Frequent Value Compression in Packet-based NoC Architectures

Ping Zhou ⋆, Bo Zhao ⋆, Yu Du †, Yi Xu ⋆, Youtao Zhang †, Jun Yang ⋆, Li Zhao ‡

⋆ ECE Department
University of Pittsburgh
Pittsburgh, PA 15261
email:{piz7,boz6,yix13,juy9}@pitt.edu

† CS Department
University of Pittsburgh
Pittsburgh, PA 15260
email: {fisherdu,zhangyt}@cs.pitt.edu

‡ System Technology Lab
Intel Corporation
Hillsboro, OR 97124
email:li.zhao@intel.com

Abstract— The proliferation of Chip Multiprocessors (CMPs) has led to the integration of large on-chip caches. For scalability reasons, a large on-chip cache is often divided into smaller banks that are interconnected through packet-based Network-on-Chip (NoC). With increasing number of cores and cache banks integrated on a single die, the on-chip network introduces significant communication latency and power consumption.

In this paper, we propose a novel scheme that exploits *Frequent Value* compression to optimize the power and performance of NoC. Our experimental results show that the proposed scheme reduces the router power by up to 16.7%, with CPI reduction as much as 23.5% in our setting. Comparing to the recent zero pattern compression scheme, the *frequent value* scheme saves up to 11.0% more router power and has up to 14.5% more CPI reduction. Hardware design of the FV table and its overhead are also presented.

I. Introduction

Chip Multiprocessors (CMPs) have recently gained popularity in both embedded and high-performance systems. By integrating multiple cores on a single die, CMP can provide better Thread-Level Parallelism (TLP) than single-core solution. To fully exploit the computing power of multiple cores, a large shared on-chip cache is preferred in addition to small private caches. The large cache is often divided into multiple banks that are interconnected through an on-chip network. This type of caches is referred as Non-Uniform Cache Architecture (NUCA) as the cache access time is not the same across different banks.

Recent studies showed that it is increasingly important to optimize on-chip interconnection network under chip area and power constraints [7]. Due to its advantages in predictability, reusability and scalability [2] [3], packet-based Network-on-Chip (NoC) is the most widely used interconnection fabric for CMPs. Unfortunately as the size of on-chip network scales, packet-based NoC introduces significant power consumption and communication latency overhead.

While many approaches have been proposed to optimize packet-based NoC designs, compressing on-chip data communication has been proven to be beneficial because it reduces the on-chip traffic and thus, optimizes both performance and power of the network.

Recently R. Das *et al.* discovered that a few value sequences i.e. *patterns* occur with very high frequencies in on-chip traffic [11]. They then proposed a scheme to exploit *frequent patterns* for packet-based NoC architectures [11]. Data values are processed through pattern matching hardware, and are encoded in forms of pattern prefix bits to accompany data packets. Among all patterns, the zero value pattern is most effective in achieving desired benefits, and requires the simplest hardware implementation. They also exploited the possibility of combining storage compression (cache compression) and communication compression (network compression) to store data in cache and increase the effective cache capacity at some addition hardware cost.

We propose to compress the NoC traffic through exploiting *frequent values* instead of value patterns. Frequent values refer to a small set of distinct values that present both spatially and temporally in the memory space. When utilized for data encoding on off-chip buses between the chip and memory, frequent values can reduce significant bus energy through lowering their switching activities. [4]. Previous effort has also used frequent value compression to reduce the energy consumption for the on-chip shared bus in CMPs [12]. This scheme incorporates a communicating value cache (CVC) to store frequent values, and leverages the snooping bus to mirror the CVC across all cores for consistent compression and decompression. However, this is not applicable to scalable NoCs with packet-based communication (instead of broadcasting) because CVC cannot be easily maintained consistently across all cores.

In this paper, we develop a frequent value (FV) based compression scheme for on-chip packet-based NoCs. We use a very small code book for end-to-end communications, and code books among different cores need not be the same. This can help to capture local FVs in addition to global FVs obtained through a uniform code book such as CVC. Overall, the contributions of this paper are:

- To the best of our knowledge, this is the first work that exploits the usage of *Frequent Value* compression in packet-based NoC architectures. We develop and compare the effectiveness of four alternative FV replacement policies.

