Extending Gem5-Garnet for Efficient and Accurate Trace-driven NoC Simulation

Ren-Min Li
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan

Chung-Ta King
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
king@cs.nthu.edu.tw

Bhaskar Das
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
das.bhaskar.1981@gmail.com

ABSTRACT

Cycle-accurate full-system simulators, such as Gem5, are indispensable for advanced architecture designs. However, as the complexity of the system architectures rises, exploiting the design space with such simulators is becoming very time-consuming. A viable alternative is to separate the system component of interest, e.g. the network-on-chip (NoC), from the simulator and use a trace-driven approach to the component simulator. Although trace-driven simulation allows fast evaluation of design alternatives, the performance accuracy will often be sacrificed. To overcome the problem, techniques based on dependence-aware traces have been proposed. However, a considerable time and storage space is required to accurately identify and capture the dependence information between simulated events. A possible solution is to generate approximate dependences on-the-fly during simulation. In this paper, we study the effectiveness of a trace-based simulation technique based on the NoC simulator of Gem5, Garnet, that exploits and approximates trace dependencies during simulation to produce accurate performance results. We modify Garnet to be dependency-aware and use Gem5 as the ground truth for comparison. Based on our evaluations, the dependency-aware Garnet can keep the performance differences within 3.22%, while a trace-based Garnet may result in errors as high as 14.97%.

CCS Concepts

•Network On Chip Simulator → General; •Full-system Simulator → *General;*

Keywords

Trace-driven simulation, network-on-chip, performance evaluation, multicore architecture

1. INTRODUCTION

The network-on-chip (NoC) is an important component in the integrated circuit, components rely on it for communication. Researchers implement various simulation techniques to evaluate and compare the design aspect of the NoC. The execution-based simulation technique produces accurate results but is very slow. On the other hand, trace-based simulation techniques are fast, but the results they produce are not as accurate as the execution-based simulation.

To evaluate the complex design of a system, cycle-accurate full-system simulators, like Gem5 [2], take a vital role. Garnet [1] is a detailed cycle-accurate interconnection network model inside Gem5. A full-system simulator provides the virtual hardware and simulates the operations of the whole computer system, including processor cores, peripheral devices, memories, and interconnected networks. Cycle-accurate simulators produce accurate performance data of the system by executing the operations of the hardware on a cycle-by-cycle basis. However, the time consumption of the cycle-accurate simulators increases considerably with the increase in the complexity of the system architectures [3].

Trace-driven simulation techniques are widely used to explore the design space of the system components, such as NoC, graphics processing unit (GPU), and caches. The simulations of processor cores are replaced with execution traces in the trace-driven simulation to reduce the simulation time. Therefore, navigation through the design space becomes manageable to find the best design option. However, trace-driven simulations often contain substantial errors, because the timings of the trace events are taken from the trace-generating machines and remain fixed during simulation. The parameters of the target system component can be varied to find the best design options. Hence, the timings of the events in the target component will also change. Therefore, trace-driven simulation using the fixed timings obtained from the trace-generating machine leads to inaccurate performance measures. The fidelity of the trace-driven simulation can be improved by incorporating the time information of the trace events according to the responses generated by the simulated component [3, 4, 5, 6]. The dependencies among the trace events are captured with dependency-aware trace-driven simulation technique and the simulated time is adjusted based on the inter-plays of these events. However, considerable time and space are required to detect trace-dependencies, generating the trace file, processing the trace file, and storing the trace file with dependency information.

In this paper, we propose a dynamic dependency generation technique, which generates and approximates trace-dependencies dynamically based on given machine models at

NoCArc'16, October 15 2016, Taipei, Taiwan
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-4792-1/16/10…$15.00
DOI: http://dx.doi.org/10.1145/2994133.2994140

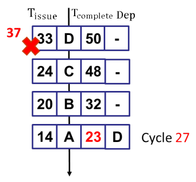

Figure 1: On-line timing adjustment according to event dependencies

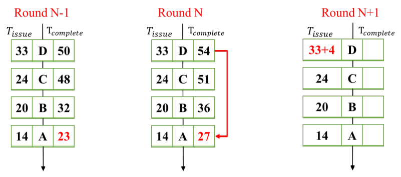

Figure 2: Off-line iterative timing adjustment according to event dependencies

the simulation time. The proposed dynamic dependency generation technique addressed the cumbersome time-consuming trace generation process by using the original trace file in the simulation with or without modification. We have modified Garnet, the NoC simulator of Gem5, to be dependency-aware and used it for dynamic dependency generation. The proposed dependency-aware Garnet is evaluated for the NoC design space exploitation and its performance is analyzed with the ground truth results obtained from Gem5. The main contribution of the study can be summarized as follows.

