In-Network Traffic Regulation for Transactional Memory

Lihang Zhao[1], Woojin Choi[1], Lizhong Chen[2], Jeffrey Draper[1]
[1]Information Sciences Institute, [2]Ming Hsieh Department of Electrical Engineering
University of Southern California, Los Angeles, CA 90089
{lihangzh, woojinch, lizhongc}@usc.edu, draper@isi.edu

Abstract

Hardware Transactional Memory (HTM) promises to simplify parallel programming on shared-memory chip multiprocessors by providing atomic execution of code blocks. Concurrently, Networks-On-Chip (NOCs) have emerged as an efficient on-chip communication infrastructure but have been largely neglected in HTM designs. In this work, we explore the interaction between the HTM paradigm and NOCs. In the process, we find a huge source of unnecessary network traffic incurred by transactional requests that are unsuccessful. This problem is identified as false forwarding that adversely affects network performance and energy efficiency. Surprisingly, 39% (up to 79% for a specific workload) of the transactional requests have incurred false forwarding over a wide spectrum of workloads. To combat this problem, we propose TMNOC, a novel approach that exploits the co-design of HTM and NOCs to mitigate false forwarding. Transactional requests that have a high probability to fail are filtered out in-network as early as possible to save energy and improve concurrency in the memory system. Experimental results show that our design reduces total network traffic by 20% on average (up to 40%) for a set of high-contention benchmarks representative of future TM workloads, thereby reducing energy consumption by an average of 24% (up to 39%). Meanwhile, the contention in the coherence directory is reduced by 66% on average. These improvements are achieved with only 5% area overhead added to a conventional on-chip router design.

1. Introduction

The past decade has seen a fundamental shift from single-core to many-core architectures to harness the increasing numbers of transistors due to process scaling [9, 15]. In the many-core era, one of the grand challenges is to write parallel applications that efficiently exploit tens to thousands of processor cores. In shared memory chip multiprocessors, concurrent data accesses from different threads must be synchronized in case of data races. The task of synchronizing concurrent accesses using traditional mutex primitives is burdensome for programmers. Transactional Memory (TM) is proposed as a concurrency control mechanism to increase the productivity in parallel programming by moving the burden of shared data access synchronization to the software runtime or hardware. Our work focuses on HTM for its accelerated performance and its tight coupling with the processor architecture that enables ample in-hardware optimization opportunities.

In general, research in HTM has focused on performance [43, 8, 24, 33], implementation issues [18, 38, 6], transaction scheduling [4, 5, 39], and hardware-software interplay [41, 35, 36]. These efforts have paved the way for HTM to be implemented in commodity systems [9, 15, 12]. 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 expands beyond the processor core into the on-chip networks for two main reasons.

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

Performance. HTM designs depend on the on-chip networks to fetch data and detect conflicts. Due to the serialization of transactions in data race conditions, memory-level parallelism [10] 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 inter-transaction communication.

To this end, our work explores the interplay between HTMs and on-chip networks. In this process, we identify a source of excessive network traffic generated by the transactions. The coherence requests from transactions could be negatively acknowledged (nacked) due to conflicts with other transactions. Before a request is nacked, it can initiate numerous messages between the requester, directory and sharers across the entire chip. However, those messages do not contribute to the continued execution of transactions if the request is nacked eventually. This problem is referred to as *false forwarding*. False forwarding generates a large number of useless coherence messages and unnecessarily degrades the energy efficiency of on-chip networks as each hop of each message consumes energy in the routers and on the links. Transactions usually keep retrying the nacked requests instead of aborting to save wasted work. Since only the last one in a series of repeating retries successfully obtains the desired data access permission, the rest are nacked and thus, intensify false forwarding in the network. According to our study, 39% of the transactional requests have incurred false forwarding over a wide range

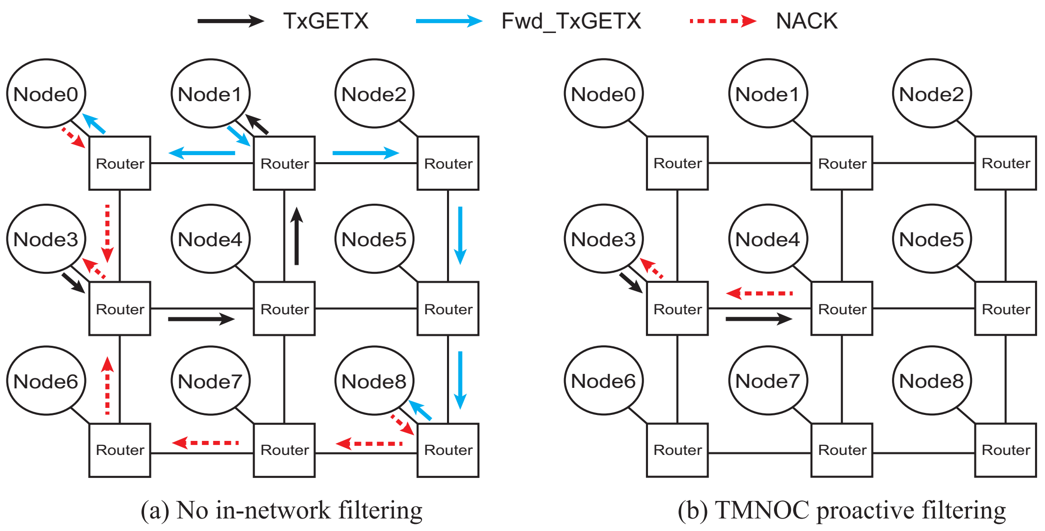


Figure 1. Network traffic in HTM conflict detection. TxGETX: transactional write request to the directory. Fwd_TxGETX: forwarded TxGETX request from the directory to the sharer. NACK: negative acknowledgement.

of workloads running on a representative HTM system. Energy waste and performance loss due to false forwarding is further exacerbated, as the number of cores scales up and coarse-grain transactions with high contention rates prevail [40]. 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.