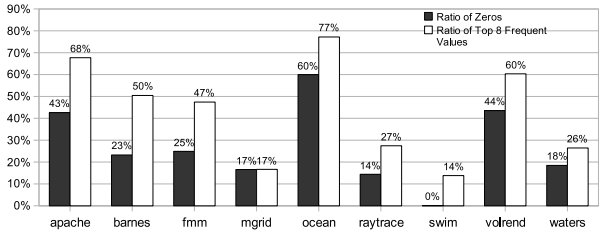

Fig. 1. Appearance ratio of top 8 FVs.

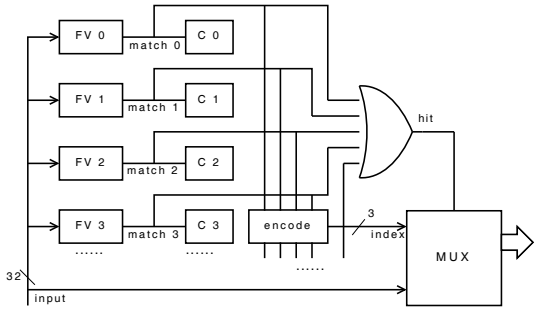

Fig. 2. FV control logic.

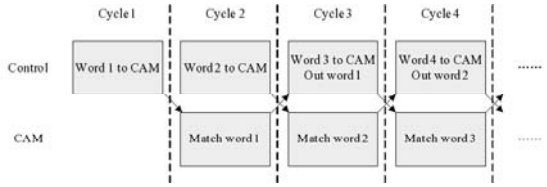

Fig. 3. Pipelined operation of CAM and controller.

- Our FV compression scheme reduces the average length of data messages by 24.0% on average — a 15.2% more reduction over the zero pattern scheme [11]. This results in a CPI reduction of up to 23.5%, a 14.5% improvement over the zero pattern scheme on average. FV compression also delivers 10.9% router power savings on average — a 7% more reduction over zero pattern scheme.
- Our FV compression scheme is general–purpose, and transparent to both the cache controller and network interface. This means it can be easily adopted in generic CMP/NoC platforms without major architectural changes.

In the following section, we discuss the motivation and design of our FV compression scheme. Section III describes our experimental platform, and Section IV presents the results of our simulation. Section V concludes the paper.

II. Frequent Value Compression on NoCs

A. Dynamic Frequent Values

Studies have shown that a small number of frequently repeated values (FV) account for a large percentage of on-chip data traffic [4], [12]. For example, Fig.1 presents the ratio of top 8 frequent values when running different applications on a 6x4 packet-based NoC architecture (with the setting described in Section III). From the figure, up to 77% appeared values are frequent values.

An important characteristic of frequent values is that they are highly dynamic. Frequent values change with different workloads, different inputs, and at different runtime intervals. In practice, this helps to achieve better compression rate and better accommodation to different workloads than fixed patterns (e.g. zero pattern [11]). As an example, the data traffic in `swim` is almost uncompressible using zero pattern while it exhibits 10% compression potential using FV scheme (Fig.1).

B. Frequent Value Table

To ensure the correct compression and decompression for an end-to-end communication channel, an FV table is maintained in a synchronized way on both sides. When sending a data message, the sender matches the values in the data message with its FV table. Each value with a hit will be substituted with its index into FV table. Otherwise the original value is used. To distinguish between compressed (hit) and uncompressed (missed) values in the encoded message, an extra flag bit is attached to each encoded value. This bit is used to indicate whether the corresponding value is compressed. Therefore, when using a FV table with 8 entries, the encoded length of a 32-bit value is either 4 bits (hit) or 33 bits (miss). On reception, the receiver decodes the message using its FV table. Since the FV tables on both sides contain the same values with the same indices, the message can be decoded properly.

We use Content-Addressable Memory (CAM) structure to realize the FV table. We implemented the FV table circuit and compressor/decompressor logic using Verilog HDL and synthesized the hardware with 45nm standard cell library [15, 16]. Fig.2 illustrates the logic of FV table controller. The CAM and control circuits operate in a pipelined way, so that the total cycles needed to process N values are N+2. Fig.3 shows the pipelined operation.

For an 8-entry FV table whose compressor and decompressor circuits work under 1GHz clock frequency, our results showed that the area overhead and dynamic power consumption of the compressor circuit are 4438.8μm^2 and 4.26mW, respectively; the area overhead and dynamic power consumption of decompressor circuit are 4320.2μm^2 and 3.95mW, respectively.

