A Filtering Mechanism to Reduce Network Bandwidth Utilization of Transaction Execution

LIHANG ZHAO, Information Sciences Institute, University of Southern California

LIZHONG CHEN, School of Electrical Engineering and Computer Science, Oregon State University

WOOJIN CHOI and JEFFREY DRAPER, Information Sciences Institute,
University of Southern California

Hardware Transactional Memory (HTM) relies heavily on the on-chip network for intertransaction communication. However, the network bandwidth utilization of transactions has been largely neglected in HTM designs. In this work, we propose a cost model to analyze network bandwidth in transaction execution. The cost model identifies a set of key factors that can be optimized through system design to reduce the communication cost of HTM. Based on the model and network traffic characterization of a representative HTM design, we identify a huge source of superfluous traffic due to failed requests in transaction conflicts. As observed in a spectrum of workloads, 39% of the transactional requests fail due to conflicts, which renders 58% of the transactional network traffic futile. To combat this pathology, a novel in-network filtering mechanism is proposed. The on-chip router is augmented to predict conflicts among transactions and proactively filter out those requests that have a high probability to fail. Experimental results show the proposed mechanism reduces total network traffic by 24% on average for a set of high-contention TM applications, thereby reducing energy consumption by an average of 24%. Meanwhile, the contention in the coherence directory is reduced by 68%, on average. These improvements are achieved with only 5% area added to a conventional on-chip router design.

Categories and Subject Descriptors: C.1.4 [Parallel Architectures]: Hardware Transactional Memory; C.1.2 [Multiple Data Stream Architectures]: Interconnection architectures

General Terms: Design, Performance

Additional Key Words and Phrases: Transactional memory, on-chip network, network traffic, energy efficiency, communication cost modeling

ACM Reference Format:

Lihang Zhao, Lizhong Chen, Woojin Choi, and Jeffrey Draper. 2015. A filtering mechanism to reduce network bandwidth utilization of transaction execution. *ACM Trans. Archit. Code Optim.* 12, 4, Article 51 (December 2015), 26 pages.

DOI: <http://dx.doi.org/10.1145/2837028>

1. INTRODUCTION

The paradigm shift from uniprocessor to the chip multiprocessor architecture enables massive thread-level parallelism. In this many-core era, one grand challenge is to write multithreaded parallel applications to effectively exploit tens to hundreds of processor cores available on a single chip. The task of synchronizing concurrent accesses using traditional locking primitives is burdensome for programmers: coarse-grain locks limit

Authors' addresses: L. Zhao, W. Choi, and J. Draper, University of Southern California, Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292; emails: lihangzhao@gmail.com, woojinch@usc.edu, draper@isi.edu; L. Chen, Assistant Professor, School of Electrical Engineering and Computer Science, College of Engineering, Oregon State University, 3113 Kelley Engineering Center, Corvallis, OR 97331.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2015 ACM 1544-3566/2015/12-ART51 \$15.00

DOI: <http://dx.doi.org/10.1145/2837028>

performance, while fine-grain locks increase complexity. Transactional Memory (TM) promises to increase the productivity in parallel programming by providing language constructs to delimit code blocks (i.e., transactions) that appear to execute atomically and in isolation with other threads. TM systems can be implemented in the software stack, in hardware or a hybrid approach. The pure software approach is inherently slow, thereby preventing its large-scale usage. This work focuses on the hardware execution of transactions (applicable to Hardware Transaction Memory (HTM) and hybrid TM), as its tight coupling with evolving parallel processor architectures continues to enable extensive in-hardware optimization opportunities.

HTM research has generally focused on performance [Negi et al. 2012; Lupon et al. 2010; Chafi et al. 2007; Yen et al. 2007], implementation issues [Sanchez et al. 2007; Blundell et al. 2007], transaction scheduling [Blake et al. 2009, 2011; Scherer III and Scott 2005], and hardware-software interplay [Shriraman et al. 2008; Rajwar et al. 2005; Rossbach et al. 2007]. These efforts have paved the way for HTM to be present in commodity systems [Tremblay and Chaudhry 2008; Haring et al. 2012; Yoo et al. 2013]. However, the majority of the research proposals on HTM either assume an ideal on-chip network with zero latency or a simple communication fabric. While packet-switched on-chip networks are viewed as the *de facto* solution for future many-core processors to supply low latency, high-bandwidth and energy-efficient on-chip communication seldom has the interaction between HTM and on-chip networks been studied. It is of vital importance that HTM research expand beyond the processor core into on-chip networks for the following three reasons.

Energy efficiency. HTM imposes an energy footprint on the network because the delivery of coherence messages from transactions incurs energy dissipation in routers as well as on links. As conflict detection requires frequent intertransaction communication, HTM designs could have a huge impact on the network energy consumption. Energy-efficient on-chip communication for HTMs cannot be achieved without an in-depth understanding and optimization of the interaction between HTM and the network.

Performance. HTM designs depend on the on-chip network to fetch data and detect conflicts (TM traffic). Due to the overhead of serializing transactions in data race conditions, the memory level parallelism is reduced significantly, making the overall performance more sensitive to the network latency of memory requests from transactions. The criticality of TM traffic demands an on-chip network that is optimized to accelerate memory requests from transactions and streamline intertransaction communication.

Quality-of-Service (QoS). Transactional applications inherently demand extensive core-to-core communication to exchange data and resolve conflicts between transactions. The bandwidth requirement grows rapidly with the transaction granularity as well as the thread count. However, the aggressive bandwidth utilization in transactional applications could increase the contention and degrade the network's responsiveness. Certain mechanisms should be available to contain the transactional communication when needed, so that the shared network resource is not monopolized.

In this work, we develop a cost model of network bandwidth in transactional execution to explore the interplay between HTMs and on-chip networks. The model identifies a set of first-order factors in determining the network bandwidth utilization of HTM, which thereby provides insight into reducing the bandwidth cost of HTM. Based on the model, we focus on one particular source of excessive network traffic that needs to be regulated: the forwarding of exclusive coherence requests to all sharer nodes. The coherence requests from transactions could be negatively acknowledged (nacked) due to conflicts with other transactions. Before the request is nacked, it can initiate numerous messages between the requester, directory, and sharers across the chip. Those messages do not contribute to the continued execution of transactions because the

Fig. 1. Comparison of network traffic for HTM conflict detection with and without in-network filtering. TxGETX: transactional write request to the directory. Fwd_TxGETX: forwarded TxGETX request from the directory to the sharer.

request is nacked eventually. This problem is referred to as *false forwarding*. False forwarding generates a large number of superfluous coherence messages and degrades the energy efficiency of the on-chip network as every hop of each message consumes energy in the routers and on the links. Moreover, the excessive intertransaction communication degrades the QoS of the network. False forwarding becomes a more serious problem in certain TM systems that allow transactions to retry the nacked requests instead of aborting to save wasted work. Since only the last one in a series of repeated retries successfully obtains the desired data access permission, the rest are nacked, which intensifies false forwarding in the network. Energy waste and performance loss due to false forwarding will be further exacerbated as the number of cores scales up and coarse-grain transactions with high contention rates prevail [Shriraman and Dwarkadas 2009]. Unfortunately, false forwarding is hard to tackle in HTM designs alone due to the tight coupling of HTM and coherence protocols and the exorbitant overhead of devising a specialized protocol.

We introduce TM-aware NOC (TMNOC) to filter out transactional requests that can incur false forwarding. The scheme consists mainly of two design components. First, a communication mechanism is proposed for the HTM and on-chip network to exchange critical information about transaction conflicts. Second, the on-chip routers are augmented to store the conflict information and predict potential conflicts. Enabled by these two mechanisms, the network filters out transactional requests that have a high probability of being nacked due to conflicts with peer transactions. The effectiveness of such a mechanism relies heavily on the accuracy of conflict prediction in the router. In the proposed scheme, each on-chip router tracks transaction conflicts by monitoring the coherence NACK and ACK messages flowing through the router. The NACK messages indicate conflicts between the source and destination nodes, while the ACKs indicate the resolution of such conflicts. The routers leverage the conflict information to predict potential recurring conflicts that subsequent requests would encounter. Those requests that are highly likely to fail due to conflicts will be filtered away in the router proactively. Moreover, TMNOC allows either the processor or the operating system to configure the conflict prediction for improved prediction accuracy, as the processor or software has better knowledge of the conflict profile.

Figure 1 illustrates the in-network filtering on a 2D mesh on-chip network. A directory coherence protocol is assumed. In the baseline system without filtering, as shown in Figure 1(a), a transactional GETX request (request for exclusive access) from Node3 is sent to the directory on Node1, which forwards the request to two sharers on Node0 and Node8, respectively. Due to conflicts, the transactions on Node0 and Node8 respond

to the requester transaction with NACK messages. As the request fails eventually, it causes false forwarding in which a large amount of messages are wasted. However, false forwarding can be proactively prevented with the in-network filtering. When the network is enabled to track conflicts between transactions, the router at Node4 might already know from past tracking records that the conflicting transactions on Node0 and Node8 will nack the request from Node3. Therefore, the router at Node4 nacks the request immediately instead of forwarding it to the directory (see Figure 1(b)). Subsequent communication is avoided.