In this paper, we introduce TMNOC, a HTM and Networks-On-Chip (NOCs) co-design to proactively filter out transactional requests that can incur false forwarding. First, a cost-effective communication mechanism is devised for the HTM and on-chip network to exchange critical information about conflicts between transactions. Second, the on-chip routers are augmented to track the conflicts by communicating with the HTM. Enabled by these two mechanisms, the network filters out transactional requests that have a high probability to be rejected by concurrent transactions. Consequently, false forwarding can be suppressed. Figure 1 illustrates the effectiveness of TMNOC on a 2D mesh on-chip network and a directory protocol. In the baseline system without TMNOC, 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 TMNOC. As the network in TMNOC is enabled to track conflicts between transactions, the router at Node4 might already have the knowledge from past tracking records that the conflicting transactions with higher priority on Node0 and Node8 will eventually nack the request from Node3. Therefore, the router at Node4 nacks the request immediately instead of forwarding it to the directory, as shown in Figure 1(b). Thus, subsequent communication is avoided. A timeout mechanism implemented in the on-chip routers prevents the routers from nacking transactions indefinitely due to stale conflict records. So, forward progress is guaranteed. Our evaluations show that TMNOC reduces network traffic by 20% on average (up to 40%) in a set of high contention benchmarks representative of future TM workloads. Therefore, average network energy consumption is reduced by 24% (up to 39%). Meanwhile, directory busy cycles are reduced by 66%. An implementation using a standard VLSI design flow shows that TMNOC incurs a marginal area overhead of 4.6% when the enabling mechanisms are added to a 4-stage virtual channel router. The contributions of this paper are three-fold:

- We identify a largely unexplored design opportunity in cross-layer optimization of HTMs and NOCs. To the best of our knowledge, our work is the first to address the interaction between HTMs and NOCs.
- We describe TMNOC, a novel approach that exploits co-designing of the HTM and NOC to regulate network traffic generated by transactions and streamline inter-transaction communication.
- We evaluate TMNOC through extensive full system simulations to demonstrate the ability of a HTM and NOC co-design in improving overall energy efficiency and performance.

The rest of the paper is organized as follows. In Section 2, we discuss the background and motivation. The TMNOC design is presented in Section 3. Experiment methodology and results are presented in Section 4. Section 5 summarizes related work, and Section 6 concludes this paper.

2. Background and Motivation

2.1. HTM-NOC Interface

HTMs rely on the on-chip networks to fetch data and detect conflicts. TM-induced network traffic usually 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

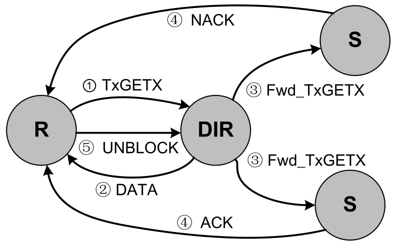


Figure 2. HTM conflict detection. R: requester; DIR: directory; S: sharer.

digits or *flits*. In typical on-chip networks, short packets (*e.g.*, coherence read request and acknowledge response) are single-flit while long packets (*e.g.*, coherence read response and write request) 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. In the HTM-NOC interplay, the on-chip routers are in the unique position of monitoring conflicts between transactions through snooping inter-transaction communication. Routers can learn about conflicts between any transactions by examining the in-transit transactional requests and responses while processor cores only know the conflicts encountered by the local transactions. Thus, routers often have a global view of transaction conflicts, which is exploited in TMNOC to develop more efficient HTM support. Moreover, placing the filter in the on-chip routers instead of directory controllers allows the potentially unnecessary transaction traffic to be filtered out as early as possible to save more energy without disruption to the directory.

2.2. Conflict Detection/Resolution in HTM

A conflict occurs when two or more concurrent transactions access the same data and at least one access is a write [7]. Any coherence protocol capable of detecting accessibility conflicts can also detect transaction conflicts [16]. Directory-based protocols provide scalable solutions to cache coherence due to a unicast nature of communication [19]. The directory can be distributed among all the nodes by statically mapping a cacheline 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 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 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. Once 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 at least one of the responses is a NACK, the requester transaction usually stalls and keeps retrying the nacked request. In what follows, the transaction that sends a NACK is often called a *nacker transaction* or *nacker*. The node on which a transaction is executed is referred to as the transaction's host node.

2.3. False Forwarding and False Blocking

False forwarding occurs when a transaction's coherence request, before being nacked 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 wastes energy since nacked requests do not contribute to the continued execution of transactions. Here, we estimate the energy waste of a nacked coherence request by counting the hops in terms of router traversals needed to accomplish the request. Equation (1) gives the average hop count of a coherence request.

$$H_{CoherenceRequest} = H_{avg} + 2 \times S_{avg} \times H_{avg} + H_{avg} \quad (1)$$

Here, H_{avg} is the average hop of a flit in the network and S_{avg} is the average number of sharers of a memory block. The first term counts the hops of the request to the directory. The second term counts the hops incurred by forwarding and acknowledging. The last term counts the hops of an UNBLOCK message to the directory. In a 4x4 2D mesh network under uniform random traffic, H_{avg} is 3.6 (including the router into which a flit is injected). Assume the requested block is read-shared by 4 nodes. Then, a GETX incurs 36 hops on average. The GETS (request for shared access) needs less hops (14 hops) as the directory only forwards the request to at most one node that owns the data. Each hop, which involves a router and link traversal, consumes a sizable amount of energy. Unfortunately, the energy is wasted in the case of false forwarding.

Besides false forwarding, the nacked requests unnecessarily blocks subsequent requests as coherence requests to the same cache block are serialized at the directory. This problem is *false blocking*, which disrupts normal cache behavior and limits the concurrency of the memory system.

To estimate the extent of false forwarding and false blocking, we track GETS/GETX coherence requests generated by transactions in a representative HTM system (see

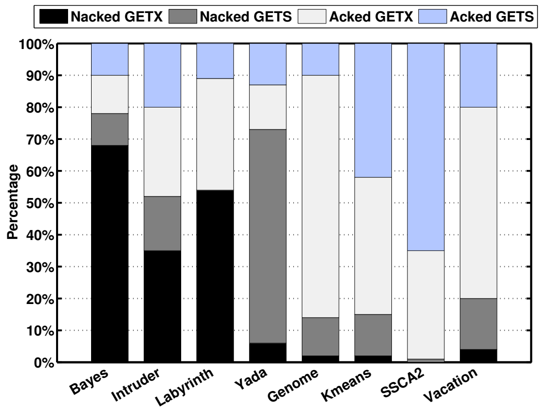


Figure 3. Breakdown of GETX/GETX coherence requests from transactions to the directory.

Section 4.1 for experiment details). Figure 3 presents the breakdown of requests based on the outcome of the requests. Across all eight workloads, nacked 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 and false blocking*.

3. TMNOC: A Co-Design of HTM and NOC

We exploit the co-design of HTM and NOC to mitigate false forwarding and false blocking for improved energy efficiency and performance. In TMNOC, the on-chip routers proactively filter out the redundant coherence requests if they are predicted to be nacked by transactions. We identify three mechanisms to support such functionality. First, the HTM should provide concise yet expressive information on transaction conflicts for the routers to track current conflicts and predict potential conflicts. Second, a cost-effective communication mechanism must be devised to deliver the information to the on-chip routers. Third, the routers must store and use the conflict information to regulate TM traffic. In the subsequent discussion, the three mechanisms are presented respectively. Based on these mechanisms, the traffic regulation policy is described, followed by walk-through examples and further discussion.