Data messages are packaged into network flits before being injected into the network. Since values are sequentially processed, we integrate the FV compressor into the packaging stage in each node's network interface device, and overlap the latency of compression and packaging operations. This helps to reduce the visible latency of FV compression to two cycles. Similarly, decompression of a message can start as soon as the first data flit arrives. Decompressed values are pushed out while following flits are being received and processed at same time. Fig.4 shows an example when decompressing a four-flit message.

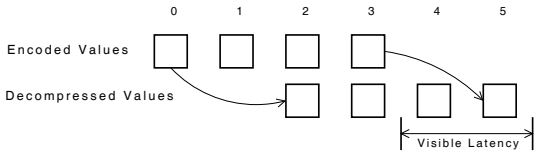

Fig. 4. Overlapping decompression with unpackaging.

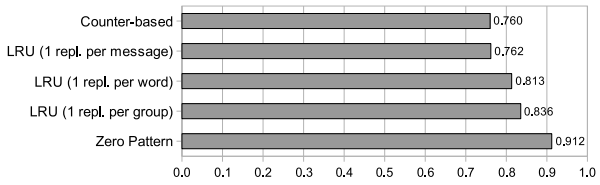

Fig. 5. Normalized message length of different FV replacement policies.

C. FV Replacement Policies

To capture dynamically appeared frequent values, the FV table needs to be updated periodically at runtime. The update should be synchronized to maintain the same table content on both the sender and the receiver sides. When replacing new values into the FV table, two factors must be considered. On one hand, replacing new values more aggressively leads to faster adaptivity to workloads' dynamic behavior. On the other hand, however, it is desirable to give old values enough time to take effect before they are evicted. We found that different FV replacement policies exhibit different trade-offs.

We implemented and evaluated four different FV replacement policies as follows:

- Counter-based replacement

  With this replacement policy, each entry in the FV table contains a 32-bit value and an 8-bit counter. To compress a data message (e.g. a cache line) that contains multiple 32-bit values, the controller tries to match each value in the FV table. To facilitate the discussion, we introduce following terms. A *hit-value* is a value in the data message that is also in the FV table; a *miss-value* is a value in the message but not in the table; a *hit-entry* is an entry in the FV table whose value appears in the current message; a *miss-entry* is an entry in the table whose value does not appear in the current message.

  To update the counter, the controller processes the values in the data message one by one. A hit increases the corresponding counter by two while multiple appearances of the same value update the same hit-entry multiple times. At the end of processing, the counters of all *miss-entries* are decreased by one. The counter value ranges from 0 to 255; it does not overflow or underflow i.e. increasing a counter with value 255 gets 255 while decreasing a counter with value 0 still gets 0.

  After processing all values, the controller tries to update the table based on counter values. If it can find a *miss-value*, and an entry whose counter is zero (it must be a *miss-entry*), then the missed value replaces the current value of the *miss-entry*. The above processing repeats until no distinct *miss-value* or no zero-counter entry can be found. The replacement process is not on the critical path.

- Approximate LRU/one replacement per message

  Using this policy, each entry in the FV table is comprised of a 32-bit value and an 8-bit timestamp. The timestamp is updated as follows. When processing values in the message, each hit left-shifts a bit-1 to the corresponding timestamp. At the end of the processing, each miss-entry left-shifts a bit-0 to its timestamp. This update policy is similar to the counter-based policy except that the timestamps of miss-entries decrease much faster.

  After processing all values in the data message, the FV controller scans the FV table to find one entry with zero timestamp and replaces it with a *miss-value*, if both the entry and the miss-entry can be found.

- Approximate LRU/one replacement per value

  Similar to the above policy, each entry in the table has a 32-bit value and an 8-bit timestamp. However, timestamp update and table replacement decisions are performed per each value instead of per message: as each value in the data message is being matched to the FV table, timestamps of FV table entries are updated based on matching results. If current value is a miss, the FV controller finds an entry with zero timestamp and replaces it with current value. This is an aggressive policy that evicts old values very quickly.

- Approximate LRU/one replacement per group

  This policy is a compromise between the above two policies. Values in a data message are divided into several groups e.g. each group contains 4 values. The update and replacement are performed per group instead of per message or per value.