- A trace-driven, dependency-aware NoC simulator based on Garnet is proposed
- The proposed technique produces accurate results and runs faster than a full-system simulator
- Investigate the effectiveness of dynamic dependency generation for dependency-aware NoC simulation

2. RELATED WORKS

Many researchers have worked on the dependency-aware trace-driven simulation techniques. Nitta et al. [6] observe that using only the total ordering information to predict system-level performance will lead to erroneous conclusions. The study demonstrates the importance of dependencies in network trace events and proposes an inferring algorithm to capture the dependencies based on a series of full-system runs using different latencies. The proposed approach is shown to yield more accurate performance results for message passing systems.

Tsim [3] is a multicore processor architecture simulator that can capture synchronization operations such as barriers and locks in shared-memory systems. It can predict relative timing performance trends with changing parameters. The simulator can simulate with a considerable less time and use much less memory when compared with a full system simulator.

A trace-based network simulation platform, Netrace [4], is studied by Hestness et al. The fidelity of a trace-based NoC simulation is increased by the proposed research work by including dependency information between network messages. Application runtime and other design evaluation can be performed by Netrace.

The work presented by Lee et al. [5] focuses on studying the execution and simulation of out-of-order processors. For trace-driven simulations, the authors proposed three strategies to measure the impact of a long latency memory access on the performance of the superscalar processor. One of the important strategy, Independent cache miss model strategy, considers all L1 cache misses in the reorder buffer as independent of each other and handles them without considering any outstanding cache misses. The cache miss events, those occur outside the reorder buffer are considered to be dependent on those present in the reorder buffer. We have implemented this strategy in the dependence-aware Garnet.

3. IMPLEMENTATION

Garnet, standalone NoC simulator, integrated into the Gem5, a full-system simulation framework. The original Garnet uses synthesis traffic to evaluate NoC design. Network testers inject packets into the Garnet, based on some probability distribution. Each packet, in the Garnet, will ultimately go to one of the directories and the one-way network latency can be recorded. In this section, we will single out the Garnet simulator, called Garnet Standalone, and extend the functionalities of Garnet to be trace-driven and dependency-aware, and examine the issues related to it.

3.1 Dynamic Dependency Detection

Garnet after receiving memory requests for cache misses from the L1 caches, simulates the operations in the NoC and delivers the requests to the other end of the NoC, which may consist of last-level caches (LLCs), directories, and/or DRAM memory controllers (MCs). Ruby, the memory model present in the Gem5, will simulate the operations in the memory hierarchy, including cache coherence operations, and inject response messages into the NoC in reply to the requests received from the L1 caches. Garnet accepts inputs from various sources, such as L1 caches, last-level caches, memory controllers, and directories, and those messages may have dependencies. We have to solve the problems mentioned below before implementing the standalone dependency-aware Garnet.

Trace generation is our first problem. We have to generate traces for all possible input sources of the Garnet. Traces may be dumped from Gem5 for L1 caches, memory controllers, last-level caches, and directories, based on the architecture of the memory hierarchy. To simplify the trace generation efforts and shorten the simulation time, an implementation decision has to be made regarding which sources should use traces and which sources can be replaced by simple architecture models.