3.1. NOC-aware HTM

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. Moreover, concise and expressive information on transaction conflicts must be prepared for the NOC to track conflicts. Other aspects of the HTM design (*e.g.*, version management, read/write set implementation, and overflow handling) are orthogonal and thus complementary to the proposed design. Any HTM designs that piggyback on the coherence protocol for conflict detection can be augmented in a similar way [30, 23, 24]

Conflict Resolution: TMNOC adopts time-based conflict resolution [34]. 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 inter-transaction communication (coherence messages). Besides ensuring

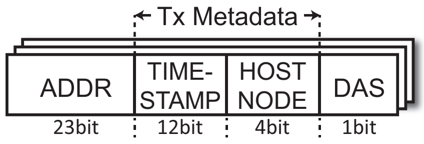


Figure 4. Format of CT-Register.

forward progress and providing good performance [39], the time-based policy provides a global transaction ordering that is straightforward for the on-chip network to identify when detecting potential conflicts.

Conflict Trace Registers: 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:

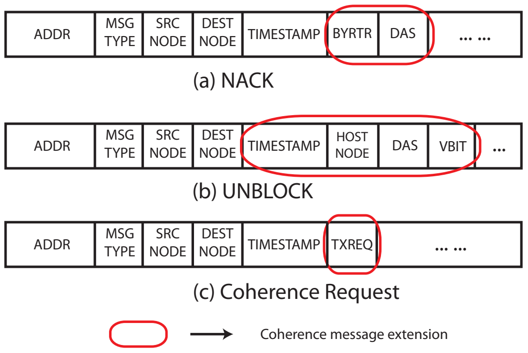
- Address of the memory block in the conflict.
- Metadata (*e.g.*, priority and host node) of the transaction that is given priority in a conflict resolution.
- Data Access Status (DAS) of the memory block, specifying whether the transaction with higher priority holds the block in read-shared or write-exclusive 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 4 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 previous one in the associated CT-Register. The number of CT-Register is bounded by the number of outstanding data requests that miss in local L1. As processors usually support a limited number of outstanding L1 misses (*e.g.*, Intel Itanium2 supports 8 [11]), the area overhead of CT-registers remains low.

3.2. Establishing Communication between HTM and NOC

As HTMs piggyback onto the cache coherence protocol to detect conflicts, coherence messages from transactions are injected into the network. 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.

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. As shown in Figure 5(a), 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



BYRTR	Whether the NACK is initiated from a router
DAS	Whether the address is in the nacker's read or write set
HOST NODE	The node executing the nacker transaction
VBIT	Whether embedded conflict trace is valid
TXREQ	Whether the request is issued within a transaction

Figure 5. Extended coherence protocol messages to support communication between HTM and NOC.

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, that 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 since 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 will have TXREQ set to 1. Figure 5 summarizes the extended protocol messages. Due to the wide on-chip channels, the extended messages can still be encapsulated into short packets. So the cost is minimized. These extensions do not change the protocol behaviors that are originally implemented in the processor.

3.3. Conflict Trace Buffer: In-Network Conflict Tracking

The on-chip routers should be able to store the conflict traces provided by the HTM through the in-transit coherence messages. For this purpose, each router is augmented with a Conflict Trace Buffer (CT-Buffer) (see Figure 6(a)). The CT-Buffer is the key structure to couple the on-chip networks with the HTM. 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). 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

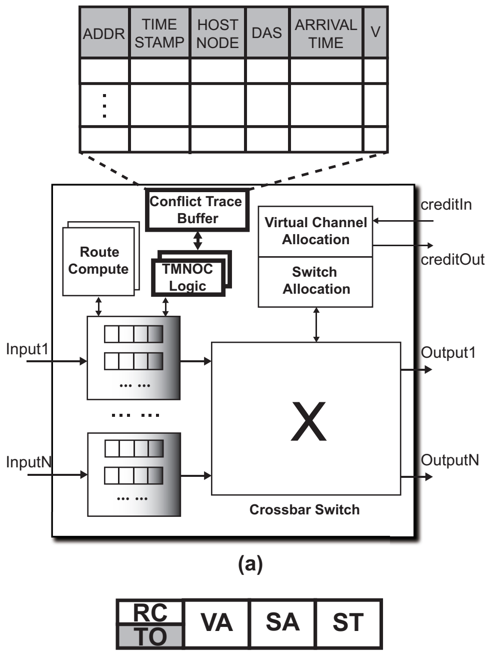


Figure 6. (a) Router microarchitecture (TMNOC-specific structures in bold rectangles). (b) Router pipeline organization (TO: TMNOC Operation).

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 case of rare contention on a read/write port, the overflowed requests are just forwarded normally without being filtered.

3.4. The TMNOC Logic

The TMNOC logic manages the CT-Buffer and performs proactive filtering on in-transit coherence requests. It operates in parallel with route computation to avoid additional delay in the router critical path [1]. The router pipeline is shown in Figure 6(b), assuming a canonical 4-stage pipeline [19]. The route computation (RC), TMNOC operation (TO) and virtual channel allocation (VA) perform computation for the head flits. And, the switch allocation (SA) and switch traversal (ST) stages operate on all the flits. Now, we discuss the functions of the TMNOC logic.

CT-Buffer management: TMNOC logic 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 the new conflict 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

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

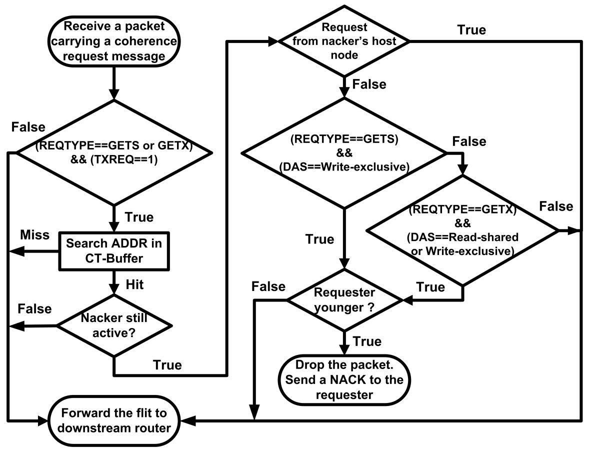

Receive a packet carrying a coherence request message
(REQTYPE==GETS or GETX) && (TXREQ==1)
False
True
Search ADDR in CT-Buffer
Miss
Hit
Nacker still active?
False
True
Request from nacker's host node
True
False
(REQTYPE==GETS) && (DAS==Write-exclusive)
False
True
(REQTYPE==GETX) && (DAS==Read-shared or Write-exclusive)
False
True
Requester younger ?
False
True
Drop the packet. Send a NACK to the requester
Forward the flit to downstream router

Figure 7. Flowchart diagram of transactional request filtering in TMNOC logic. REQTYPE and TXREQ are the fields in the in-transit request message. DAS is from the matching conflict trace.

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 2 entries. As the router only tracks the conflict traces regarding 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 as the requests are always destined to home nodes. A more aggressive scheme will be introduced shortly.