We evaluated the effectiveness of the four FV replacement policies with multiple workloads (the settings are described in section III). Fig.5 compares the normalized data message lengths of these four policies. As we can see, the counter-based policy has the best average compression rate, therefore, we adopt it in our following study. The results are also used in distributed random traffic to study the effects of different compression schemes.

D. Frequent Value Compression for Packet-based NoCs

As we discussed, to ensure the correct compression and decompression for an end-to-end communication channel, the updates to FV tables on both ends must be synchronized. It is easy to achieve for bus interconnect as all nodes attached to a snooping bus can see all sent and received messages in the same order and update their local FV tables accordingly.

However the synchronization becomes more complicated in a mesh NoC. Given an end-to-end communication node pair, one node can send out messages without knowing the status of the other one. Therefore the nodes on two ends may see different orders of their exchanged messages. In Fig.6 the order of processed messages on A is $A1 \rightarrow A2 \rightarrow B1$ while on node B the order is $A1 \rightarrow B1 \rightarrow A2$. To remove this problem, we

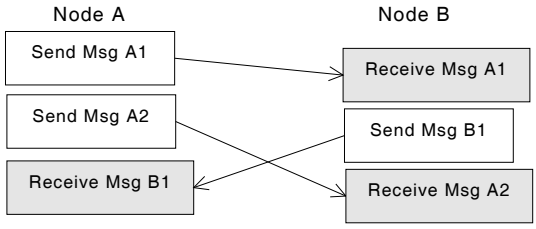

Fig. 6. Interleaved messages in bidirectional communication.

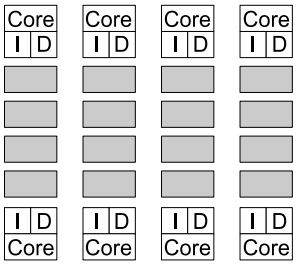

Fig. 7. On-chip network layout.

maintain two FV tables for each node pair — one for A-send/B-receive and the other one for A-receive/B-send i.e. both node A and B get $A1 \rightarrow A2$ and $B1$ for the two tables respectively.

Since a node on a CMP processor may simultaneously communicate with multiple other nodes, it needs to maintain two distinct FV tables for each node it talks to. Although it might be possible that several nodes share a same FV table, the complexity of synchronization and the need for a centralized controller offset the benefits it might bring. Although each node has multiple FV tables, only those that correspond to current communication channels are active.

III. Experiment

A. Simulation Model

We simulated an 8-core CMP using Simics [13] full system simulator and GEMS [8] toolset. Each core has a 32KB private L1 cache; all cores share an 8MB L2 cache that is divided into 16 512KB banks each with a local directory. The 16 L2 banks together with 8 cores are interconnected using a 6x4 mesh NoC (Fig.7). We used a state-of-art fixed pipeline router with deterministic X-Y routing algorithm in the NoC. The details are summarized in Table I.

Since zero pattern takes majority of frequent patterns in data traffic [11], and is simple to implement in hardware, we set zero pattern scheme as the baseline and compared our FV compression scheme with it.

B. Workloads

We collected a diverse set of parallel computation workloads from SPLASH2 [9] and OpenMP2001, as well as server workload. The details of workloads are specified in Table II. In all simulation runs, we skipped the initialization phase to avoid biased data values.

TABLE I
Simulation Model

Component	Parameters
Processor Cores	Eight cores, each at 1GHz clock.
L1 Cache	Each core has a 32KB private L1 cache, split I and D cache, 4-way set associate, 64 byte per line, 3 cycles access time.
L2 Cache	Shared L2 Cache divided into 16 512KB banks, 8-way set associative, 64 bytes per line, 6 cycles access time.
Memory	4GB RAM, 200 cycles access time.
Network	6x4 mesh network, with 8 cores on two sides and 16 L2 banks in middle (Fig.7).
Router	Fixed five-stage router with X-Y routing, 2 VCs per PC, 64-bit flits. Routers run at same frequency as processor (1GHz). An unencoded data message is comprised of 1 header flit and 8 data flits which stores a cache line.
FV Table	Each table has 8 entries, and use counter-based replacement policy.

TABLE II
Workloads

Benchmarks	Description
Barnes	Standard 8 processor input set.
Ocean	514x514 ocean with 8 processors.
FMM	8 processor input set with 16384 particles.
Raytrace	8 processor, 256MB shared memory and 2-pixel anti-aliasing.
Volrend	8 processors, standard input set ("head").
Water-Spatial	Standard input set for 8 processors.
Swim	8 processors, reference input set.
Mgrid	8 processors, reference input set.
Apache	We use Apache 2.2.6 for Solaris 9 with default configuration, and use SURGE [10] to generate requests (16 clients, 100 threads per client).