The second problem analyzes the process of determining the dependencies among the trace events. This problem can be divided into two categories. The first category of the problem deals with the chain of messages generated in response to a memory request from the L1 cache and such relationship must be extracted and recorded in the trace file. This process involves tracking a memory request from the high-level packages down to flits and back to the high-level packages. The second category of the problem deals with the architecture dependencies and programmatic dependencies [4] among the trace events from the cores,

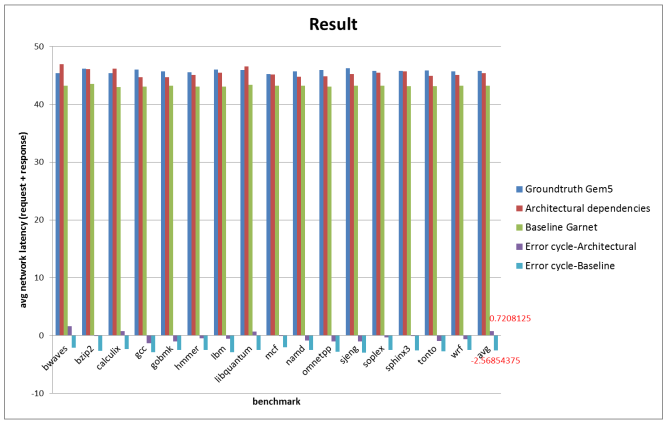

Figure 3: Average network latency without using programmatic dependencies

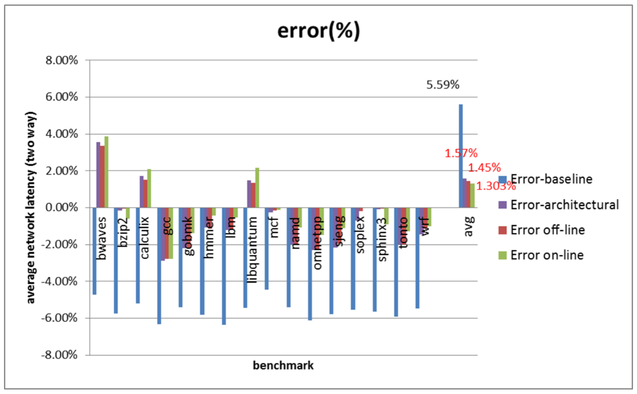

Figure 4: Error rate with architectural and programmatic dependencies

which occur indirectly through instructions. Tracking the dependencies among the instructions within the same thread is a difficult work, but it becomes more challenging if there are synchronizations among the threads executing on different cores. An intuitive idea is to track and record the def-use chain of the executed instructions in Gem5, which requires quite an effort in dumping the traces and also increases the trace size. However, a more suitable approach is to infer or approximate the dependencies among the instructions and generate the trace events from the cores.

The third problem deals with the different types of information those are included in the dump. Normally a trace event for NoC can be characterized by its source, destination, and cycle time of injection. To mimic the operations in Gem5, the size of the message need to be determined as it will affect the buffering time in the routers. After generating dependency-aware traces from Gem5, the next problem is to modify Garnet Standalone to make it dependency-aware.

The next problem examines the process of making Garnet Standalone trace-driven and dependency-aware. Garnet Standalone was originally implemented to inject packets using a random number generator based on a user-specified packet generation function. In this paper, we generate traffic based on trace events. Trace events and NoC packets have dependencies. Garnet Standalone need to be modified so that it not only issue the events according to the time indicated in the traces but also to the dependency relationship.

The fifth problem deals with the process of handling two-way traffic. The Garnet Standalone originally performs one-way delivery of packets, from the source to the destination. However, Gem5 designates some end nodes as cores, cache banks, and memory controllers. The NoC traffic starts from the cores or L1 caches, goes to the LLCs or directories, to the memory controllers, and then return back to the LLCs and L1 caches. The messages in the NoC passed as a two-way traffic involving requests and responses. Garnet Standalone need to be modified to simulate two-way traffic.

3.2 Trace Generation

In this section, we discuss the process of trace generation for all possible input sources connected to Garnet Standalone form the Gem5. In this study, we will consider only the interactions between L1 caches and memory controllers. The memory requests, from L1 caches to memory controllers and their replies to the L1 caches, are examined in this work. The Garnet Standalone receives inputs from two types of end nodes, L1 caches and memory controllers. Synchronizations among the cores have not studied in this work.

The message processing technique of Gem5 contains three steps. Processor cores issue a RubyRequest message in Gem5 to executes a load or store instruction. The event ends with a L1 cache hit, but if the L1 cache has a miss, then the L1 cache sends a RequestMsg to the Garnet NoC simulator. Garnet after receiving the RequestMsg, delivers it to the designated memory controller (MC), which replies back with a ResponseMsg after retrieving the data block from the L2 cache/main memory.