Transactional requests filtering: Upon receiving a packet carrying a coherence request from a transaction (TXREQ=1), 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 evaluate the freshness of the conflict trace. 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 uses the arrival time of the conflict trace to predict. If the conflict trace arrives at the CT-Buffer within a threshold number of cycles, the nacker is predicted to be active. The threshold value can be set to the average transaction length heuristically. TMNOC uses the Transaction Profiling Table in [45], which tracks the moving average of transaction length on a per transaction basis. Our RTL implementation shows that a table capable of tracking 64 transactions adds 0.24% more area to the Sun Rock core [9].

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 the request from the host node of the nacker transaction that is recorded in the matching conflict trace. Figure 7 provides the procedure implemented in TMNOC logic to decide whether to filter out in-transit requests.

TMNOC-aggressive: 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 only 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 since 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. Other than the difference in CT-Buffer management policy, both TMNOC variants follows the same procedure to filter out transactional requests (see Figure 7).

3.5. Operation Examples

Update CT-Register (Figure 8(a)): Transaction A (TxA)@node1 sends GETX to the directory@node2. The request is forwarded to two sharers at node3 and node4. Both nodes respond with NACKs. The NACK from node3 is recorded into the CT-Register of node1 since the nacker transaction on node3 is younger than the nacker on node4.

Update CT-Buffer (Figure 8(a)): After receiving responses from both sharers, node1 sends an UNBLOCK message to the directory. The CT-Register content is attached. The home node router records the conflict trace into its CT-Buffer. The arrival time of the conflict trace is 500.

Home node router nacks a request (Figure 8(b)): TxD@node5 sends GETX to the directory@node2. The router@node2 predicts the GETX to be nacked by TxB@node3. So the GETX is dropped and not forwarded to the directory.A NACK is sent to node5. As the NACK is from a router, node5 does not update its CT-Register.

Router buffers in-transit NACK (Figure 8(c)): The NACK message from node3 to node1 flows through the router@node6, which buffers the conflict trace.

Enroute router nacks a request (Figure 8(d)): TxD@node5 sends GETX to the directory@node2. The

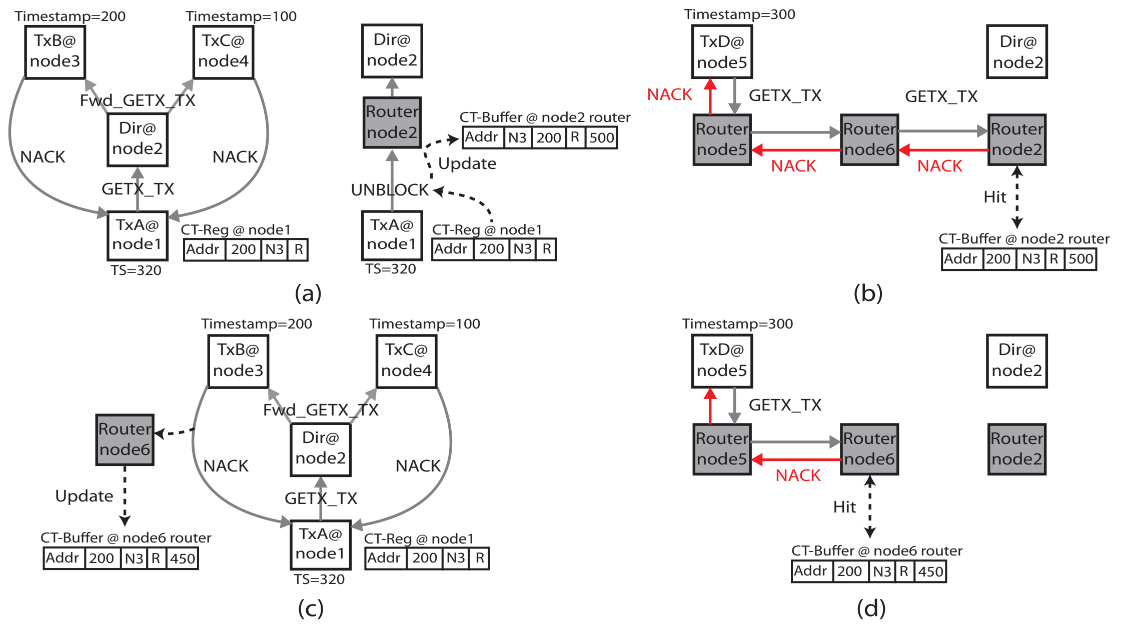


Figure 8. Operation examples. (a) and (b): TMNOC-base. (c) and (d): TMNOC-aggressive. All the requests, responses and coherence states are with regard to the same cache block. Dir: directory.

GETX flows through the router@node6, which predicts the GETX to be nacked by TxB@node3. So the router@node6 drops the GETX instead of forwarding the request. A NACK is sent to node5. As the NACK is initiated from a router, node5 does not update its CT-Register.

3.6. 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 mis-prediction occurs *i)* when a nacker transaction is predicted inactive though it is still active; *ii)* when a nacker transaction is predicted active though it has already finished. In the first case, the router forwards the request as normal. In the second case, the router could nack the request conservatively. However, the router cannot block the request indefinitely, as the nacker is predicted inactive after the corresponding conflict trace has stayed in the router's CT-Buffer for a certain amount of time. So livelock (lack of forward progress) due to mis-prediction 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.

4. Evaluation

4.1. Methodology

We conduct cycle-accurate full system simulation using SIMICS [25] and the GEMS tool set [27]. Garnet [1] is used to model the timing of the on-chip network while the Orion power model [42] is used to estimate the energy consumption of the routers and links in the network. Results are presented for all the eight workloads in the STAMP benchmark suite [29] that is widely used to evaluate HTM designs. Table 1 lists the details of each benchmark.

Table 1. Benchmark input parameters

Benchmark	Input Parameters	Contention
Bayes (BA)	32 var, 1024 records, 2 edge/var	high
Intruder (IN)	2k flow, 10 attack, 4 pkt/flow	
Labyrinth (LA)	32*32*3 maze, 96 paths	
Yada (YA)	1264 elements, min-angle 20	
Genome (GE)	32 var, 1024 records	low
Kmeans (KM)	16K seg. 256 gene. 16 sample	
SSCA2 (SS)	8k nodes, 3 len, 3 para edge	
Vacation (VA)	16K record. 4K req. 60% coverage	

Table 2. System configuration

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

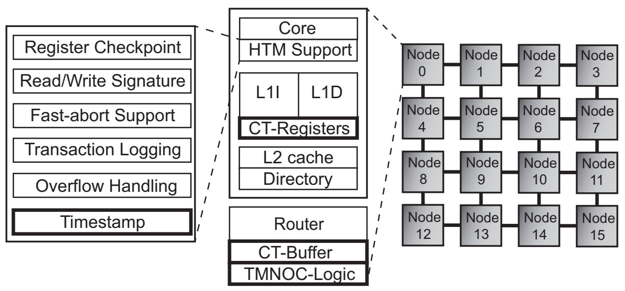


Figure 9. Simulated chip multiprocessor architecture. TMNOC augmentations in bold rectangles.