C. Distributed Random Traffic

In order to fully evaluate our FV compression scheme under various NoC traffic conditions, we developed a tunable traffic generator which is capable of generating uniform distributed random traffic at specified rate. Such random traffic exists at the background of the benchmarks under test and does not affect their execution. Their existence emulates the situation when there are certain levels of network traffic contention due to multi-threading, multiple workloads etc. that are overly time-consuming in full system simulation. When configured with a *random traffic rate* between 0 and 1, a node can generate distributed random traffic so that:

$$G \approx T * R$$

where T is the time (cycles) elapsed since simulation is started, R is the random traffic rate and G is the number of random traffic flits generated. These random traffic flits are sent with a

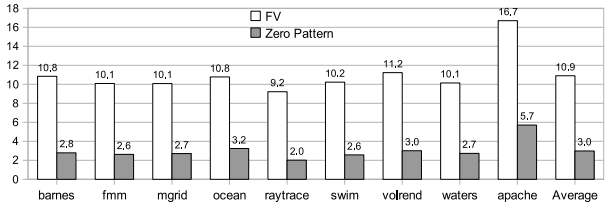

Fig. 8. Router energy saving % under random traffic rate.

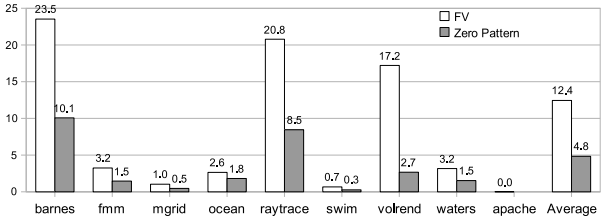

Fig. 9. CPI reduction % under random traffic rate.

special flag so that they can be specially handled at receiver side without disturbing the timing model of the Simics simulator.

The data messages injected by the random traffic generator conforms with the value characteristics that we observed in different benchmarks i.e. the compression rate is kept at about the same as (Fig.5) when simulating different compression schemes.

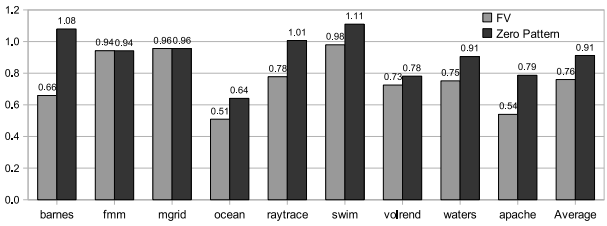

Fig. 10. Normalized data message length.

IV. Results

A. Router Energy Saving

To compare the router energy saving of FV compression and zero pattern, we ran both schemes with same workloads under random traffic rate 0.39 (0.39 flit/cycle per node on average). We compared their router energy savings against non-compression scheme and summarized the results in Fig.8. From the figure, FV compression yields as much as 16.7% router energy reduction, and up to 11.0% more reduction than zero pattern.

To measure the energy overhead of FV compression, we calculated the average energy consumption for transmitting one data message using the Orion [14] power model (at 45nm), and compared it with the energy consumed by FV compression and decompression. Our results show that the compression and decompression energy per message is $0.148nJ$ while the average energy for a message to traverse the network is $3.56nJ$. Therefore the energy overhead from FV compression operations is reasonable comparing to its router energy savings.

B. Performance Improvement

We then measured the performance changes using FV compression and zero pattern respectively. We ran both schemes with same workloads under random traffic rate 0.39, and compared their reduction of Cycles-Per-Instruction (CPI) number against non-compression scheme in Fig.9. Our simulation results show that FV compression yields up to 23.5% CPI reduction, and up to 14.5% more reduction than zero pattern compression.

C. Compression Rate

Next we studied the average length of data messages using FV compression and zero pattern compression (Fig.10). From the figure, FV compression yields up to 42% more reduction, and 15% more reduction on average. In particular, FV compression adapts to a larger variety of workloads, e.g. for Barnes, zero pattern compression increases the length by 8% while our FV compression achieves 34% length reduction.