We propose a simple memory model in this study to capture memory latency data accurately. We will record the memory latency data for each memory request in Gem5. Before replying to a request, the MC will wait for the memory latency time associated with that memory request. The L1 cache miss penalty of a memory request, which is required to obtain the memory latency, is given by the following equation.

$$L1\ cache\ miss\ penalty = T_{complete} - T_{issue} \tag{1}$$

Where, T_{issue} is the cycle time when the memory request is injected into the NoC and $T_{complete}$ is the time when the response message is returned to the L1 cache. The log file of Gem5 provides the T_{issue} and the $T_{complete}$ parameters.

The NoC time can be calculated by the following equation.

$$NoC\ time = T_{request} + T_{response} \tag{2}$$

Where, $T_{request}$ is the time when NoC receives the request message and $T_{response}$ is the time when NoC replied back. $T_{request}$ and $T_{response}$ parameters are obtained from Garnet simulation.

The memory latency can be calculated from the equation 1 and the equation 3 using the following equation.

$$Memory\ latency = L1\ cache\ miss\ penalty - NoC\ time \tag{3}$$

The 64-bit address field of a message in Garnet constitutes of 54-bit cache line address field, a 4-bit destination field, and a 6-bit block offset field. We have modified this address

Table 1: Configuration of Gem5 serving as the groundtruth

	Parameters
Frequency	2GHz
ISA	ALPHA
L1 cache	Private 64KB LRU Latency = 3
Reorder buffer size	192 entry
Memory model	Simple Memory Size = 512MB
NoC topology	Mesh
NoC Routing function	Dimension order
GarnetNetwork	4 virtual channels 4 buffers per data virtual channel 1 buffer per control virtual channel Flit size = 16 byte

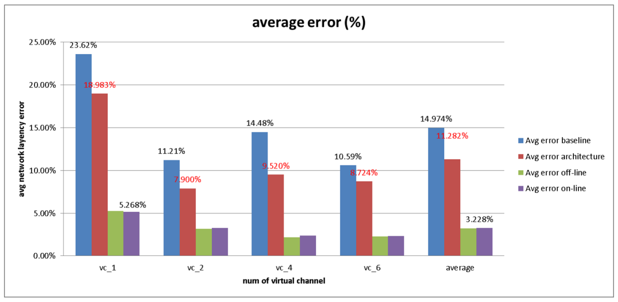

Figure 5: Average network latency error in each virtual channels

field to track messages in such a way that the cache line address field now consists of a 49-bit message identifier, 1-bit message type identifier, 4-bit source address, and 4-bit destination address, and the block offset field will have 6-bit. A model, such as independent cache miss model [5], is adopted in this study to infer the dependency relationship with an approximate mechanism. It views all L1 misses inside the reorder buffer (ROB) as an independent and outstanding event. On the other hand, L1 miss events occurred outside the ROB depend on those exist inside.

3.3 Trace Simulation

In the previous sub-section, we have discussed the methods to prepare a trace file from the Gem5, which can be used by the dependency-aware Garnet. In this sub-section, we discuss techniques to modify the Garnet Standalone simulator to be trace-driven and dependency-aware. Garnet Standalone simulator injects packages using a random number generator based on a user-specified packet generation function. We have modified the Garnet Standalone in such a way that it injects messages according to the event recorded in the trace files. The size of the messages is determined by the message type field in the trace events.

Dependency-awareness can be implemented in the Garnet Standalone by using two methods, on-line and off-line adjustment methods. The Garnet simulator can inject messages according to the time recorded in the trace file with their dependency relationship. This method applies to the case when the memory controllers inject their response messages