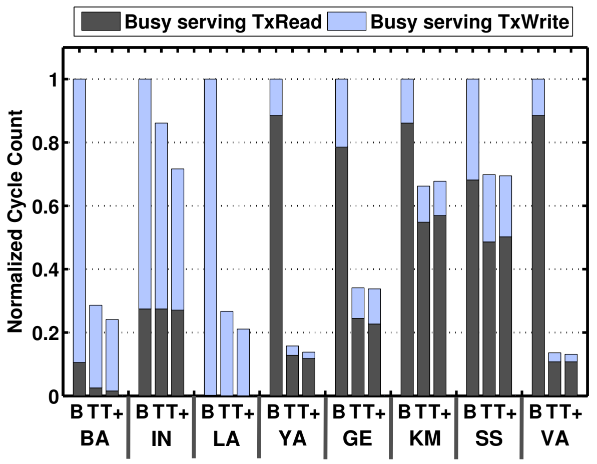


Figure 10. Normalized cycle count when the directory is busy serving transactional requests (B: baseline w/o TMNOC; T: TMNOC-base; T+: TMNOC-aggressive).

The baseline chip multiprocessor architecture for our experiment is depicted in Figure 9. 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 non-uniform cache architecture [21] that uses the directory-based MESI protocol to maintain coherence. The width of coherence control messages is 64-bit. The L2 cacheline tags are augmented with directory entry state. The processor core provides hardware support for log-based HTM. Pre-transaction states are written to a software managed log while speculative states are propagated to the memory hierarchy eagerly. Pre-transaction states are also stored to a dedicated buffer for fast abort recovery. After receiving NACK, transactions back off for a fixed period of 20 cycles before retrying. The performance of the baseline HTM is comparable to contemporary eager HTM designs (*e.g.*, FASTM [23]) that manage both data versions in cache for fast abort and commit. The 2d mesh on-chip network uses dimension-order routing and credit-based virtual channel flow control. Multiple virtual networks are used to avoid protocol-level deadlock. The routers are pipelined into 4 stages. The system configuration is listed in Table 2.

We implement TMNOC-base and TMNOC-aggressive in the simulator. The Garnet router model is augmented with the TMNOC logic and CT-Buffer. Both TMNOC alternatives use a 32-entry CT-Buffer in each on-chip router to track transaction conflicts. Since the TMNOC logic works in parallel with route computation, the router latency is not affected. For energy estimation, we synthesize the TMNOC design. The power dissipation of the SRAM-based CT-Buffer is estimated using CACTI [32]. Based on the obtained results, we modify Orion to carefully account for the energy overhead of TMNOC in 40nm technology and 0.9V on-chip voltage. For the overhead of extending the coherence messages, no extra flit is needed as the flit size is large enough to accommodate the extended fields (a flit size of 128-bit is used in the simulations as most current NOCs have 128-bit or 256-bit channel width).

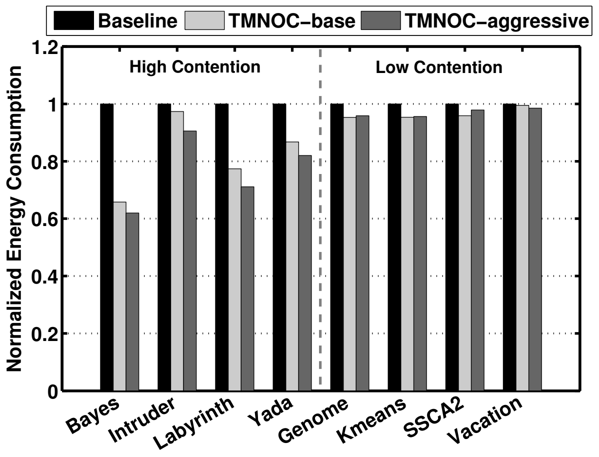


Figure 11. Normalized network energy.

4.2. Reduction in Directory Blocking

Figure 10 shows the impact of TMNOC on the number of cycles the directory is blocked by coherence requests from transactions. The values 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%. The reduction in directory blocking allows more requests to be serviced by the directory instead of waiting or being rejected, thereby increasing the concurrency in the memory system. 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. Since GETX requests usually have a large energy footprint on the network (as discussed in Section 2.3), it is expected that TMNOC will provide significant energy savings in high contention workloads.

4.3. Reduction in Network Energy

One of the primary goals of this work is to improve energy efficiency of the on-chip network in supporting of HTM operations. Figure 11 shows the normalized energy consumption of the network including routers and links. It is observed that 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 38%). 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 the avoidance of forwarding the requests from the directory to other concurrent transactions, whereas TMNOC-aggressive achieves additional savings by saving the hops from the requester to the home node. Since the majority of the traffic and energy waste is due to directory forwarding (multicasting to several nodes) rather than the requests to the home node (unicast between two nodes), TMNOC-base

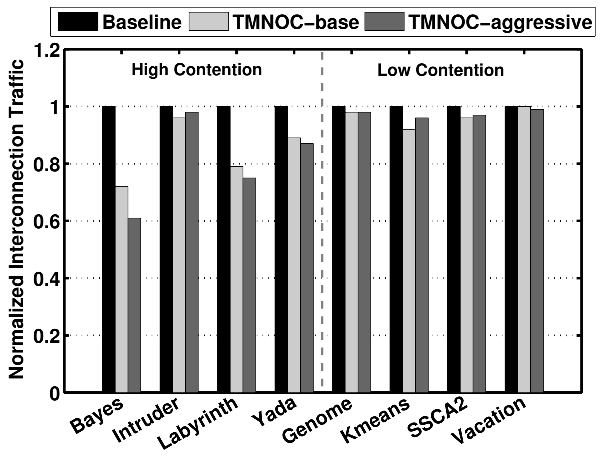


Figure 12. Normalized interconnection traffic.

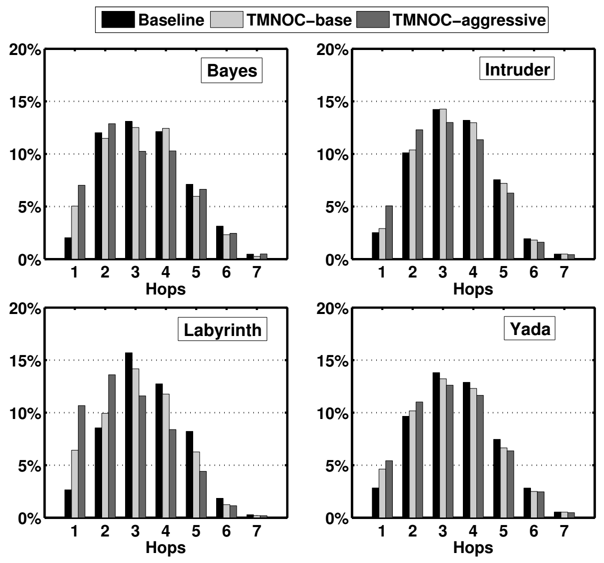


Figure 13. Hop count distribution in high contention workloads (measured in router traversals by flits).

can achieve much of the energy savings with relatively small incremental benefits from the more aggressive scheme. However, 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 in the baseline system (see Figure 3), which offers more energy saving opportunities by 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. Overall, both TMNOC variants achieve the goal of improving energy efficiency in the on-chip network.

