per packet by each router is given by:

$$E_p = \frac{E_{avg}}{N_p} = \frac{(T_{sim} - T_{warm})}{NN_p} \sum_{i=1..N} \sum_{\forall j} [\alpha_{ij} P_{act,j} + (1 - \alpha_{ij}) P_{inact,j}] \tag{6}$$

where N_p is the total number of packets transferred on the network and is given by $N_p = \sum_{i=1..N} N_i$.

Similar to router power, energy can be scaled correspondingly to the supply voltage and clock frequency. Power and energy can also scaled to a given CMOS process node from the data at 65 nm CMOS based on the scaling rule described in the book by Rabaey *et al.* [4]; however, due to the differences in technology factors of standard cells made by different vendors even at the same CMOS technology node, we recommend using post-layout data of router components according to a certain CMOS cell library for getting accurate results rather than only naively scaling to that technology node [5]. If used for relative comparison among router designs, NoCTweak can be incorporated with the ORION tool [6, 7] for easily obtaining the power data at different CMOS nodes based on its computational power models although they may be far from accurate compared to the post-layout and real chip data.

4 Simulation Examples

We show here the statistic results on network latency, throughput, router power and energy per packet reported by NoCTweak for a few common network configurations. Wormhole routers with 3-pipeline stages, round-robin arbiters and 1000-µm links are used in all examples. Assuming the technology node is 65 nm CMOS and the network operates at 1.0 V and 1.0 GHz. Traffic pattern is *uniform random* with packet inter-injection time has an exponential distribution and packet length is ten flits. Each simulation runs in 100,000 cycles with 20,000 cycles for warmup.

4.1 Different Network Sizes

Listing 7: Running NoCTweak Simulator In a Terminal

```
./noctweak -seed 1234 -volt 1.0 -freqmode fixed -freq 1000 -dimx 8 -dimy 8
-pptype 3_1 -platform synthetic -traffic random -simmode cycle -sim 100000
-warm 20000 -routing xy -outsel credit -bsize 8 -plength fixed -length 10
-sa rr -llength 1000 -fir 0.30 -log output.log -vcd waveform.vcd
```

For running NoCTweak, we open a terminal and type a command similar to the one in Listing A.7 above. This command runs a simulation for a 8x8 2-D mesh network of 3-stage wormhole routers with 8-flit buffers, XY routing and round-robin switch arbitration at a flit injection rate of 0.30 flits/cycle/node over *uniform random* synthetic traffic. The results will be written into the “output.log” file and signal waveform traces will be recorded in the “waveform.vcd” file.

In this example, we simulate the performance, power and energy consumption of the same router in different network sizes. Four network sizes considered are 4×4, 6×6, 8×8 and 10×10. To change network

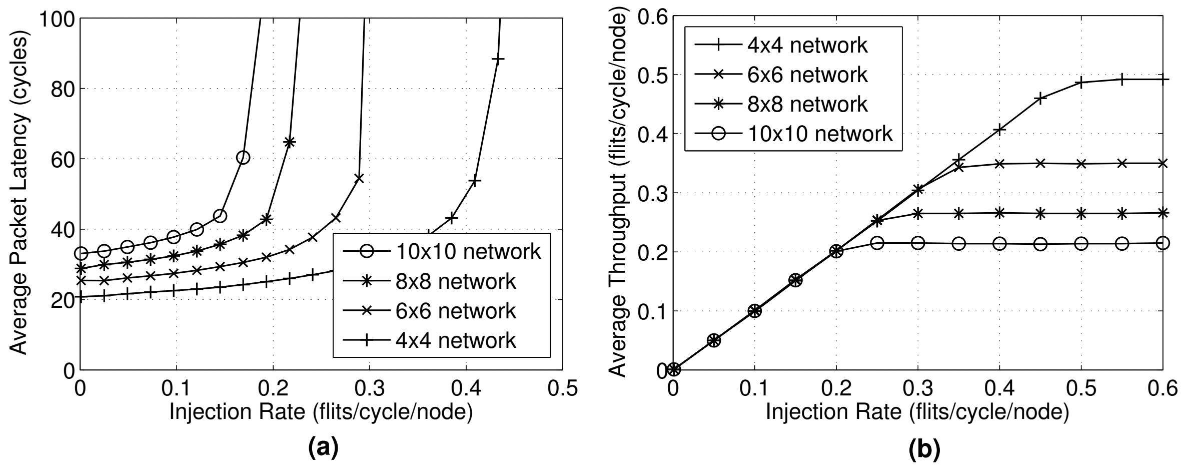

Figure 2: Performance of the networks in different sizes: a) Average packet latency vs. flit injection rate; b) Average network throughput vs. flit injection rate.