Our evaluations using full-system simulation show that the in-network filtering mechanism reduces 21% (up to 39%) of the network traffic in a set of high-contention applications representative of TM workloads. Consequently, the network energy consumption is reduced by 24% (up to 39%). Directory busy cycles are reduced by 68%. An implementation using a standard VLSI design flow shows that TMNOC incurs a marginal area overhead of 5% for a baseline four-stage virtual channel router.

The contributions and organization of this article are as follows. In Section 2, we develop a cost model to analyze the network bandwidth utilization of HTM. With the model, we are able to identify a set of key factors that determine the communication cost of a HTM design. Additionally, the network traffic of a set of TM applications is characterized to understand the benefit in applying the filtering mechanism to reduce bandwidth cost in HTM execution. Section 3 describes the in-network filtering mechanism and its implementation. A comprehensive discussion on the filtering algorithms is provided. In Section 4, we evaluate the proposed mechanism through extensive full-system simulations to demonstrate its ability to improve overall energy efficiency and performance. A set of sensitivity studies for key design parameters are also included. In Section 5, we summarize related work on coherence traffic regulation, application-NOC interplay, and transaction conflict prediction. Section 6 concludes this article.

2. A COST MODEL OF NETWORK BANDWIDTH IN HTM

2.1. Interaction between HTM and NOC

In chip multiprocessors, the HTM and NOC are tightly coupled. Transactions fetch data and communicate with each other via the on-chip networks. TM-induced network traffic often takes the form of coherence messages. As the messages are injected into the network, they are encapsulated into short or long packets, which are further divided into flow control digits or flits. In typical on-chip networks, short packets (e.g., coherence read requests and acknowledge responses) are single-flit, while long packets (e.g., coherence read responses and write requests) have multiple flits. Once injected into the network, the packet is forwarded hop by hop by routers to the destination node. After being reassembled at the destination node, the coherence messages are ejected from the network. Then, the transaction at the destination is notified of receiving a message from the remote transaction.

Conflict detection and resolution guarantees the correctness of transaction execution. Any coherence protocol capable of detecting accessibility conflicts can also detect transaction conflicts [Herlihy and Moss 1993]. Directory-based protocols provide scalable solutions to cache coherence due to a unicast nature of communication [Enright Jerger and Peh 2009]. The directory can be distributed among all the nodes by statically mapping a cache line address to its home node. The home node is responsible for ordering coherence requests to the same cache block. The majority of HTM designs assume directory protocols for conflict detection. Our work follows suit so that the proposed design can be readily migrated to such HTMs. Nonetheless, the proposal is also applicable to systems adopting snooping protocols on a totally ordered broadcast network. In general, the eager and lazy conflict detection schemes have their own benefits

Fig. 2. HTM conflict detection. R: Requester; DIR: home node directory; S: Sharer.

regarding on-chip communication overhead. This work mainly targets the wasted traffic in eager conflict detection. However, the basic principle is applicable to the lazy conflict detection where the committing transactions usually use eager conflict detection to protect their write sets.

When a transaction is executing, the load address (store address) is added into the transaction's *read set* (*write set*). Upon receiving a request from another transaction, the transaction checks the request against its read and write set to see if any conflict occurs. Conflicts are resolved by serializing the execution of conflicting transactions. The execution order of conflicting transactions is determined by conflict resolution policies. A conflicting transaction with lower priority should stall or abort while one with higher priority continues executing. Figure 2 depicts TM conflict detection using the MESI (Modified, Exclusive, Shared, Invalidate) directory protocol. The requester transaction issues a GETX to the directory ①, which replies to the requester with data ②. The directory state of the block is set to busy (i.e., incoming requests to the same block are blocked). Then, the request is forwarded to the nodes currently sharing the block ③. Depending on the outcome of conflict detection and resolution, the sharing transactions respond with either a NACK (negative acknowledgement) or an ACK ④. Upon receiving all the responses, the requester sends an UNBLOCK message to the directory to conclude the request ⑤. If all the responses are ACKs, the requester transaction continues executing. If one of the responses is a NACK, the requester transaction stalls and keeps retrying the nacked request until all the high-priority sharer transactions have finished executing. In what follows, the transaction that sends a NACK message is often called *nacker transaction* or *nacker*. The node on which a transaction is executed is referred to as the transaction's host node.

2.2. Categorization of Communication Cost in HTM

To analyze the interaction between HTM and NOC, we develop an analytical model of the network bandwidth cost in transaction execution. This model categorizes costs based on high-level operation and type of system event (e.g., cache coherence and transaction conflict) so that it is applicable to generic CMP architectures with HTM support. In subsequent sections, we use this model to gain insights into the viable approaches to reduce the communication cost of a HTM system.

On-chip communication in transactional systems is mainly attributed to data transfer and the corresponding control. Ideally, the on-chip communication of a transactional data request would be the round-trip cost of transferring data between the home node and the requesting node. Here, a home node is defined as the node that either caches the requested data or hosts the memory controller to fetch data from off-chip memory. This cost is termed *inherent cost*, which is the minimal cost for a processing element to fetch data into its private memory (e.g., L1 cache). The uniprocessor scheme achieves

this minimal cost trivially. However, for shared memory CMPs, the cache coherence protocol usually incurs extra on-chip communication that is necessary for correctness. In a directory-based coherence protocol, when the data is not shared by any other cores, the home node can respond to requestors without initiating further communication. In this case, the data request only incurs the inherent cost. Otherwise, if the data is shared (owned), the home node has to ask the sharers (owners) to invalidate their private copies and respond to the requestors. The communication between the home node and the sharers (owners) constitutes the *coherence cost*, which is essential for achieving cache coherence. In general, the inherent cost and coherence cost is incurred in almost all shared memory CMPs.

In a transactional system, multiple requests from a thread to the shared memory are packed into a transaction (chunks of requests) that is executed atomically and in isolation from other transactions. Transactional requests mainly take the form of coherence requests with extra information fields in the message. As with plain coherence requests, transactional requests also incur the inherent cost and the coherence cost. In addition, extra on-chip communication is required. The additional cost is termed *transactional cost*, which can be further categorized into conflict cost, squash cost, and utility cost.

The *conflict cost* is incurred by transactional requests that fail due to conflicts between the requestors and other concurrent transactions in the system. Each failed request incurs communication to the home node and to the sharer nodes. In HTM systems that allow transaction stalling, a request can fail multiple times before eventually succeeding in obtaining data access permission, thereby making the conflict cost a multiple of the inherent cost and coherence cost combined.

The *squash cost* is the aggregate communication cost of those transactions that are aborted. Data accesses in each transaction incur inherent cost, coherence cost, and conflict cost in the process of requesting data. When the transaction aborts due to conflicts, the associated cost contribute to the overall squash cost, which is essentially a waste of network bandwidth.

The *utility cost* is the communication to facilitate the mechanisms in HTM designs. This cost is highly dependent on the specific system. For instance, the Scalable TCC [Chafi et al. 2007] uses a specific message type for the committing transaction to request a commit ID and to probe the target directory. In the EazyHTM [Tomić et al. 2009], the abort messages from a committing transaction to peer transactions in its racer list is another example of the utility cost.

2.3. Cost Function of HTM On-Chip Communication

We use the network hop count to quantify the communication cost in this cost model. The hop count is a proper abstraction of the underlying communication fabric and proportional to the latency and energy cost. To decouple the generic analysis with the specific network topology, we use the average hop count (h) as the basic unit to measure the node-to-node communication cost (i.e., the hop count between any pair of nodes is h , on average).