4.4. Effect of Network Traffic Regulation

The interconnection traffic has a fundamental impact on the network energy consumption. Figure 12 shows the normalized interconnection traffic measured in router traversals by flits. It is observed that TMNOC-base reduces the traffic in high contention benchmarks by 16% on average (up to 28%) while TMNOC-aggressive reduces the figure by 20% on average (up to 40%). Across all the benchmarks, TMNOC-base and TMNOC-aggressive reduce interconnection traffic by 10% and 11%, respectively. The reduction in interconnection traffic is translated into energy savings.

Figure 13 shows the distribution of flits according to their hops. We only show the results of high contention benchmarks as they show more significant reduction of interconnection traffic and are more representative of future TM workloads with coarse-grain transactions. It is observed that both TMNOC variants reduce the proportion of long-distance flits while increasing the proportion of short-distance flits. Compared with TMNOC-base, TMNOC-aggressive further increases the proportion of 1- and 2-hop flits due to the more aggressive policy to filter out in-transit requests as early as possible. This observation demonstrates the effectiveness of TMNOC in regulating network traffic.

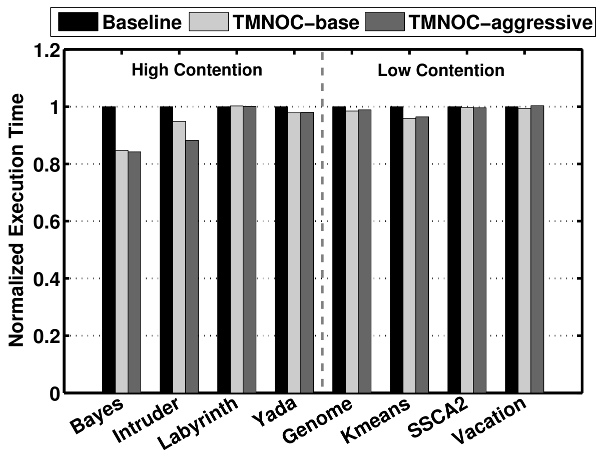


Figure 14. Normalized execution time.

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 nack may degrade overall performance and potentially offset the benefit of increased concurrency in the memory system. Figure 14 shows the normalized execution time. It is observed that TMNOC does not impose a performance penalty on the system in order to increase concurrency and save energy. On the contrary, Bayes and Intruder have more than 15% performance improvement indicating further energy savings in the cores. The performance improvement stems 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 (conflict hot spots) benefit the most from the alleviation of false blocking, as requests to the hot spot are serviced more promptly instead of being blocked unnecessarily. Bayes and Intruder are two such workloads.

Table 3. Percentage of valid NACKs

Benchmark	TMNOC-base	TMNOC-aggressive
Bayes	68%	84%
Intruder	53%	68%
Labyrinth	89%	98%
Yada	88%	93%
Genome	65%	74%
Kmeans	66%	66%
SSCA2	62%	68%
Vacation	93%	95%

Yada shows negligible performance improvement as it does not exhibit the bottleneck of conflict hot spots [45]. In Labyrinth, each transaction reads the entire global maze grid at the beginning and writes to part of the grid at the end. This behavior effectively serializes the transaction execution preventing the workload from taking advantage of the reduced directory contention. Due to the in-order processor cores and well-optimized parallel applications in our experiment, the memory system is not fully stressed. Consequently, the reduction of directory busy cycles is not fully translated into performance improvement. Nevertheless, as future applications are expected to incorporate a large number of transactions, contention on shared data would inevitably become intensive, implying more performance improvement potentials for TMNOC.

Table 3 shows the proportion of valid NACKs among all the NACKs initiated from on-chip routers. A NACK from a router to a requester transaction is *valid* provided that the request from the requester transaction will also get nacked by concurrent transactions if not being nacked by the router. On average, 67% of the NACKs from routers are valid in TMNOC-base while 75% are valid in TMNOC-aggressive. The routers in TMNOC-aggressive obtain conflict traces from the UNBLOCK messages and any in-transit NACKs whereas the routers in TMNOC-base obtain conflict traces from the UNBLOCK messages only. So, the routers in TMNOC-aggressive have better knowledge of transaction conflicts, thereby issuing a larger portion of valid NACKs. Although an invalid NACK could adversely affect the transaction being nacked, it can benefit other transactions that will be in conflict with the nacked transaction. Therefore, the execution time does not necessarily correlate with the proportion of valid NACKs.

Figure 15 shows the breakdown of NACK messages based on their sources. It is observed that more than 30% of NACKs are initiated from routers in both TMNOC variants. As a large portion of the NACKs from routers are valid according to Table 3, false blocking and false forwarding are reduced significantly. Compared with TMNOC-base, TMNOC-aggressive could have a larger percentage of NACKs from routers as it allows any enroute routers to nack the transactional requests. However, the routers at the home nodes have a better chance to intercept the requests since the requests from different nodes are all destined to the home node. So, TMNOC-aggressive does not see a significantly larger portion of NACKs from routers.

4.6. Sensitivity Study

The microarchitecture design trade-off between performance and hardware overhead is mainly affected by the size of the CT-Buffer. A larger CT-Buffer can store conflict traces regarding more cache blocks leading to potentially more accurate filtering of transactional requests. We explore TMNOC's sensitivity to the size of the CT-Buffer in terms of overall execution time. 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 16 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 hot spots. For the TM workloads evaluated, a small CT-Buffer size is sufficient to achieve significant energy savings and effective traffic regulation.

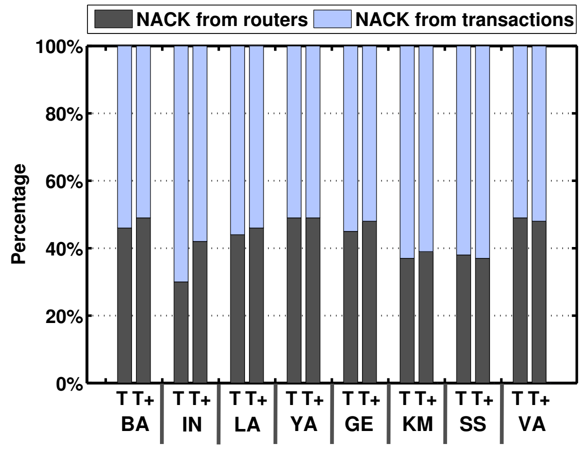


Figure 15. Breakdown of NACK message sources (T: TMNOC-base; T+: TMNOC-aggressive).