D. Latency Reduction

We also ran simulation to collect the average latency per flit (including network latency and queuing latency) under different random traffic rates. We compared the latency trend of different schemes in Fig. 11. Our results show that FV compression reduces average latency and significantly defers the saturation point.

V. Conclusion

In this paper, we examine how FV compression can be applied in NoC architectures. We evaluated four FV replacement policies from which we picked up the counter-based policy as it provides the best average compression rate. We also explored the hardware implementation of FV table as well as its area and energy overhead.

Our simulation results indicate that the FV compression scheme reduces message length by as much as 49%, 15% more comparing to zero pattern on average. As a result, FV compression delivers up to 11.0% more router energy savings and up to 14.5% more CPI reduction over zero pattern (under random traffic rate 0.39).

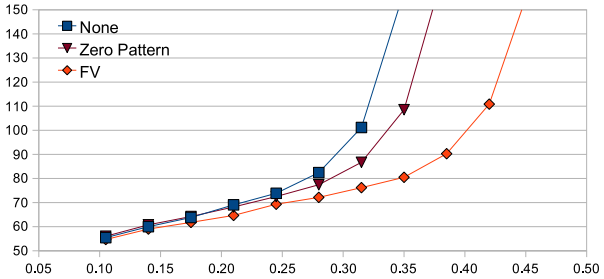

Fig. 11. Average latency under different random traffic rates.

In addition, our FV compression scheme is transparent to both cache controller and network interface. Thus it can be easily adopted into existing CMP/NoC architecture without major architectural change.

REFERENCES

[1] B. M. Beckmann and D. A. Wood. "Managing wire delay in large chip-multiprocessor caches," *MICRO 37: Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, pp. 319–330, 2004.

[2] A. Jantsch and H. Tenhunen. *Networks on Chip*, Kluwer Academic Publishers, pp. 9–14, 2003.

[3] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," *DAC '01: Proceedings of the 38th Conference on Design Automation*, pp. 684-689, 2001.

[4] J. Yang, R. Gupta and C. Zhang. "Frequent value encoding for low power data buses," *ACM Trans. Des. Autom. Electron. Syst.*, pp 354–384, 2004.

[5] J. Yang and R. Gupta. "FV encoding for low-power data I/O." *In Proc. ACM/IEEE International Symposium on Low Power Electronics and Design, Huntington Beach, CA*, August 2001.

[6] J. Yang, Y. Zhang and R. Gupta. "Frequent value compression in data caches." *In Proc. 33rd IEEE/ACM International Symposium on Microarchitecture, Monterey, CA*, December 2000.

[7] H. Wang, L. -S. Peh, and S. Malik. "Power-driven design of router microarchitectures in on-chip networks. *In MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, pp 105, 2003.

[8] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill and D. A. Wood. "Multifacets General Execution-driven Multiprocessor Simulator (GEMS) Toolset", *In Computer Architecture News (CAN)*, September 2005.

[9] J. P. Singh, W. Weber, and A. Gupta. "SPLASH: Stanford Parallel Applications for Shared-Memory". *In Computer Architecture News*, vol. 20, no. 1, pp 5–44.

[10] P. Barford and M. Crovella. "Generating representative web workloads for network and server performance evaluation." *In SIGMETRICS '98/PERFORMANCE '98: Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pp 151–160, 1998.

[11] R. Das, A. K. Mishra, C. Nicopoulos, D. Park, V. Narayanan, R. Iyer, M. S. Yousif and C. R. Das. "Performance and Power Optimization through Data Compression in Network-on-Chip Architectures." *In HPCA: Proceedings of the 14th International Symposium on High-Performance Computer Architecture*, pp 215–225, February 2008.

[12] C. Liu, A. Sivasubramaniam and M. Kandemir. "Optimizing Bus Energy Consumption of On-Chip Multiprocessors Using Frequent Values." *In EUROMICRO-PDP'04: Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, 2004.

[13] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. "Simics: A full system simulation platform." *Computer*, 35(2):50–58, 2002.

[14] H. Wang, X. Zhu, L. -S. Peh and S. Malik, "Orion: A Power-Performance Simulator for Interconnection Networks." *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture*, pp 294–305, November 2002.

[15] 45nm BSIM4 model card for bulk CMOS: V1.0, Feb 22, 2006, http://www.eas.asu.edu/∼ptm/latest.html

[16] PTM interconnect model, http://www.eas.asu.edu/∼ptm/interconnect.html