Table 1: Performance, saturation power and energy of routers in networks with different sizes

Buffer Size	Zero-Load Latency (cycles)	Saturation Throughput (flit/cycle/node)	Saturation Power (mW/router)	Saturation Energy (pJ/packet/router)
4×4 network	20.79	0.492	10.150	13.061
6×6 network	25.41	0.349	9.611	7.794
8×8 network	28.83	0.265	9.085	5.361
10×10 network	33.09	0.214	8.769	4.129

size, we adjust the values of "dimx" and "dimy" in Listing A.7. For each run, we change the value of "fir" so that we can get the results of network latency, throughput, router power and energy corresponding to various flit injection rates for quantitative comparisons.

Fig. 2 shows the network latency and throughput of wormhole routers in different network sizes. All routers have the same buffer size of 8 flits per input port. As shown, increasing network size increases network latency and reduces network throughput. This is because, over *random* traffic, a larger network size causes longer the average source-destination distance hence the packets would take more cycles to travel to their destinations given the same router design. In the same effect, because packets must travel on more immediate routers causing more network congestion hence reducing the overall network throughput. Therefore, a network with larger size would saturate sooner a smaller one. For reference, Column 2 and 3 in Table 1 lists the absolute values of zero-load latency and saturation throughput of networks with different sizes.[1]

Router power and energy per packet corresponding to various injection rates of networks are shown in Fig. 3. Router consumes more power when the injection rate increases because the router is more active. When the network becomes saturated, router's activities also become stable hence router power no longer increases and is stable at a value called saturation power. At the first glance, when the network load is

[1]Because the simulator cannot run with flit injection rate be equal to zero (which means there is no packet injected into the network), hence the zero-load latency is taken at an very low injection rate of 0.001 flits/cycle/node.

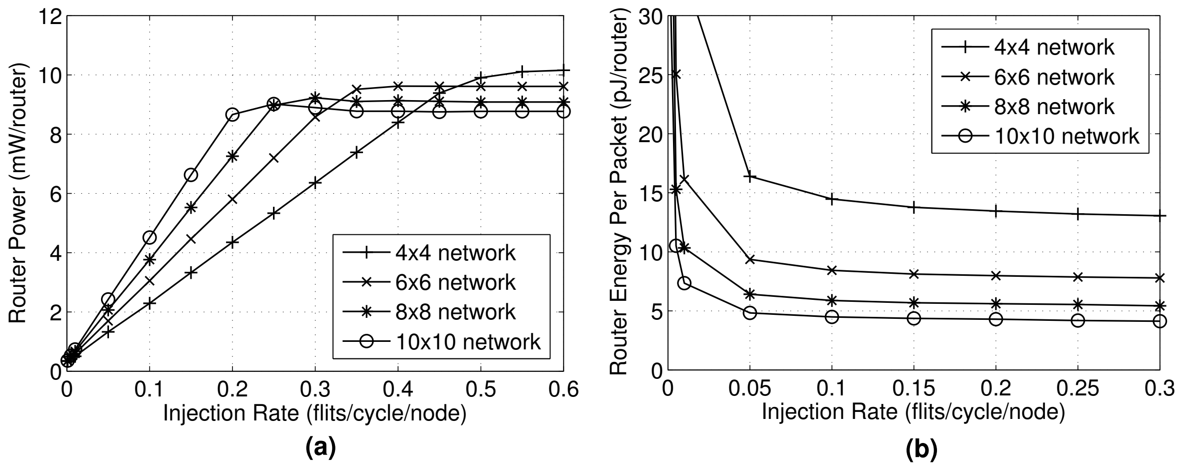

Figure 3: Power and energy consumption of routers in different network sizes: a) Average router power vs. flit injection rate; b) Average energy per packet vs. flit injection rate.

low (e.g. less than 0.2 flits/cycle/node), at the same injection rate, routers in a network with larger size consume more power than in a smaller one. This is because although having the same injection rate, the larger network has larger number of processors hence inject more packets into the network. Therefore, each router would have more packets to handle thus is more active and consumes more power. However, routers in a larger network size consume less saturation power than in a smaller one because they saturate sooner as explained in the saturation throughput of networks.