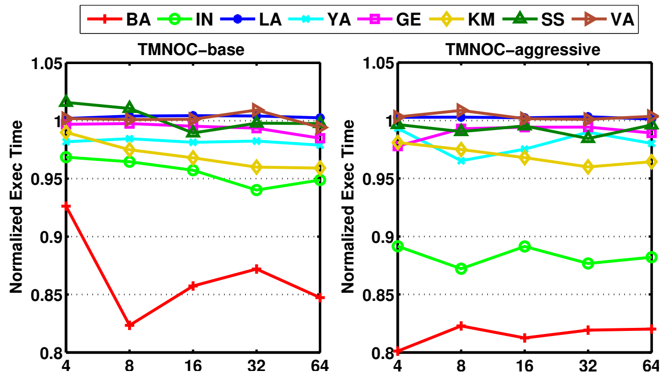


Figure 16. Performance vs. Number of CT-Buffer entries.

4.7. Area 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

Table 4. Result of area overhead estimation

Components	Estimated Area (um^2)
Baseline router	145901
Conflict Trace Buffer	6563
TMNOC Logic	162
Overhead	**4.6%**

buffer is implemented as a 32x64bit 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. The router configurations are identical to those used in the full-system simulation, as shown in Table 2. The design is synthesized using Synopsys Design Compiler targeting TSMC 40nm technology. The clock frequency is set to 1GHz. Table 4 reports the estimated area overhead of TMNOC. TMNOC incurs a reasonable 4.6% area overhead as compared to the baseline virtual channel router. This area overhead is well justified by the energy savings and performance improvement of TMNOC.

5. Related Work

To the best of our knowledge, no previous work addresses the interplay of HTM and on-chip networks. So we discuss the most closely related works from three aspects.

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, due to the substantial network bandwidth and power requirement, various hardware filtering techniques have been proposed. Early works focus on source and destination filtering. In [26], 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 [31, 37] 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 [2] 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, [28] exploits the memory access isolation across VMs in virtualized systems to reduce coherence traffic in a two-level directory protocol. Proximity coherence [3] 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 inter-transaction 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, [20] 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.

Interaction between on-chip networks and applications. Several application-aware NOCs designs have been proposed that use application-level characteristics in optimizing the network topology [17], prioritization [13, 14], and/or routing [22]. However, those designs optimize applications that use a conventional programming models rather than the TM model. Moreover, they do not reduce the number of end-to-end coherence messages whereas the proposed TMNOC design filters redundant in-transit messages, and thus creates more opportunities for saving energy.

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) [44] 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. Unlike ATS, the TMNOC router can improve the prediction accuracy by using not only the local history but also the global conflict information obtained through monitoring on-chip communication. Meanwhile, Proactive Transaction Scheduling (PTS) [4] and Bloom Filter Guided Transaction Scheduling (BFGTS) [5] 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 non-null 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 of the graph structure and the latency of scanning the graph in each conflict detection.

6. Conclusion and Future Work

We explore the largely neglected interaction between HTMs and NOCs. In the process, a potential energy and performance pitfall is identified as false forwarding, which causes a large amount of avoidable TM-induced network traffic. According to our study, 39% of the transactional requests in a wide range of workloads incur false forwarding. To mitigate false forwarding, we propose TMNOC, a novel approach that exploits the co-design of HTMs and NOCs to regulate transactional network traffic. In TMNOC, the on-chip routers track conflicts between transactions by monitoring in-transit TM traffic. Then, routers use the conflict information to filter out transactional requests as early as possible, before the requests incur false forwarding. Evaluation results show that TMNOC is capable of reducing 20% 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 66%. Implemented TMNOC mechanisms result in only a 5% area overhead to a conventional NOC router.

The concept of co-designing HTMs and NOCs presents abundant research opportunities for our future work. We plan to develop a set of high-performance conflict resolution policies based on TMNOCs framework. Moreover, TM-aware prioritization mechanisms in the NOCs merit investigation for accelerated transaction execution.

References

[1] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha. Garnet: A Detailed On-Chip Network Model inside a Full-system Simulator. In *Procs. of Int'l. Symp. on Performance Analysis of Systems and Software*, 2009.

[2] N. Agarwal, L.-S. Peh, and N. K. Jha. In-network Coherence Filtering: Snoopy Coherence without Broadcasts. In *Procs. of the 42nd Int'l. Symp. on Microarchitecture*, 2009.

[3] N. Barrow-Williams, C. Fensch, and S. Moore. Proximity Coherence for Chip Multiprocessors. In *Procs. of the Int'l. Conf. on Parallel Architectures and Compilation Techniques*, 2010.

[4] G. Blake, R. G. Dreslinski, and T. Mudge. Proactive Transaction Scheduling for Contention Management. In *Procs. of the Int'l. Symp. on Microarchitecture*, 2009.

[5] G. Blake, R. G. Dreslinski, and T. Mudge. Bloom Filter Guided Transaction Scheduling. In *Procs. of the Int'l. Symp. on High Performance Computer Architecture*, 2011.

[6] C. Blundell, J. Devietti, E. C. Lewis, and M. M. K. Martin. Making the Fast Case Common and the Uncommon Case Simple in Unbounded Transactional Memory. In *Procs. of the Int'l. Symp. on Computer Architecture*, 2007.

[7] J. Bobba, K. E. Moore, H. Volos, L. Yen, M. D. Hill, M. M. Swift, and D. A. Wood. Performance Pathologies in Hardware Transactional Memory. In *Procs. of the Int'l. Symp. on Computer Architecture*, 2007.

[8] H. Chafi, J. Casper, B. D. Carlstrom, A. McDonald, C. C. Minh, W. Baek, C. Kozyrakis, and K. Olukotun. A scalable, non-blocking approach to transactional memory. In *Procs. of the 13th Int'l. Symp. on High Performance Computer Architecture*, 2007.

[9] S. Chaudhry, R. Cypher, M. Ekman, M. Karlsson, A. Landin, S. Yip, H. Zeffer, and M. Tremblay. Rock: A High-Performance SPARC CMT Processor. *Micro, IEEE*, 29(2), March-April 2009.

[10] Y. Chou, B. Fahs, and S. Abraham. Microarchitecture Optimizations for Exploiting Memory-level Parallelism. In *Procs. of the Int'l. Symp. on Computer Architecture*, 2004.

[11] Intel Corp. Intel Itanium2 Processor Reference Manual.

[12] Intel Corp. Intel Architecture Instruction Set Extensions Programming Reference, February 2012.

[13] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das. Application-aware Prioritization Mechanisms for On-chip Networks. In *Procs. of the Int'l. Symp. on Microarchitecture*, 2009.