Based on the definition of inherent cost as the round-trip cost between the requesting node and home node, the inherent cost of a request can be calculated using Equation (1).

$$C_{inherent} = 2 \times h \quad (1)$$

The coherence cost depends on how many nodes share the requested data block because the home node must notify the sharers through forwarding. It can be calculated with Equation (2), in which n_{fwd} is the number of nodes to receive the forwarded request

and $2h$ is the average round-trip cost between the home node and the sharer node.

$$C_{coherence} = n_{fwd} \times 2h \quad (2)$$

If the request is nacked by at least one of the sharers due to transaction conflicts, the request fails. The inherent and coherent cost incurred by a failed request is the conflict cost. The request could be retried and succeed eventually. A successful request's cost is the sum of the conflict cost associated with its failed attempts plus the inherent and coherence cost with the last attempt that succeeds in obtaining the data. The conflict cost is calculated using Equation (3), where n_{retry} is the number of retries before the request is serviced successfully without conflict. The coherence cost in each retry changes with the number of sharers.

$$C_{conflict} = \sum_{i=1}^{n_{retry}} (C_{inherent} + C_{coherence(i)}) \quad (3)$$

The transaction issuing a request could be aborted even though a request is successful without conflict. In this case, the request's cost is the squash cost, which wastes the network bandwidth because the data is squashed. If a transaction aborts multiple times before committing, there could be multiple squash costs associated with the request within the transaction. The aggregated squash cost is calculated using Equation (4). $n_{restart}$ is the number of transaction restarts before committing.

$$C_{squashed} = \sum_{j=1}^{n_{restart}} (C_{inherent} + C_{coherence(j)} + C_{conflict(j)}) \quad (4)$$

When a transaction commits, the request is materialized because any machine state change associated with the request becomes non-speculative. The cost of a materialized request is the sum of (1) squash costs in aborted transactions and (2) the inherent, coherence and conflict costs associated with the request in the committed transaction. Thus, the overall communication cost incurred by a transactional request that eventually commits to memory can be calculated as below:

$$\begin{aligned} C_{total} = & 2h + n_{fwd} \times 2h + \sum_{i=1}^{n_{retry}} (2h + n_{fwd(i)} \times 2h) \\ & + \sum_{j=1}^{n_{restart}} \left(2h + n_{fwd(j)} + \sum_{i=1}^{n_{retry(j)}} (2h + n_{fwd(i)} \times 2h) \right) \end{aligned} \quad (5)$$

Assuming n_{fwd} , n_{retry} , and $n_{restart}$ are independent variables. The cost function can be calculated with the mean of n_{fwd} , n_{retry} , and $n_{restart}$ as in Equation (6).

$$C_{total(reduced\ form)} = (1 + \overline{n_{fwd}}) \times (1 + \overline{n_{retry}}) \times (1 + \overline{n_{restart}}) \times 2h \quad (6)$$

2.4. Key Factors in Determining HTM Communication Cost

The above cost function identifies the first-order factors that contribute to the cost. Thus, the cost can be reduced through controlling the individual key factors.

n_{fwd} . The number of nodes to receive the coherence forwarding from the home node is determined by current nodes that share the requested data block. The sharing dimension is specified mainly in the parallel programs. There are established software techniques (e.g., Privatization [Spear et al. 2007]) to reduce the sharing dimension without compromising performance. On the hardware side, several techniques (e.g.,

Fig. 3. Breakdown of GETS/GETX coherence requests from transactions to the directory.

Zhao et al. [2014] and Zhao and Draper [2014]) have been proposed to mitigate forwarding when it is unnecessary for all sharers to receive the forwarding in order to resolve the conflict.

n_{retry} . The number of retries before obtaining the data access permission is largely determined by two factors. The first factor is the transaction characteristics. If transactions have high contention, n_{retry} increases. Also, coarse-grain transactions run longer, thereby forcing other transactions to retry more. The second factor is the conflict detection and resolution mechanisms of a HTM system. For instance, if conflicts are detected lazily and resolved with the committer-win policy, the committing transaction's requests always succeed without the need to retry. Another example is the fine-grain backoff after a request is rejected.

$n_{restart}$. The number of transaction restarts before committing is also determined by the transaction characteristics in the application and the HTM design, similar to n_{retry} . In general, transaction aborting should be avoided when possible because it is detrimental to performance and energy efficiency.

h . The average hop count is determined by the topology of the underlying communication fabric. Reducing the hop count not only shortens the communication latency but also saves energy as fewer routers and links need to be traversed.

2.5. Motivating Pathology: False Forwarding

False forwarding is a pathological behavior that occurs when a transaction's coherence request, before being naked eventually, initiates numerous messages from the requestor to the directory, from the directory to each sharer/owner, and from each sharer/owner to the requestor. False forwarding unnecessarily increases n_{fwd} , which in turn adds traffic overhead.

To estimate the extent of false forwarding, we first track the GETS/GETX coherence requests generated by transactions in a representative HTM system. Figure 3 presents the breakdown of requests based on the outcome of the requests. Across all eight workloads, naked requests account for 39% of all the requests from transactions. So, more than one third of all the TM-induced coherence requests incur false forwarding. Further, we categorize transactional communication into two types, namely, effective and abortive. The *effective transactional communication* is the acknowledged coherence requests and the associated responses in transactional execution, whereas the *abortive transactional traffic* is the naked coherence requests and the associated responses. As observed in Figure 4, abortive transactional communication accounts for an average

Fig. 4. Breakdown of on-chip network traffic.

Fig. 5. Format of CT-Register.

87.6% of the total transactional on-chip communication in four high-contention applications. Across all eight applications, 56.7% of the transactional on-chip communication is abortive. This observation indicates the significance as well as the benefit of mitigating false forwarding, as a substantial portion of abortive transactional communication is contributed by false forwarding.

3. IN-NETWORK FILTERING OF TM TRAFFIC

We propose the in-network filtering scheme to reduce the communication cost in transaction execution through mitigating false forwarding and false blocking. The scheme is based on the notion of TM and NOC co-design. In subsequent discussion, the filtering algorithm and supporting hardware mechanisms are presented, followed by further elaboration on design decisions and system-wide impact.

3.1. Conflict Trace

We define a *conflict trace* as the sufficient yet minimal piece of information to (i) describe conflicts among transactions and (ii) enable other system components (e.g., on-chip routers) to detect potential conflicts. A generic conflict trace consists of:

- Address of the memory block in the conflict.
- Metadata (e.g., priority and host node) of the transaction that is given priority in a conflict resolution.
- Data Access Status (DAS) of the memory block, specifying whether the transaction with higher priority holds the block in read-shared or write-exclusive state.

The L1 cache controller is augmented with a set of Conflict Trace Registers (CT-Registers) to record conflicts encountered by the outstanding requests. Figure 5 depicts the CT-Register. Every outgoing coherence request is assigned a CT-Register. If the request is nacked due to a conflict, the conflict trace obtained from the NACK message is stored into the associated CT-Register. The extension to NACK messages to supply all the needed pieces of information to construct conflict traces will be discussed below. When multiple NACKs to a request are received, the conflict trace from the latest NACK overwrites the previous one in the associated CT-Register. The number of CT-Registers

is bounded by the number of outstanding data requests that miss in the last-level private cache. As processors usually support a limited number of outstanding misses (e.g., the latest Intel Itanium 9500 series supports 32 [Intel 2012]), the area overhead of CT-Registers remains moderate. For smaller area and power profile, a small RAM could be used instead of registers.

To enable a network to track and predict conflicts between transactions, the conflict resolution policy used by the HTM should be straightforward for the NOC to adopt. Without loss of generality, subsequent discussion uses time-based conflict resolution [Rajwar and Goodman 2002]. Conflicts are resolved by stalling or aborting the younger transaction in favor of the older one. Each transaction is assigned a timestamp when it begins. The timestamp is attached to all the intertransaction communication (coherence messages). Besides ensuring forward progress and providing good performance [Scherer III and Scott 2005], the time-based policy provides a global transaction ordering that is straightforward for the on-chip network to identify when detecting potential conflicts.

3.2. Communicating Conflict Trace

Coherence messages from transactions are injected into the network as HTMs piggy-back onto the cache coherence protocol to detect conflicts. Furthermore, the on-chip routers can easily examine the in-transit coherence messages. Thus, the coherence messages are cost-effective mechanisms for delivering the conflict traces from HTMs to the routers. For this purpose, three coherence messages are extended. The message extensions do not change the protocol behaviors that are originally implemented in the multiprocessor.

First, the NACK messages from the nacker transactions to the conflicting transactions contain almost all the information (i.e., memory address, timestamp and host node of the nacker transaction) to construct conflict traces. A single DAS bit is added to the NACK message to specify whether the data in conflict is currently read-shared or written-exclusive by the nacker transaction. Besides, a single BYRTR (By Router) bit is also added to indicate whether the NACK is initiated from routers or not, as TMNOC allows the routers to nack requests (as described later). When a destination node receives a NACK with BYRTR set, the coherence controller at the destination neither waits for acknowledgements from other nodes nor does it send an UNBLOCK message to the directory. In this particular case, the request is nacked by an enroute router and has not yet been serviced at the directory. Second, the UNBLOCK message, which is destined to the directory to conclude a request, is extended to carry the content of the CT-Register associated with the request. A VBIT (valid bit) is needed because the embedded conflict trace is valid only if the request is nacked by a transaction due to conflict. Third, as the network attempts to regulate TM traffic, transactional requests must be distinguished from non-transactional requests. A 1-bit TXREQ (transactional request) is attached to coherence request messages (e.g., GETS and GETX). TM requests have TXREQ set to 1. Due to the wide on-chip channels, the extended messages can still be encapsulated into short packets. Therefore, the cost is minimized.