Lower saturation power consumption along with the larger number of packets transferred in a larger network size make its routers consume less average energy per packet than routers in a network with smaller size as shown in Fig. 3(b). Saturation power and energy per packet of routers in different network sizes are listed in Column 4 and 5 of Table 1.

4.2 Different Buffer Depths

In this example, we consider the effect of buffer depth on performance and energy of routers in the same network size of 8×8. Four buffer depths considered are 2, 4, 8 and 16 flits per buffer queue. To change buffer size of router, we adjust the value of "bsize" in Listing A.7. Network latency and throughput of routers over the *random* traffic pattern is shown in Fig. 4. Clearly, increasing buffer depth improves network performance as shown in the figure. Because the router has 3 pipeline stages, routers with at least 5 flits per buffer have the same zero-load network latency. Due to not enough buffers to cover round-trip flow control signaling, the router with buffer depth of 2 flits achieves the worst network performance.

Increasing buffer depth from 2 flits to 4 and 8 flits improves saturation network throughput by 2.1 and 3.4 times, and reduces zero-load latency by 26.9% and 35.8%, respectively; while increasing from 8 flits to 16 flits only improves 1.2 times in throughput and has the same zero-load latency. Zero-load latency and saturation throughput of routers are listed in Column 2 and 3 of Table 2.

Due to high power buffer cost, increasing buffer depth dramatically increases the overall router power (note that each router has five input buffers). As shown in Fig. 5, increasing buffer depth from 2 flits to 4, 8 and 16 flits increases the saturation power by 2.2, 4.1 and 7.5 times. However, because larger buffer depth achieves higher network throughput which allows transferring more packets in a certain time window, the

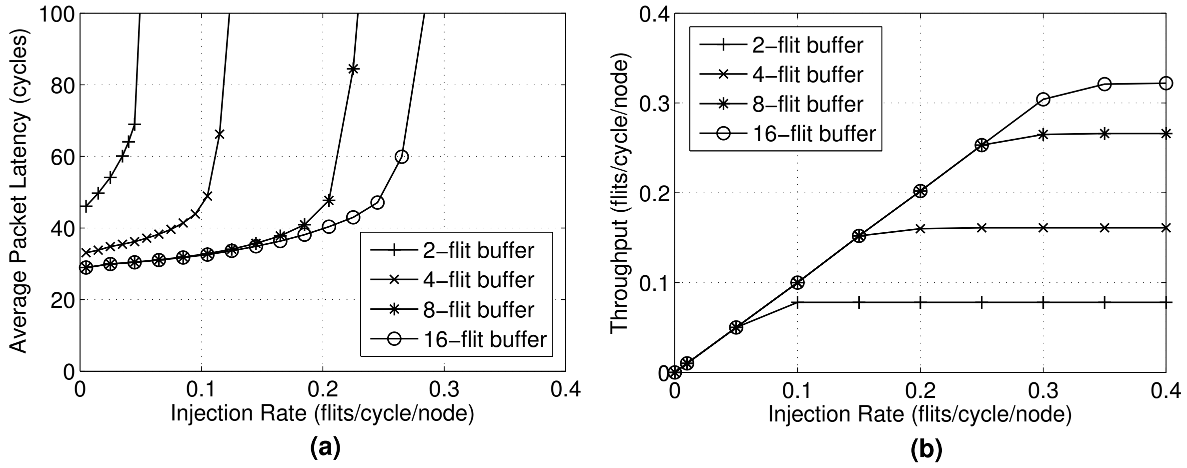

Figure 4: Performance of the networks of routers with different buffer depths: a) Average packet latency vs. flit injection rate; b) Average network throughput vs. flit injection rate.

Table 2: Performance, saturation power and energy of routers with different buffer depths

Buffer Size	Zero-Load Latency (cycles)	Saturation Throughput (flit/cycle/node)	Saturation Power (mW/router)	Saturation Energy (pJ/packet/router)
2 flits/buffer	44.93	0.078	2.195	4.419
4 flits/buffer	32.84	0.162	4.729	4.572
8 flits/buffer	28.83	0.265	9.085	5.361
16 flits/buffer	28.83	0.319	16.524	8.099

router with 4 flits per buffer consumes almost the same saturation energy per packet as the one with 2 flits per buffer. Router with 8 and 16 flits per buffer are 17.3% and 77.1% higher energy per packet compared to the router with 4 flits per buffer. Router saturation power and energy per packet are listed in Column 4 and 5 of Table 2.