[14] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das. Aergia: Exploiting Packet Latency Slack in On-chip Networks. In *Procs. of the Int'l. Symp. on Computer Architecture*, 2010.

[15] R. Haring, M. Ohmacht, T. Fox, M. Gschwind, P. Boyle, N. Chist, C. Kim, D. Satterfield, K. Sugavanam, P. Coteus, P. Heidelberger, M. Blumrich, R. Wisniewski, A. Gara, and G. Chiu. The IBM Blue Gene/Q Compute Chip. *Micro, IEEE*, 2011.

[16] M. Herlihy, J. Eliot, and B. Moss. Transactional Memory: Architectural Support for Lock-free Data Structures. In *Procs. of the Int'l. Symp. on Computer Architecture*, 1993.

[17] W. H. Ho and T. M. Pinkston. A Design Methodology for Efficient Application-Specific On-chip Interconnects. *IEEE Trans. Parallel Distrib. Syst.*, 17(2), Feb 2006.

[18] S. A. R. Jafri, M. Thottethodi, and T. N. Vijaykumar. LiteTM: Reducing Transactional State Overhead. In *Procs. of the Int'l. Symp. on High Performance Computer Architecture*, 2010.

[19] N. E. Jerger and L.-S. Peh. *On-Chip Networks*. Morgan Claypool, 1st edition, 2009.

[20] N. E. Jerger, L.-S. Peh, and M. H. Lipasti. Virtual Tree Coherence: Leveraging Regions and In-network Multicast Trees for Scalable Cache Coherence. In *Procs. of the Int'l. Symp. on Microarchitecture*, 2008.

[21] C. Kim, D. Burger, and S. W. Keckler. An Adaptive, Non-uniform Cche Structure for Wire-delay Dominated On-chip Caches. In *Procs. of the Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, 2002.

[22] M. A. Kinsy, M. H. Cho, T. Wen, E. Suh, M. van Dijk, and S. Devadas. Application-aware Deadlock-free Oblivious Routing. In *Procs. of the Int'l. Symp. on Computer Architecture*, 2009.

[23] M. Lupon, G. Magklis, and A. Gonzalez. FASTM: A Log-based Hardware Transactional Memory with Fast Abort Recovery. In *Procs. of the Int'l. Conf. on Parallel Architectures and Compilation Techniques*, 2009.

[24] M. Lupon, G. Magklis, and A. Gonzalez. A Dynamically Adaptable Hardware Transactional Memory. In *Procs. of the Int'l. Symp. on Microarchitecture*, 2010.

[25] 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, 2002.

[26] M. M. K. Martin, P. J. Harper, D. J. Sorin, M. D. Hill, and D. A. Wood. Using Destination-set Prediction to Improve the Latency/Bandwidth Tradeoff in Shared-memory Multiprocessors. In *Procs. of the Int'l. Symp. on Computer Architecture*, 2003.

[27] 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. Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) toolset. *SIGARCH Comput. Archit. News*, 33, November 2005.

[28] M. R. Marty and M. D. Hill. Virtual Hierarchies to Support Server Consolidation. In *Procs. of the Int'l. Symp. on Computer Architecture*, 2007.

[29] C. C. Minh, J. Chung, C. Kozyrakis, and K. Olukotun. STAMP: Stanford Transactional Applications for Multi-Processing. In *Procs. of Int'l. Symp. on Workload Characterization*, 2008.

[30] K. E. Moore, J. Bobba, M. J. Moravan, M. D. Hill, and D. A. Wood. LogTM: Log-based Transactional Memory. In *Procs. of the Int'l. Symp. on High Performance Computer Architecture*, 2006.

[31] A. Moshovos, G. Memik, A. Choudhary, and B. Falsafi. Jetty: Filtering Snoops for Reduced Energy Consumption in SMP Servers. In *Procs. of the Int'l. Symp. on High-Performance Computer Architecture*, 2001.

[32] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0. In *Procs. of the Int'l. Symp. on Microarchitecture*, 2007.

[33] A. Negi, R. Titos-Gil, M. E. Acacio, J. M. Garcia, and P. Stenström. pi-TM: Pessimistic Invalidation for Scalable Lazy Hardware Transactional Memory. In *Procs. of the Int'l. Symp. on High Performance Computer Architecture*, 2012.

[34] R. Rajwar and J. R. Goodman. Transactional Lock-free Execution of Lock-based Programs. *SIGARCH Comput. Archit. News*, 30(5), 2002.

[35] R. Rajwar, M. Herlihy, and K. Lai. Virtualizing Transactional Memory. In *Procs. of the Int'l Symp. on Computer Architecture*, 2005.

[36] C. J. Rossbach, O. S. Hofmann, D. E. Porter, H. E. Ramadan, B. Aditya, and E. Witchel. TxLinux: Using and Managing Hardware Transactional Memory in An Operating System. *SIGOPS Oper. Syst. Rev.*, 41(6), Oct. 2007.

[37] V. Salapura, M. Blumrich, and A. Gara. Design and Implementation of the Blue Gene/P Snoop Filter. In *Procs. of the Int'l. Symp. on High Performance Computer Architecture*, 2008.

[38] D. Sanchez, L. Yen, M. D. Hill, and K. Sankaralingam. Implementing Signatures for Transactional Memory. In *Procs. of the Int'l. Symp on Microarchitecture*, 2007.

[39] W. N. Scherer III and M. L. Scott. Advanced Contention Management for Dynamic Software Transactional Memory. In *Procs. of the Symp. on Principles of Distributed Computing*, 2005.

[40] A. Shriraman and S. Dwarkadas. Refereeing Conflicts in Hardware Transactional Memory. In *Procs. of the Int'l. Conf. on Supercomputing*, 2009.

[41] A. Shriraman, S. Dwarkadas, and M. L. Scott. Flexible Decoupled Transactional Memory Support. In *Procs. of the Int'l. Symp. on Computer Architecture*, 2008.

[42] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik. ORION: a Power-performance Simulator for Interconnection Networks. In *Procs. of Int'l. Symp. on Microarchitecture*, 2002.

[43] L. Yen, J. Bobba, M. R. Marty, K. E. Moore, H. Volos, M. D. Hill, M. M. Swift, and D. A. Wood. LogTM-SE: Decoupling Hardware Transactional Memory from Caches. In *Procs. of the Int'l Symp. on High Performance Computer Architecture*, 2007.

[44] R. M. Yoo and H.-H. S. Lee. Adaptive Transaction Scheduling for Transactional Memory Systems. In *Procs. of the Symp. on Parallelism in Algorithms and Architectures*, 2008.

[45] L. Zhao, W. Choi, and J. Draper. SELTM: Selective Eager-Lazy Management for Increased Concurrency in Transactional Memory. In *Procs. of the International Parallel and Distributed Processing Symposium*, 2012.