3.3. In-Network Buffering of Conflict Trace

The on-chip routers examine the in-transit coherence messages to extract and store conflict traces. For this purpose, each router is augmented with a Conflict Trace Buffer (CT-Buffer) (see Figures 6(a) and 6(b)). The CT-Buffer is the key structure to couple the on-chip networks with transaction execution. Each CT-Buffer entry stores a piece of conflict trace regarding a memory block. The time when the conflict trace arrives is recorded to handle replacement and improve prediction accuracy (as described below).

Fig. 6. (a) Router microarchitecture (TMNOC-specific structures in bold rectangles); (b) Router pipeline organization (TO: TMNOC Operation).

In addition, each line is augmented with a valid bit. The CT-Buffer uses 2-way set-associative mapping. To reduce energy and area overhead, the conflict traces in the CT-Buffer can be shared by all input ports in a router. However, the number of the CT-Buffer's read/write ports can be less than the number of input ports in a router if the area budget is tight, as the probability that packets at the head of multiple input ports are all transactional requests and those requests incur accesses to the CT-Buffer in the same cycle is relatively low. In the rare case of contention on a read/write port, the overflowed requests are just forwarded normally.

3.4. In-Network Filtering Algorithm

The in-network filtering algorithm leverages the conflict information in the CT-Buffer to perform proactive filtering on in-transit transactional requests. The algorithm is implemented in the TMNOC logic (see Figure 6(a)). The router latency is not adversely affected as the logic operates in parallel with route computation.¹ The router pipeline is presented in Figure 6(b), assuming a canonical four-stage pipeline [Enright Jerger and Peh 2009].

In-network conflict tracking. the router examines every incoming packet. If the packet carries an UNBLOCK message with a valid conflict trace (VBIT is set) and is destined to the directory on the node to which the router is attached, the embedded conflict trace is buffered in the router's CT-Buffer. If a valid conflict trace regarding the same memory block already exists, it gets updated provided that the new conflict

¹In the case of lookahead routing in some router designs, the operation of TMNOC logic can overlap with virtual channel allocation or switch allocation.

trace records a younger nacker transaction. The freshness of the conflict trace can be preserved by always tracking a younger nacker, as the conflict traces become stale if the nacker transactions have finished. If no valid conflict trace is found, the new one is buffered. CT-Buffer replacement is handled by evicting the entry with the earlier arrival time within the set of two entries. As the router tracks the conflict traces regarding only memory blocks whose home node is attached to the router, requests can be filtered away only by the home node router (i.e., the router attached to the home node). This is an intuitive design choice because the requests will always be destined to home nodes.

In the above scheme, transactional requests can be filtered out only by the home node router. Here we propose a more aggressive design that allows requests to be nacked by any enroute routers. In our discussion, the aforementioned scheme and this more aggressive scheme are referred to as *TMNOC-base* and *TMNOC-aggressive*, respectively.

The main difference between the two TMNOC variants lies in the CT-Buffer management policy. In TMNOC-aggressive, the on-chip router not only records the conflict traces embedded in the UNBLOCK messages destined to the node to which the router is attached (same as TMNOC-base) but also extracts conflict traces from any in-transit NACK messages the router has forwarded. As the routers can record conflict traces regarding any memory blocks, transactional requests could in turn be filtered out by any routers along the route to the home node. Consequently, more energy savings can be attained by further reducing the network traffic. TMNOC-aggressive needs a larger CT-Buffer because the routers are allowed to buffer conflict traces of any blocks. To alleviate buffer contention and guarantee forward progress, the routers are forbidden to extract conflict traces from NACK messages that are initiated from routers. While conflict traces are captured aggressively from enroute NACK messages, TMNOC-aggressive also actively invalidates the existing traces in the CT-Buffer. In addition to the timeout mechanism discussed before, routers use the enroute ACK messages to invalidate the buffered conflict traces. Specifically, if the nacker transaction in a recorded conflict trace sends an ACK message to requestors to the memory block, it indicates that the nacker transaction no longer conflicts with other transactions on the memory block. Thus, the corresponding conflict trace is stale and, hence, can be invalidated.

Maintain up-to-date conflict traces. As discussed, a conflict trace in the CT-Buffer becomes stale if the nacker transaction has finished, so it is important to verify that the nacker is still active. The latency and energy overhead of directly contacting the nacker is prohibitive, so TMNOC implements a local timeout mechanism to invalidate stale conflict traces. As the arrival time of each conflict trace is recorded in the CT-Buffer, the router identifies a stale conflict trace if the trace has stayed in the CT-Buffer longer than a threshold cycle count (i.e., the timeout threshold). Theoretically, the conflict trace is invalid once the conflicting transaction finishes execution. Thus, the transactions running length in terms of cycles is a close approximation of the optimal timeout threshold. In Zhao et al. [2012b], the router is augmented with a Transaction Profiling Table [Zhao et al. 2012a] that tracks the moving average of transaction length at a per-transaction basis. However, this approach incurs nonnegligible hardware costs and intensifies the communication between cores and routers. Alternatively, we propose a cost-effective approach that exploits the tight coupling between cores and routers in a tiled many-core architecture. Specifically, each router is augmented with a single 32-bit Timeout Threshold register that is mapped into the processor cores memory space. This register provides the flexibility of controlling the timeout threshold in either hardware or software. In the hardware approach, the processor tracks the transaction length (TxLen) based on the timestamps of the TBEGIN and TEND instructions. Then, the

Fig. 7. Flowchart depicting transactional request filtering. REQTYPE and TXREQ are the fields in the in-transit request message. DAS is from the matching conflict trace.

processor updates the Timeout Threshold register using the following formula.

$$TT_{new} = \frac{TT_{old} + TxLen_{latest}}{2} \quad (7)$$

Also, the register can be updated by the operating system in supervisor mode. The software can leverage more sophisticated algorithms to derive the timeout threshold.

Transaction requests filtering. Upon receiving a packet carrying a coherence request from a transaction, the TMNOC logic searches in the CT-Buffer for a conflict trace regarding the requested block. If nothing is found, the router continues forwarding the request as normal. Otherwise, TMNOC logic uses the *matching conflict trace* to predict whether the request will be rejected by the nacker transaction that is recorded in the conflict trace. The prediction requires two steps. The first step is to verify the freshness of the conflict trace using the timeout threshold. The arrival time of the conflict trace and current timestamp are used to determine whether it has been updated within the timeout period. If not, the conflict trace is invalidated, and the filtering algorithm is terminated. The algorithm is straightforward in terms of ALU operations.

The prediction proceeds to the second step if the nacker is predicted to be active. The type of the request (transactional read or write) and the data access status of the conflict trace (read-shared or written-exclusive by the nacker transaction) are used to detect a potential conflict that violates the “single-writer-multi-reader” invariant. Upon a conflict, the requester and nacker’s timestamps are compared. If the requester is older (i.e., has higher priority), the request is forwarded as normal. Otherwise, the router stops forwarding and discards the packet. Meanwhile, a router-initiated NACK message (BYRTR=1) is sent to the requester. To prevent a transaction from being nacked by itself, routers do not nack a request from the host node of the nacker transaction that is recorded in the matching conflict trace. Figure 7 depicts the procedure of request filtering logic.

3.5. Discussion

Correctness. In TMNOC, the routers filter out coherence requests that are predicted to be rejected by the nacker transactions that are recorded in the CT-Buffers. A misprediction occurs (i) when a nacker transaction is predicted inactive even though it

is still active, and (ii) when a nacker transaction is predicted active even though it has already finished. In the first case, the router forwards the request as normal. Correctness is guaranteed by the HTM system. In the second case, the router could nack the request conservatively while the request may encounter no conflicts. However, the router cannot block the request for long, as the nacker transaction is predicted inactive after the corresponding conflict trace has stayed in the router's CT-Buffer for a certain amount of time. Therefore, livelock (lack of forward progress) due to misprediction never occurs in TMNOC. Overall, as coherence requests from transactions are either forwarded to the HTM system or nacked by the routers, TMNOC does not affect the correctness of transaction execution (strong isolation and conflict serializability), which is guaranteed by the conflict detection mechanism in the HTM.

In-network vs. In-directory. The filtering provided by TMNOC-base can be placed in front of the coherence directory while the filtering provided by TMNOC-aggressive cannot. There are two main reasons to implement filtering in the network instead of in the directory. First, the on-chip routers are better poised for a fresh and broad view of conflicts through eavesdropping on the intertransaction communication. Second, filtering the traffic in-network as early as possible could achieve further energy savings and avoid disrupting the directory controller. Besides, if the directory is distributed among the home nodes, each node (instead of each router) is equipped with the CT-Buffer and logic. Therefore, the hardware cost of placing the filter in the directory will be similar to that of TMNOC. As a result, for a fixed hardware budget, placing the filter in the network can be more effective. Also, placing the filter in routers might incur even less hardware overhead in concentrated NOCs [Balfour and Dally 2006].