5 Related Work

A few on-chip network simulators have been developed recently. Booksim developed in C++ by Jiang *et al.* allows simulating on-chip networks in a broad range of topology, buffer size, routing algorithm, arbitration policy configurations [8]. Currently, Booksim only supports virtual-channel (VC) routers with synthetic traffic patterns. The output results are only network latency and throughput versus an injection rate. Our NoCTweak supports multiple router types (wormhole, virtual-channel, shared queues, bufferless, circuit-switched) over both synthetic traffic and embedded application patterns. Moreover, it also reports power and energy consumption of routers in the network at different CMOS technologies, operating voltages and clock frequencies.

NIRGAM developed by Jain *et al* in SystemC is a NoC simulator for mesh and torus topologies [9]. It can simulate different routing algorithms, buffer depths and configurable traffic patterns. Currently, it supports VC routers and reports only network performance. Similarly, Noxim was also developed in SystemC

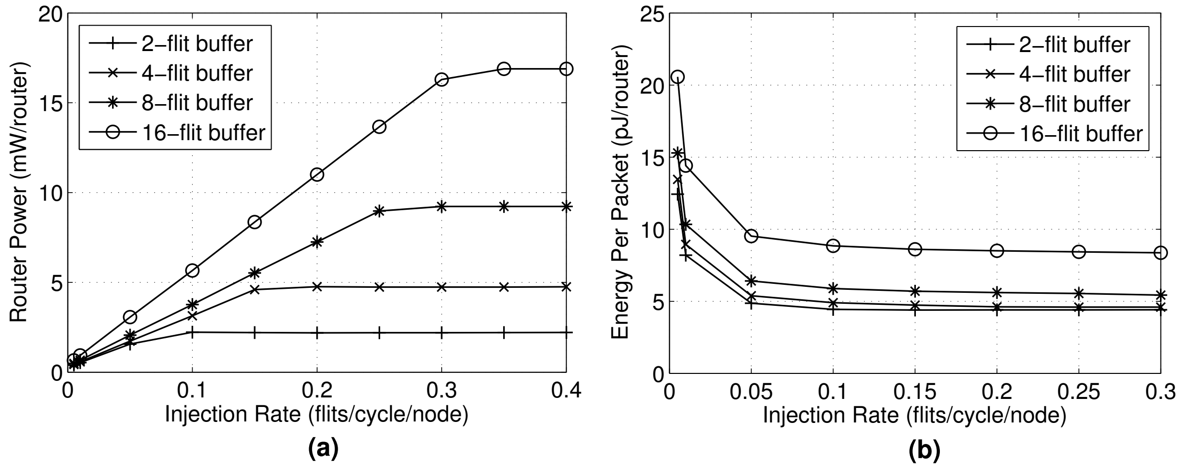

Figure 5: Power and energy consumption of routers with different buffer depths: a) Average router power vs. flit injection rate; b) Average energy per packet vs. flit injection rate.

by Palesi *et al.* [10], it allows computing router power and energy based on the ORION tool [7]. Noxim only supports wormhole routers over synthetic traffic patterns. It allows changing simulation parameters via a command line which was adopted by our NoCTweak. NoCTweak uses post-layout timing and power data from commercial CMOS standard-cell libraries which show highly accurate within 5% compared to measurement results on a chip [11, 12].

Al-Nayeem and Islam developed gpNoCsim in Java which supports butter fly, fat tree, torus and mesh networks [13]. Simulators having similar features are NoCsim by Jones [14], NoCSim by Grecu *et al.* [15] and Nostrum by Lu *et al.* [16]. These simulators only support VC routers with synthetic traffics and do not report router power and energy. Ocin_tsim by Prabhu [17], GARNET by Agarwal *et al.* [18], SICOSYS by Puente *et al.* [19] and Darsim by Lis *et al.* [20] support computing router power but based on the ORION model [7] which may be far from accurate compared to the post-layout power data. They, however, can incorporate with full-system multicore simulators to run parallel benchmarks such as SPLASH-2 [21] or PARSEC [22]. Supporting these benchmarks in NoCTweak is left for our future work.