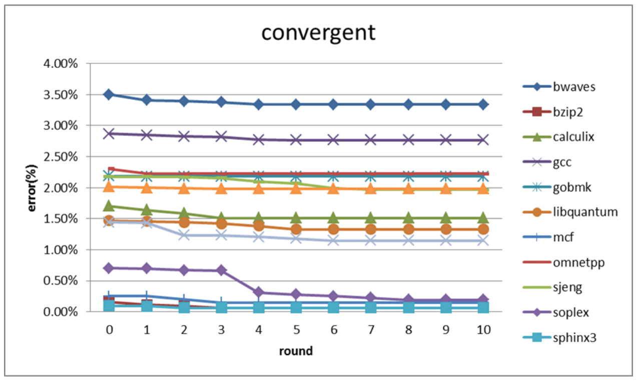

Figure 6: Convergence of off-line adjustment

based on the simple memory model discussed in this work. The on-line process used in the study is described in the Figure 1. Let us assume that, according to a trace file, an event A is intended to be issued at cycle 14 in the Gem5 and its response is returned at the cycle 23. However, in the dependency-aware Garnet, the response message is simulated to be returned at cycle 27. If an event D depends on the event A through the ROB then the issue time of the event D must be postponed to cycle 37.

Off-line adjustment methods can also be used to include dependency-awareness in the Garnet Standalone. The resultant cycle times noted by the Garnet simulation output are compared with the resultant cycle times recorded by the Gem5 outputs. The issue time of the corresponding input trace events is adjusted for those events where the resultant cycle times obtained from Garnet and Gem5 do not match. The off-line process implemented in the study is presented in the Figure 2. Let us assume that an event D depends on an event A according to the independent cache miss model [5] and after comparing the outputs generated by trace-driven Garnet and Gem5, it is observed that the event A responded 4 cycles later in Gem5. Therefore, the issue cycle of event D should be delayed for 4 cycles. The issue times of the events are adjusted in the input trace file and the modified trace file is fed into the Garnet for the second round of simulation. The outputs obtained from the second round of simulation are compared with the outputs generated by the first round, and their timings are modified for the third round of simulation in Garnet. This process repeats until the output timings converge [7].

The Garnet Standalone delivers a message in one-way, from the cores to the directories. The destination nodes can receive messages transmitted by the source node, but can not reply back. To solve this problem, we have attached one core and one memory controller to each node present in the NoC. The core attached with a node will issue memory requests to another node according to the trace events and receive responses from the memory controller present in that node. A node, in our modified Garnet, serves as both message source and destination.

The proposed modified Garnet creates two buffers in each node, a RequestMsg buffer and a ResponseMsg buffer. A node reads the trace file and pushes trace events belonging to the node in the RequestMsg buffer. The corresponding events then issue request messages in the NoC according to the time noted in the trace file and based on the dependencies