Scalability. The number of processing cores in the CMP architecture is expected to scale up. This trend offers more opportunities for the in-network filtering mechanism to reduce communication costs for two main reasons. First, growing core count means higher contention rate, which leads to more frequent false forwarding. Second, the cost of false forwarding increases as the average hop count increases, according to the cost function. Thus, the cost reduction due to mitigating false forwarding can potentially be more significant. If the number of cores executing transactions does not grow with the total core count, it is possible that transactional requests from the sparsely distributed "transactional cores" do not traverse the same router, which could affect the effectiveness of TMNOC-aggressive. However, TMNOC-base is not affected, as all the requests are destined to the home node. If a large portion of the cores are running transactions, it is very likely for requests from adjacent cores to share the route partially, which provides TMNOC-aggressive with abundant opportunities to filter out requests as early as possible.

Conflict resolution policy. This work assumes the timestamp-based conflict resolution policy, which provides fairness and has been used in the HTM implementation of IBM Blue Gene Q chip [Wang et al. 2012]. Similar to the timestamp-based policy, many conflict resolution policies rely on transaction metadata that can be determined at the beginning of a transaction. Such metadata includes programmer/compiler-specified priority as well as abort/success rate, among others. For these policies, the TMNOC scheme can make filtering decisions with good accuracy (as shown in next section) as the outcome of dueling transactions is consistent throughout the lifetime of the transaction instance. On the other hand, there is another category of conflict resolution policies that leverages certain dynamic metadata of transactions (e.g., amount of memory access, number of enemy transaction). This type of information can also be embedded in the coherence messages and be captured by the routers to predict conflicts. However, the prediction accuracy could be compromised due to the dynamic nature of the information. We leave a comprehensive study of the conflict resolution policy's impact as future work.

Table I. System Configuration

Unit	Value
Core	in-order, single-issue, 16 SPARC V9 cores
L1 Cache	32KB, 4-way associative, write-back, 1-cycle
L2 Cache	8MB, 8-way associative, 20-cycle latency, 16 banks
Coherence	MESI protocol, static cache bank directory
Memory	4GB, 4 memory controller, 200-cycle latency
Network	4x4 2D mesh, DOR, 1-cycle link
Router	4-cycle, 5 vnet, 4 VCs/vnet, 4-flit/VC
TMNOC	32-entry CT-Buffer per router

Table II. Benchmark Input Parameters and Characteristics

Benchmark	Input Parameter	Tx Time	Abort Rate	r-set size	w-set size	Avg Tx Length (cycles)
Bayes	32 var, 1024 records, 2 edge/var	90.58%	96.53%	93.19	50.37	125,945
Intruder	2k flow, 10 attack, 4 pkt/flow	75.83%	56.08%	7.47	3.50	1,490
Labyrinth	32*32*3 maze, 96 paths	99.92%	98.72%	145.71	92.28	373,619
Yada	1,264 elements, min-angle 20	99.69%	38.63%	52.96	31.61	35,854
Genome	32 var, 1024 records	79.07%	3.06%	32.39	3.32	4,792
Kmeans	16K seg. 256 gene. 16 sample	7.12%	4.69%	6.23	1.75	306
SSCA2	8k nodes, 3 len, 3 para edge	10.22%	0.33%	2.99	1.99	131
Vacation	16K record. 4K req. 60% coverage	97.38%	46.24%	64.63	18.62	17,002

4. EVALUATION

4.1. Methodology

The efficacy of the in-network filtering mechanism is evaluated with cycle-accurate full-system simulation using SIMICS [Magnusson et al. 2002] and the Ruby memory model [Martin et al. 2005]. Garnet [Agarwal et al. 2009a] is used to model the timing of the on-chip network, while the DSENT power model [Sun et al. 2012] is used to estimate the energy consumption of the routers and links in the network.

The baseline tiled CMP architecture is used in our experiment. Each of the 16 nodes consists of an in-order SPARC core with a private L1 and a shared L2. The shared L2 is organized as a static nonuniform cache architecture that uses the directory-based MESI protocol to maintain coherence. The width of coherence control messages is 64 bits. The L2 cache-line tags are augmented with directory entry state. The system configuration is listed in Table I. Results are presented for all eight workloads in the STAMP benchmark suite [Minh et al. 2008] that is widely used to evaluate HTM designs. Table II lists the input parameter and characteristics of each benchmark.

The processor core provides support for log-based HTM similar to the FASTM [Lupon et al. 2009]. Pretransaction states are written to a software managed log, while speculative states are propagated to the memory hierarchy eagerly. Pretransaction states are also stored to a dedicated buffer for fast abort recovery. Conflicts are detected eagerly leveraging the existing coherence protocol. Detected conflicts are resolved using the time-based policy [Rajwar and Goodman 2002], which favors older transactions. After receiving a NACK, transactions back off for a fixed period of 20 cycles before retrying. As conflicts are detected progressively as transactions execute, no commit time validation is necessary. Multiple transactions can commit at the same time. The performance of the baseline HTM is comparable to contemporary high-performance HTM designs.

Both base and aggressive filtering schemes are modeled in the simulator. The Garnet router model is augmented with the TMNOC logic and CT-Buffer. Since the TMNOC logic works in parallel with route computation, the router latency is not affected. For

Fig. 8. Normalized cycle count when the directory is busy serving transactional requests (B: baseline w/o TMNOC; T: TMNOC-base; T+: TMNOC-aggressive).

energy estimation, we implemented and synthesized the TMNOC design in 40nm technology. The power dissipation of the SRAM-based CT-Buffer is estimated using CACTI [Muralimanohar et al. 2007]. Based on the obtained results, we modified DSENT [Sun et al. 2012] to carefully account for the energy overhead of TMNOC in 40nm technology and 0.9V on-chip voltage. A flit size of 128 bits is used in the simulations as most current NOCs have 128-bit or 256-bit channel width. Therefore, no extra flit is needed as the flit size is large enough to accommodate the extended fields in the coherence messages. Due to the dynamic nature of multi-threaded execution, we run each simulation 20 times with random seeds to collect the data. The error bar indicates the variation of the data. We refer to TMNOC-base as TMNOC, and TMNOC-aggressive as TMNOC+ in the figures.

4.2. Reduction in Directory Blocking

As the coherence directory serializes requests to the same address, directory blocking caused by one request has a severe impact on the overall throughput of the memory hierarchy. The pathological false blocking in transaction execution could further degrade the throughput. Figure 8 shows the impact of TMNOC on mitigating false blocking of the coherence directory. The number of cycles the directory is blocked by coherence requests from transactions are obtained by accumulating the cycles during which directory entries stay in the busy transient state while servicing transactional requests. It is observed that TMNOC-base reduces the TM-induced directory blocking by 43% on average and up to 87%. TMNOC-aggressive reduces the blocking by 66% on average and up to 88%. Another observation is that high-contention benchmarks show a significant reduction in the cycles the directory is blocked by transactional write requests. This observation indicates that a large portion of transactional write requests are filtered out as transactions in high-contention benchmarks tend to update shared data frequently. The reduced directory blocking also indicates less coherence activity due to the in-network filtering. The reduced coherence activity indicates less cache access and reduced processing by the cores to perform conflict detection against the incoming requests. Therefore, the overall system energy consumption is expected to be reduced.

4.3. Reduction in Network Energy

One primary goal of this work is to reduce energy consumption of the on-chip network in supporting transaction execution. Figure 9 shows the normalized energy consumption

Fig. 9. Normalized network energy.

of the network including routers and links. As observed, TMNOC-base reduces the network energy consumption in high-contention benchmarks by 20%, on average (up to 35%), while TMNOC-aggressive reduces the figure by 24% (up to 39%). Across all the benchmarks, TMNOC-base and TMNOC-aggressive reduces average network energy consumption by 12% and 15%, respectively. The energy savings of TMNOC-base are achieved by eliminating the coherence cost (multicasting to several nodes) in the cost model, whereas TMNOC-aggressive achieves additional savings by further reducing the inherent cost (unicast between two nodes). The reduced traffic is directly translated into dynamic energy reduction. As coherence cost is usually much larger than the inherent cost, TMNOC-base can achieve much of the energy savings with relatively small incremental benefits resulting from TMNOC-aggressive. However, the inherent cost grows rapidly with the number of processing nodes in a NOC. Thus, TMNOC-aggressive has more energy-saving potential than TMNOC-base in large-scale CMP processors. Moreover, it is worth noting that TMNOC-aggressive does not incur extra overhead for the extra energy savings.

High-contention benchmarks exhibit more energy savings for two reasons. First, high-contention benchmarks have more requests being nacked causing more energy waste due to false forwarding (see Figure 3) in the baseline system, which offers TMNOC more energy saving opportunities through mitigating false forwarding. Second, frequent conflicts provide the routers with plenty of information about transaction conflicts, hence increasing the prediction accuracy of the TMNOC logic. Besides contention rate, the type of the filtered-out requests also affects how much energy could be saved. For instance, in the Vacation benchmark, TMNOC-base filters out a large portion of transactional reads according to Figure 8. However, the energy savings is marginal since GETS requests are not the major source of energy waste in false forwarding. On the other hand, Bayes has an energy reduction of 38% because a large number of transactional writes in Bayes are filtered out by TMNOC. Otherwise, those transactional writes would initiate extensive communication between multiple nodes before being eventually nacked, which wastes a considerable amount of energy. Overall, both TMNOC variants achieve the goal of improving NOC energy efficiency.