6 Summary

We have described NoCTweak, a simulator for early exploration of performance and energy efficiency of networks on-chip. The simulator is an open-source tool based on SystemC, a C++ plugin, which is more flexible and provides higher simulation speed than RTL simulators. The tool is highly parameterizable allowing users to setup and simulate a broad range of network configurations such as router type, network size, buffer size, routing algorithm, arbitration policy, pipeline stages, supply voltage, clock frequency, traffic pattern, packet length, injection rate, simulation and warmup times. The statistic output results provided by the simulator are the average network latency, throughput, router power and energy per transferred packet. Area, timing and power of router components are post-layout data based on a commercial 65 nm CMOS standard-cell library.

Acknowledgments

This work was supported by a VEF Fellowship, SRC CSR Grant 1659 and SRC GRC Grant 1971, NSF Grants 0903549, 1018972 and CAREER Award 0546907, ST Microelectronics, UC Micro and Intel.

References

[1] Accellera, “Download SystemC,” Online, http://www.accellera.org/downloads/standards/systemc/.

[2] A. T. Tran and B. M. Baas, “DLABS: A dual-lane buffer-sharing router architecture for networks on chip,” in *IEEE Workshop on Signal Processing Systems (SiPS)*, Oct. 2010, pp. 327–332.

[3] A. T. Tran and B. M. Baas, “RoShaQ: High-performance on-chip router with shared queues,” in *IEEE International Conference on Computer Design (ICCD)*, Oct. 2011, pp. 232–238.

[4] J. M. Rabaey, A. Chandrakasan, and B. Nikolić, *Digital Integrated Circuits: A Design Perspective*, Prentice-Hall, New Jersey, U.S.A, second edition, 2003.

[5] A. T. Tran et al., “A low-cost high-speed source-synchronous interconnection technique for GALS chip multiprocessors,” in *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2009, pp. 996–999.

[6] H.-S. Wang et al., “Orion: a power-performance simulator for interconnection networks,” in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2002, pp. 294 – 305.

[7] A.B. Kahng et al., “ORION 2.0: A fast and accurate noc power and area model for early-stage design space exploration,” in *Design, Automation Test in Europe Conference Exhibition (DATE)*, Apr. 2009, pp. 423 –428.

[8] N. Jiang et al., “BookSim interconnection network simulator,” Online, https://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/BookSim.

[9] L. Jain et al., “NIRGAM - a simulator for noc interconnect routing and application modeling,” Online, http://nirgam.ecs.soton.ac.uk/.

[10] R. Palesi et al., “Noxim - the noc simulator,” Online, http://noxim.sourceforge.net/.

[11] A. T. Tran et al., “A GALS many-core heterogeneous DSP platform with source-synchronous on-chip interconnection network,” in *ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, May 2009, pp. 214–223.

[12] A. T. Tran et al., “A reconfigurable source-synchronous on-chip network for GALS many-core platforms,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 29, no. 6, pp. 897–910, Jun. 2010.

[13] A. Al-Nayeem and T. Z. Islam, “gpNoCsim: General purpose simulator for network-on-chip,” Online, http://www.buet.ac.bd/cse/research/group/noc/index.html.

[14] M. Jones, “NoCsim : a versatile network on chip simulator,” Online, https://circle.ubc.ca/handle/2429/16550.

[15] C. Grecu et al., “A flexible network-on-chip simulator for early design space exploration,” in *Microsystems and Nanoelectronics Research Conference (MNRC)*, Oct. 2008, pp. 33 –36.

[16] Z Lu et al., “NNSE: Nostrum network-on-chip simulation environment,” *Swedish System on Chip*, 2005.

[17] S. Prabhu et al., “Ocin tsim- DVFS aware simulator for NoCs,” Online, 2009.

[18] N. Agarwal et al., “GARNET: A detailed on-chip network model inside a full-system simulator,” in *IEEE Intl. Symp. on Performance Analysis of Systems and Software (ISPASS)*, Apr. 2009, pp. 33 –42.

[19] V. Puente et al., “SICOSYS: an integrated framework for studying interconnection network performance in multiprocessor systems,” in *Euromicro Workshop on Parallel, Distributed and Network-based Processing*, 2002, pp. 15 –22.

[20] M. Lis et al., “Darsim: A parallel cycle-level NoC simulator,” Online, http://dspace.mit.edu/handle/1721.1/59832.

[21] S. C. Woo et al., “The SPLASH-2 programs: characterization and methodological considerations,” in *intl. symp. on Computer architecture (ISCA)*, 1995, pp. 24–36.

[22] C. Bienia et al., “The PARSEC benchmark suite: characterization and architectural implications,” in *Intl. Conf. on parallel architectures and compilation techniques (PACT)*, 2008, pp. 72–81.