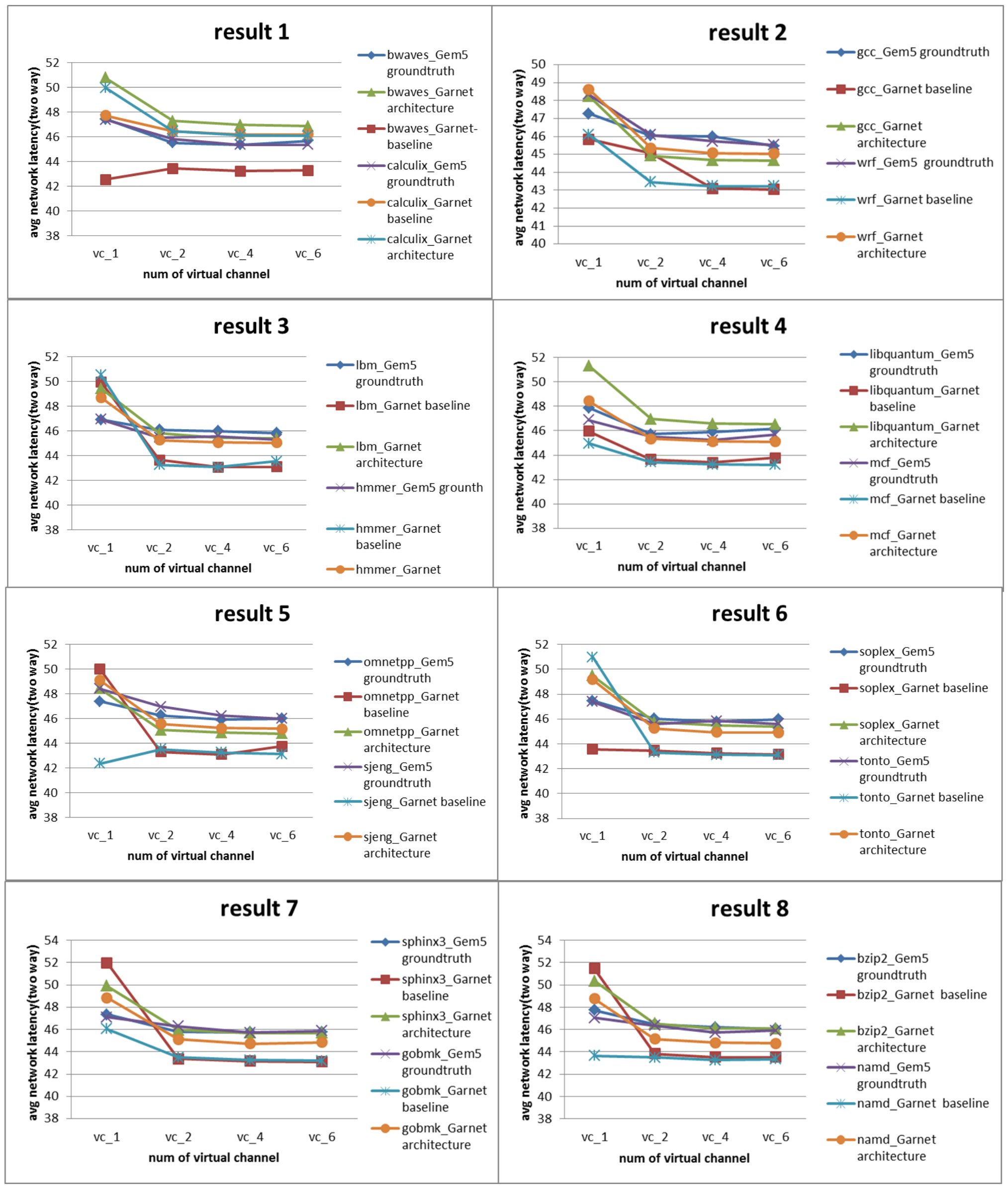

Figure 7: Designs with varying number of virtual channels without using programmatic dependencies

among the events. A node pushes a response event into the ResponseMsg buffer, according to the simple memory model implemented in the modified Garnet, after receiving a request message for memory access from another node.

4. PRELIMINARY EVALUATION

4.1 Evaluation Setup

The parameters, listed in Table 1, are used to configure the Gem5 to simulate a multicore system with 16 nodes, where each node contains one 2-GHz out-of-order core and a memory controller. Each core incorporates a reorder buffer with 192 entries and a 64-KB L1 cache. The main memory is 512 MB. The Gem5 does not simulate cache coherence and synchronization primitives because multilevel caches, cache coherence, and synchronization among the cores have not considered in this study. The nodes are arranged in a mesh topology in the NoC and applied dimension-order routing. Each port of a router has four virtual channels, one buffer for control virtual channel, and four buffers for a data virtual channel. SPEC CPU2006 is used as benchmark suite in our evaluation. For each benchmark, sixteen cores run the same program at the same time. We have not considered cache coherence and synchronization of traffic among the cores for evaluation. Average network latency is used in this study as the primary metric to compare the performance. It is the time elapsed from the time L1 cache issued a memory request and received the request back.

4.2 Results

To validate the fidelity of the proposed dependency-aware Garnet, we set the same system configuration in both Garnet and Gem5. The traces extracted from Gem5 are used in the Garnet to compare its performance in terms of average two-way network latency. Figure 3 illustrates the comparison of the proposed dependency-aware Garnet with the two-way average network latency of Gem5 using architectural dependencies. From Figure 3 it can be observed that the traditional trace-driven baseline Garnet produces an average error of 2.56 cycle, while the proposed architecture dependency-aware Garnet generates 0.72 cycle average error. The simple memory model may be the reason for the error that is produced by the proposed model.

Figure 4 demonstrates the error rate generated by the architecture dependency-aware Garnet and programmatic dependency-aware Garnet in terms of average network latency. It can be observed from Figure 4, that the baseline has an average error rate of 5.59%. Almost all the benchmarks leveraging architecture dependency have an error rate below 2% with a maximum error rate of 3.56% and an average error rate of 1.57%. Figure 4 also depicts, the average network latency with dynamic on-line adjustment and iterative off-line adjustment of the timings, in the input trace events, based on programmatic dependency. From Figure 4 it is evident that the programmatic dependency-aware Garnet produced more accurate results than the architecture dependency-aware Garnet and reduced the error rate from 1.57% to 1.3%.