4.4. Reduction of Network Traffic

The interconnection traffic has a fundamental impact on the network energy consumption. Figure 10 shows the normalized interconnection traffic measured in router traversals by flits. It is observed that TMNOC-base reduces the interconnection traffic in high-contention benchmarks by 16%, on average (up to 28%), while TMNOC-aggressive reduces the figure by 24%, on average (up to 39%). Across all the

Fig. 10. Normalized network traffic.

Fig. 11. Hop count distribution (measured in router traversals by flits).

benchmarks, TMNOC-base and TMNOC-aggressive reduce interconnection traffic by 10% and 12%, respectively. The traffic cost of false forwarding falls into the abortive traffic category shown in Figure 4 because false forwarding does not contribute to the continued execution of transactions. As the TMNOC mitigates false forwarding, the abortive traffic due to false forwarding is reduced. The reduction in interconnection traffic translates directly into energy savings. Moreover, the reduced HTM traffic indicates improved QoS of the network for non-HTM traffic (e.g., coherence traffic to support concurrent STM execution in a hybrid TM system).

Figure 11 shows the distribution of network flits according to their hops. It is observed that both TMNOC variants reduce the proportion of long-distance flits through proactive filtering while increasing the proportion of short-distance flits. This trend is particularly noticeable in applications with high contention, which hence exhibit substantial reductions in network traffic. TMNOC-aggressive further increases the proportion of 1- and 2-hop flits and reduces flits with a large hop count. This observation demonstrates the effectiveness of the aggressive scheme in filtering out in-transit requests early on before they arrive at the home node. The capability of converting

Fig. 12. Normalized execution time.

long-distance flits to short-distance ones based on application-level information reduces the average hop count even if the network topology remains the same. As shown in the cost model of Section 2.3, the communication cost is linearly proportional to the average hop count. Thus, reducing the average hop count can reduce the bandwidth utilization and enable the network to be more responsive and energy efficient. This observation demonstrates the effectiveness of TMNOC in regulating network traffic. As CMPs are increasingly distributed, the impact of in-network filtering on long-distance flits as well as network traffic will become growingly substantial.

4.5. Impact on Performance

Although TMNOC shows the potential to increase concurrency in the memory system, the proactive filtering could nack a transaction's request conservatively, thereby stalling the transaction needlessly. This situation happens when the router decides to nack a request based on a previous NACK from a transaction that has already finished. Such conservative nacks may degrade overall performance and potentially offset the benefit of increased concurrency in the memory system. Figure 12 shows the normalized execution time. It is observed that TMNOC does not impose a performance penalty on the system in order to regulate the network traffic in transactional systems. On the contrary, Bayes and Intruder exhibit performance improvements of 18% and 12%, respectively, indicating further static energy savings, as shown in Figure 9. These performance improvements stem from the fact that TMNOC reduces the contention on the directory by mitigating false blocking. Workloads with a small set of memory addresses being contended fiercely among transactions (i.e., memory hotspots) benefit the most from the alleviation of false blocking, as requests to the hotspot are serviced more promptly instead of being blocked unnecessarily. Bayes and Intruder are two such workloads. Although Yada has a high contention rate, it shows negligible improvement in performance, as it does not exhibit the bottleneck of a few memory hotspots [Zhao et al. 2012a]. In Labyrinth, each transaction reads the entire global maze grid at the beginning and writes to part of the grid at the end. This sharing pattern effectively serializes the transaction execution preventing the workload from taking advantage of the reduced directory contention. Due to the in-order execution and well-optimized parallel applications in our experiment, the memory subsystem is not fully stressed. Consequently, the reduction of directory busy cycles is not fully translated into performance improvement. Nevertheless, CMPs are expected to scale up substantially in the number of out-of-order cores for more memory-level parallelism, and contention on shared data will inevitably become increasingly intensive. This trend implies more performance improvement potential for TMNOC.

Fig. 13. Performance vs. number of CT-Buffer entries.

4.6. Sensitivity Study

CT-Buffer size. The microarchitecture design trade-off between performance and hardware overhead is mainly affected by the size of the CT-Buffer. The CT-Buffer tracks conflicts between transactions on memory blocks. If the router observes multiple transactions conflicting on the same memory block, the CT-Buffer tracks the conflict with one entry, storing the address to the memory block and the highest-priority transaction in the conflict. Thus, the optimal size of CT-Buffer is essentially determined by the number of “conflict hotspot” in the workload. If the conflict hotspot is a global data structure, a larger number of transactions/cores does not necessarily put more pressure on the CT-Buffer capacity. However, when multiple transactions are working on sub-blocks of shared data, conflict could occur due to overstepping. In this case, increasing the number of transactions/cores demands a larger CT-Buffer as conflicting memory blocks grows with the transaction count. We explore the sensitivity of the TMNOC to the size of the CT-Buffer in terms of overall execution time using the 16-core configuration. As CT-Buffer read/write operations are not on the router critical path (see Section 3.4), the increased access latency due to a larger CT-Buffer does not affect the router latency. Figure 13 shows the impact of CT-Buffer size on the overall execution time. It is observed that the majority of the benchmarks, especially those with low contention rates, are not sensitive to the size of the CT-Buffer. This is mainly due to the fact that those benchmarks have a small set of memory hotspots. Bayes sees a 10% performance improvement when the buffer size is increased from four to eight, as its large transactions conflict on more memory hotspots. As Bayes is particularly sensitive to transaction interleaving, the change of buffer size that leads to the change of transaction interleaving causes a performance variation (within in 5%) when further increasing the buffer size. For the TM workloads evaluated, a small CT-Buffer size is sufficient to achieve significant energy savings and effective traffic regulation.

CT-Buffer timeout threshold. Recall that the router leverages a timeout mechanism to invalidate stale conflict traces in the CT-Buffer. A timeout threshold value too small reduces the effectiveness of TMNOC by limiting routers’ capabilities to identify conflicts, whereas unnecessarily large thresholds introduce unwarranted NACKs from the routers. Theoretically, the lifetime of a conflict trace should be no longer than the lifetime of the conflicting transaction. Here, we study the sensitivity of TMNOC to the timeout threshold by disabling the dynamic approach and assigning a static threshold value instead. The results are presented in Figure 14. All the results are normalized to

Fig. 14. Performance vs. timeout threshold.

Table III. Result of Area Overhead Estimation

Components	Estimated Area (μm^2)
Baseline Router	145,901
Conflict Trace Cache	6,563
TMNOC Logic	162
Overhead	4.6%

the baseline. Two observations can be made. First, both the execution time and network traffic are relatively insensitive to the timeout threshold when its value is less than 2,000 cycles. Two applications (Intruder and Kmeans) exhibit substantial degradation as the threshold value increases beyond 2,000. As these two applications mainly consist of fine-grain transactions that typically finish within 1,000 cycles, a large timeout threshold gives the conflict traces an unnecessarily longer lifetime than the actual conflicting transactions themselves, thereby introducing more stale conflict traces. The second observation is that no single threshold value can deliver good performance across the full spectrum of applications that are evaluated in our experiments. This observation emphasizes the need to dynamically determine the timeout threshold.

4.7. Area and Energy Overhead

The additional storage and processing logic in the on-chip routers introduce area overhead. We estimate the area of the CT-Buffer using a commercial memory compiler. The buffer is implemented as a 32x64-bit dual-port SRAM. We implement the TMNOC logic at the RTL level. The virtual channel router implementation is based on the open-source design from Stanford University [Becker 2012]. The router configurations are identical to those used in the full-system simulation, as shown in Table I. The design is synthesized using Synopsys Design Compiler targeting TSMC 40nm technology. The clock frequency is set to 1GHz. Table III reports the estimated area overhead of TMNOC. TMNOC incurs a reasonable 4.6% area overhead to the virtual channel router. This area overhead is justified by the energy savings and performance improvement.

The CT-Buffer incurs an energy overhead of 1.95% to the router. The network energy in Figure 9 already accounts for this added overhead. This net positive reduction of network energy indicates that the network energy savings exceeds the energy overhead of the filtering mechanism.

5. RELATED WORK

The proposed in-network filtering mechanism is extended from the base scheme presented in Zhao et al. [2013] with two main improvements. First, the conflict tracking mechanism leverages not only the NACK but also the ACK message to determine the lifetime of a conflict so as to increase accuracy. It further demonstrates the on-chip router's capability of exploiting application-specific information in network messages to optimize bandwidth utilization. The second improvement is in the timeout mechanism of conflict traces. The new mechanism enables the processor or software-stack to update the timeout as they usually have a well-optimized solution at hand for transaction profiling (e.g., hardware event counter and software-based profiler). The mechanism in this article inherits the name TMNOC in the conference paper. The base scheme does not see an improvement in network energy saving over the improved design, as the home node will nevertheless invalidate stale conflict trace when the requester unblocks the directory upon receiving an ACK. On the other hand, the aggressive scheme achieves additional energy savings (2%–3%) with the improved design in certain high contention workloads (Intruder and Labyrinth). For the majority of workloads in the STAMP benchmark, the number of static transactions and the lifetime variation of their dynamic instances are small. Thus, the timeout mechanism can predict conflicts fairly accurately without the assist of ACK message. As future TM workloads incorporate more transactions with frequent conflicts, the execution will be increasingly dynamic. The additional information from ACK messages will be growingly important for the network to maintain the accuracy of conflict traces.

Techniques to regulate coherence traffic. Two types of coherence protocols, namely snooping protocol and directory protocol, are widely adopted in shared memory multiprocessors. For snooping protocols, various hardware filtering mechanisms have been proposed. Early works focus on source and destination filtering. In Martin et al. [2003], the source node predicts the set of nodes that should observe the request before multicasting the request, thus avoiding broadcasting across the entire chip. Destination filtering [Moshovos et al. 2001; Salapura et al. 2008] uses local filtering information to filter away snoop requests that will miss in the local cache. Thus, cache-tag lookups are avoided to save energy and reduce cache port contention. Recent work proposes to filter redundant coherence traffic in-network [Agarwal et al. 2009b] by augmenting on-chip routers with coherence filters that track region-level sharing information. The in-network filtering mechanism requires routers to exchange sharing information explicitly through dedicated physical links, which has power and area implications. The above filtering mechanisms work only on snooping protocols and thus are not applicable to directory protocols which are used by most HTM designs. As for directory protocols, Marty and Hill [2007] exploit the memory access isolation across VMs in virtualized systems to reduce coherence traffic in a two-level directory protocol. Proximity coherence [Barrow-Williams et al. 2010] optimistically forwards L1 load misses to nearby caches via new dedicated links. If nearby caches can satisfy the request, network traffic and L1 miss latency are reduced. Despite their effectiveness in reducing coherence traffic, these mechanisms do not distinguish between TM and non-TM traffic and, therefore, cannot use the HTM-specific information to reduce network traffic, whereas routers in TMNOC track the sharing information (conflict traces) through monitoring the intertransaction communication and exploit the information to regulate coherence traffic from transactions.

Besides snooping and directory protocols, there are other novel mechanisms to provide cache coherence. For example, Enright Jerger et al. [2008] uses virtual trees to connect and order sharers. Coherence requests are multicast through the virtual trees to reduce network traffic due to broadcasting. However, it does not target HTM and,

if adopted by HTM designs, cannot reduce wasted network traffic caused by false forwarding.

Techniques to predict conflicts. Various techniques for conflict prediction are proposed to proactively avoid transaction conflicts in HTM. In particular, the Adaptive Transaction Scheduling (ATS) [Yoo and Lee 2008] technique uses the local commit/abort history to calculate the per-transaction conflict pressure. Transactions with high conflict pressure are serialized through a central waiting queue to avoid potential conflicts. ATS and TMNOC are complementary techniques to reduce the HTM network traffic, as ATS can reduce the number of request retry (n_{retry}), while TMNOC mainly reduces the number of request forwarding (n_{fwd}). On the other hand, Proactive Transaction Scheduling (PTS) [Blake et al. 2009] and Bloom Filter Guided Transaction Scheduling (BFGTS) [Blake et al. 2011] use a software graph structure to track the likelihood of conflicts between transactions. Bloom filters are used to track the read/write set of individual transactions. A nonnull intersection of the bloom filters of two serialized transactions cause an increase in the confidence a conflict will occur between the two transactions. Nonetheless, these two techniques are not suited for on-chip routers due to the storage overhead and the graph-scanning latency in each conflict detection.

6. CONCLUSION

In this work, we develop a cost model of on-chip communication for HTM. The model identifies the key contributing factors to the communication overhead of transaction execution. This model is used to analyze a network traffic characterization of a HTM system and isolates the pathological false forwarding incurred by failed transactional requests. False forwarding is an excessive waste of network bandwidth. To mitigate false forwarding, we propose a novel in-network filtering mechanism that enables the on-chip routers to exploit the transaction conflict information in network messages to predict the probability of a request being unsuccessful. Requests that have a high probability to fail are filtered out in-network as early as possible to reduce their network bandwidth utilization. Evaluation results from full system simulation show that the proposed mechanism is capable of reducing 24% of the network traffic on average over a set of high-contention benchmarks, which is translated into an average energy savings of 24% and a directory contention reduction of 68%. Implemented TMNOC mechanisms result in less than a 5% area overhead to a conventional NOC router.

REFERENCES

- Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraj K. Jha. 2009a. GARNET: A detailed on-chip network model inside a full-system simulator. In *Proceedings of International Symposium on Performance Analysis of Systems and Software*. DOI: <http://dx.doi.org/10.1109/ISPASS.2009.4919636>
- Niket Agarwal, Li-Shiuan Peh, and Niraj K. Jha. 2009b. In-network coherence filtering: Snoopy coherence without broadcasts. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 42)*. ACM, New York, NY, 232–243. DOI: <http://dx.doi.org/10.1145/1669112.1669143>
- James Balfour and William J. Dally. 2006. Design tradeoffs for tiled CMP on-chip networks. In *Proceedings of the 20th Annual International Conference on Supercomputing (ICS'06)*. DOI: <http://dx.doi.org/10.1145/2591635.2667187>
- Nick Barrow-Williams, Christian Fensch, and Simon Moore. 2010. Proximity coherence for chip multiprocessors. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (PACT'10)*. ACM, New York, NY, 123–134. DOI: <http://dx.doi.org/10.1145/1854273.1854293>
- Daniel U. Becker. 2012. *Efficient Microarchitecture for Network-on-Chip Routers*. Ph.D. Dissertation. Stanford University.
- Geoffrey Blake, Ronald G. Dreslinski, and Trevor Mudge. 2009. Proactive transaction scheduling for contention management. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 42)*. DOI: <http://dx.doi.org/10.1145/1669112.1669133>