Figure 5 illustrates the average network latency error in each virtual channels. The proposed method produce the error as low as 3.22% compared to the trace-driven Garnet Standalone. The baseline, on the other hand, has an average error rate of 14.97%.

Figure 6 represents the convergence rate of the off-line adjustment method and it is clear that the dependency-aware Garnet can get convergent performance results in 5 rounds for most benchmarks.

Traces from the full-system Gem5 are used to evaluate the effectiveness of using the architecture and programmatic dependency-aware Garnet running with a different configuration. The trace obtained from the Gem5 consists of a NoC that employs four virtual channels per port. The trace is then used to exploit the design space of NoC by architecture and programmatic dependency-aware Garnet by varying the number of virtual channels from one to six. Figure 7 illustrates the two-way average network latency obtained from the dependency-aware Garnet by varying the number of virtual channels with 1, 2, 4, and 6 using the trace generated by a NoC consists of four virtual channels. From Figure 7 it is evident that the proposed architectural dependency-aware Garnet can produce average network latency results very close to the Gem5 with a difference of one cycle in most cases and maximum difference four cycles. However, in the case of one virtual channel, the performance difference is 4 cycles. This is caused by the average network latency, that remains same for the light NoCs. When the NoC becomes congested, the trace-driven NoC simulator tends to inject packets faster than the full-system simulator, because the former emits packets according to the timing of a lighter NoC. As a result, the trace-driven simulator reports a more congested network than the full-system simulator.

5. CONCLUSIONS

Garnet, the NoC simulator of the popular cycle-accurate full-system multicore simulator Gem5 is modified, in this paper, for dependency-aware trace-driven simulation. We have validated our experiments of the dependency-aware Garnet using Gem5 with the same NoC configuration. The experiments using SPEC CPU2006 benchmarks show that dependency-aware Garnet can keep the average network latency within 1.57%. The architecture and programmatic dependency-aware Garnet can reduce the average error to 1.3%. The study presented in this paper demonstrates the usefulness and effectiveness of dependency-aware trace-driven simulation for NoC. We plan to consider micro-architectural dependencies in the future because the dependency-aware Garnet will be more useful if it could also detect and process micro-architectural dependencies. We will also address the dependencies due to synchronization operations among the cores, which is essential for the practical parallel applications.

6. ACKNOWLEDGMENTS

This work was financially supported by the Ministry of Science and Technology of Taiwan under Grants MOST 103-2622-E-002-034 and 104-2220-E-007-017, and sponsored by MediaTek Inc., Hsinchu, Taiwan.

7. REFERENCES

[1] N. Agarwal, T. Krishna, L. S. Peh, and N. K. Jha. GARNET: A detailed on-chip network model inside a full-system simulator. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 33–42, April 2009.

[2] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, 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. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, August 2011.

[3] S. Cho, S. Demetriades, S. Evans, L. Jin, H. Lee, K. Lee, and M. Moeng. TPTS: A novel framework for very fast manycore processor architecture simulation. In *Proceedings of the 37th International Conference on Parallel Processing*, pages 446–453, September 2008.

[4] J. Hestness, B. Grot, and S. W. Keckler. Netrace: Dependency-driven trace-based network-on-chip simulation. In *Proceedings of the Third International Workshop on Network on Chip Architectures*, NoCArc '10, pages 31–36, New York, NY, USA, 2010. ACM.

[5] K. Lee, S. Evans, and S. Cho. Accurately approximating superscalar processor performance from traces. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2009)*, pages 238–248, April 2009.

[6] C. Nitta, K. Macdonald, M. Farrens, and V. Akella. Inferring packet dependencies to improve trace based simulation of on-chip networks. In *Proceedings of the Fifth IEEE/ACM International Symposium on Networks on Chip (NoCS)*, pages 153–160, May 2011.

[7] T.-Y. Wang. Making ordinary trace-driven simulators accurate for design space exploitation. MS Thesis, Department of Computer Science, National Tsing Hua University, Taiwan, 2014.