- Geoffrey Blake, Ronald G. Dreslinski, and Trevor Mudge. 2011. Bloom filter guided transaction scheduling. In *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture (HPCA11)*.
- Colin Blundell, Joe Devietti, E. Christopher Lewis, and Milo M. K. Martin. 2007. Making the fast case common and the uncommon case simple in unbounded transactional memory. In *Proceedings of 34th International Symposium on Computer Architecture*. DOI : <http://dx.doi.org/10.1145/1250662.1250667>
- Hassan Chafi, Jared Casper, Brian D. Carlstrom, Austen McDonald, Chi Cao Minh, Woongki Baek, Christos Kozyrakis, and Kunle Olukotun. 2007. A scalable, non-blocking approach to transactional memory. In *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture (HPCA'07)*. DOI : <http://dx.doi.org/10.1109/HPCA.2007.346189>
- Natalie Enright Jerger and Li-Shiuan Peh. 2009. *On-Chip Networks* (1st ed.). Morgan Claypool.
- Natalie Enright Jerger, Li-Shiuan Peh, and Mikko H. Lipasti. 2008. Virtual tree coherence: Leveraging regions and in-network multicast trees for scalable cache coherence. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 41)*. IEEE Computer Society, Washington, DC, 35–46. DOI : <http://dx.doi.org/10.1109/MICRO.2008.4771777>
- Ruud Haring, Martin Ohmacht, Thomas Fox, Michael Gschwind, David Satterfield, Krishnan Sugavanam, Paul Coteus, Philip Heidelberger, Matthias Blumrich, Robert Wisniewski, alan gara, George Chiu, Peter Boyle, Norman Chist, and Changhoan Kim. 2012. The IBM blue gene/Q compute chip. *IEEE Micro* 32, 2 (March 2012), 48–60. DOI : <http://dx.doi.org/10.1109/MM.2011.108>
- Maurice Herlihy and J. Eliot B. Moss. 1993. Transactional memory: Architectural support for lock-free data structures. In *Proceedings of the 20th Annual International Symposium on Computer Architecture (ISCA'93)*. ACM, New York, NY, 289–300. DOI : <http://dx.doi.org/10.1145/165123.165164>
- Intel. 2012. Intel Itanium Processor 9500 Series Reference Manual. Retrieved from <http://www.intel.com/content/www/us/en/processors/itanium/itanium-9500-reference-manual.html>.
- Marc Lupon, Grigorios Magklis, and Antonio Gonzalez. 2009. FASTM: A log-based hardware transactional memory with fast abort recovery. In *Proceedings of the 18th International Conference on Parallel Architectures and Compilation Techniques*. DOI : <http://dx.doi.org/10.1109/PACT.2009.19>
- Marc Lupon, Grigorios Magklis, and Antonio Gonzalez. 2010. A dynamically adaptable hardware transactional memory. In *Proceedings of the 43rd International Symposium on Microarchitecture*. DOI : <http://dx.doi.org/10.1109/MICRO.2010.23>
- Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hällberg, Johan Högberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner. 2002. Simics: A full system simulation platform. *Computer* 35, 2 (February 2002), 50–58. DOI : <http://dx.doi.org/10.1109/2.982916>
- Milo M. K. Martin, Pacia J. Harper, Daniel J. Sorin, Mark D. Hill, and David A. Wood. 2003. Using destination-set prediction to improve the latency/bandwidth tradeoff in shared-memory multiprocessors. In *Computer Architecture, 2003. Proceedings of the 30th Annual International Symposium on Computer Architecture*. 206–217. DOI : <http://dx.doi.org/10.1109/ISCA.2003.1207001>
- Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood. 2005. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *SIGARCH Comput. Archit. News* 33 (November 2005). Issue 4. DOI : <http://dx.doi.org/10.1145/1105734.1105747>
- Michael R. Marty and Mark D. Hill. 2007. Virtual hierarchies to support server consolidation. In *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA'07)*. ACM, New York, NY, 46–56. DOI : <http://dx.doi.org/10.1145/1250662.1250670>
- Chi Cao Minh, JaeWoong Chung, C. Kozyrakis, and K. Olukotun. 2008. STAMP: Stanford transactional applications for multi-processing. In *Proceedings of International Symposium on Workload Characterization*.
- Andreas Moshovos, Gokhan Memik, Alok Choudhary, and Babak Falsafi. 2001. JETTY: Filtering snoops for reduced energy consumption in SMP servers. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture (HPCA'01)*. IEEE Computer Society, Washington, DC, 85.
- Naveen Muralimanohar, Rajeev Balasubramonian, and Norm Jouppi. 2007. Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0. In *Proceedings of the 40th International Symposium on Microarchitecture*. DOI : <http://dx.doi.org/10.1109/MICRO.2007.30>
- Anurag Negi, Ruben Titos-Gil, Manuel E. Acacio, Jose M. Garcia, and Per Stenstrom. 2012. pi-TM: Pessimistic invalidation for scalable lazy hardware transactional memory. In *Proceedings of the 18th International Symposium on High Performance Computer Architecture*. DOI : <http://dx.doi.org/10.1109/HPCA.2012.6168951>
- Ravi Rajwar and James R. Goodman. 2002. Transactional lock-free execution of lock-based programs. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X)*. DOI : <http://dx.doi.org/10.1145/605397.605399>

- Ravi Rajwar, Maurice Herlihy, and Konrad Lai. 2005. Virtualizing transactional memory. In *Proceedings of the 32Nd Annual International Symposium on Computer Architecture (ISCA'05)*. DOI: <http://dx.doi.org/10.1109/ISCA.2005.54>
- Christopher J. Rossbach, Owen S. Hofmann, Donald E. Porter, Hany E. Ramadan, Bhandari Aditya, and Emmett Witchel. 2007. TxLinux: Using and managing hardware transactional memory in an operating system. In *Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP'07)*. DOI: <http://dx.doi.org/10.1145/1294261.1294271>
- Valentina Salapura, Matthias Blumrich, and Alan Gara. 2008. Design and implementation of the blue gene/P snoop filter. In *Proceedings of the IEEE 14th International Symposium on High Performance Computer Architecture*. 5–14. DOI: <http://dx.doi.org/10.1109/HPCA.2008.4658623>
- Daniel Sanchez, Luke Yen, Mark D. Hill, and Karthikeyan Sankaralingam. 2007. Implementing signatures for transactional memory. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 40)*. DOI: <http://dx.doi.org/10.1109/MICRO.2007.24>
- William N. Scherer III and Michael L. Scott. 2005. Advanced contention management for dynamic software transactional memory. In *Proceedings of the 24th Symposium on Principles of Distributed Computing*. DOI: <http://dx.doi.org/10.1145/1073814.1073861>
- Arrvinth Shriraman and Sandhya Dwarkadas. 2009. Refereeing conflicts in hardware transactional memory. In *Proceedings of the 23rd International Conference on Supercomputing (ICS'09)*. DOI: <http://dx.doi.org/10.1145/1542275.1542299>
- Arrvinth Shriraman, Sandhya Dwarkadas, and Michael L. Scott. 2008. Flexible decoupled transactional memory support. In *Proceedings of the 35th Annual International Symposium on Computer Architecture (ISCA'08)*. DOI: <http://dx.doi.org/10.1109/ISCA.2008.17>
- Michael F. Spear, Virendra J. Marathe, Luke Dalessandro, and Michael L. Scott. 2007. Privatization techniques for software transactional memory. In *Proceedings of the 26th Annual ACM Symposium on Principles of Distributed Computing (PODC'07)*. ACM, New York, NY, 338–339. DOI: <http://dx.doi.org/10.1145/1281100.1281161>
- Chen Sun, Chia-Hsin Owen Chen, George Kurian, Lan Wei, Jason Miller, Anant Agarwal, Li-Shiuan Peh, and Vladimir Stojanovic. 2012. DSENT - A tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *Proceedings of the 2012 6th IEEE/ACM International Symposium on Networks on Chip*. DOI: <http://dx.doi.org/10.1109/NOCS.2012.31>
- Saša Tomić, Cristian Perfumo, Chinmay Kulkarni, Adrià Armejach, Adrián Cristal, Osman Unsal, Tim Harris, and Mateo Valero. 2009. EazyHTM: Eager-lazy hardware transactional memory. In *Proceedings of the 42nd International Symposium on Microarchitecture*. DOI: <http://dx.doi.org/10.1145/1669112.1669132>
- Marc Tremblay and Shailender Chaudhry. 2008. A third-generation 65nm 16-core 32-thread plus 32-scout-thread CMT SPARC processor. In *Proceedings of the IEEE International Solid-State Circuits Conference. Digest of Technical Papers*. DOI: <http://dx.doi.org/10.1109/ISSCC.2008.4523067>
- Amy Wang, Matthew Gaudet, Peng Wu, José Nelson Amaral, Martin Ohmacht, Christopher Barton, Raul Silvera, and Maged Michael. 2012. Evaluation of blue gene/Q hardware support for transactional memories. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT'12)*. DOI: <http://dx.doi.org/10.1145/2370816.2370836>
- Luke Yen, Jayaram Bobba, Michael R. Marty, Kevin E. Moore, Haris Volos, Mark D. Hill, Michael M. Swift, and David A. Wood. 2007. LogTM-SE: Decoupling hardware transactional memory from caches. In *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture (HPCA'07)*. DOI: <http://dx.doi.org/10.1109/HPCA.2007.346204>
- Richard M. Yoo, Christopher J. Hughes, Konrad Lai, and Ravi Rajwar. 2013. Performance evaluation of Intel® transactional synchronization extensions for high-performance computing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'13)*. DOI: <http://dx.doi.org/10.1145/2503210.2503232>
- Richard M. Yoo and Hsien-Hsin S. Lee. 2008. Adaptive transaction scheduling for transactional memory systems. In *Proceedings of the 20th Annual Symposium on Parallelism in Algorithms and Architectures (SPAA'08)*. DOI: <http://dx.doi.org/10.1145/1378533.1378564>
- Lihang Zhao, Lizhong Chen, and Jeffrey Draper. 2014. Mitigating the mismatch between the coherence protocol and conflict detection in hardware transactional memory. In *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS'14)*. IEEE Computer Society, Washington, DC, 605–614. DOI: <http://dx.doi.org/10.1109/IPDPS.2014.69>
- Lihang Zhao, Woojin Choi, Lizhong Chen, and Jeffrey Draper. 2013. In-network traffic regulation for transactional memory. In *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA'13)*. DOI: <http://dx.doi.org/10.1109/HPCA.2013.6522346>

- Lihang Zhao, Woojin Choi, and Jeffrey Draper. 2012a. SEL-TM: Selective eager-lazy management for improved concurrency in transactional memory. In *Proceedings of the IEEE 26th International Parallel Distributed Processing Symposium (IPDPS)*. 95–106. DOI: <http://dx.doi.org/10.1109/IPDPS.2012.19>
- Lihang Zhao, Woojin Choi, and Jeffrey Draper. 2012b. TMNOC: A case of HTM and NoC co-design for increased energy efficiency and concurrency. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT'12)*. ACM, New York, NY. DOI: <http://dx.doi.org/10.1145/2370816.2370885>
- Lihang Zhao and Jeffrey Draper. 2014. Consolidated conflict detection for hardware transactional memory. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation (PACT'14)*. ACM, New York, NY, 201–212. DOI: <http://dx.doi.org/10.1145/2628071.2628076>

Received April 2015; revised October 2015; accepted October 2015