Eliminating On-Chip Traffic Waste: Are We There Yet?

Robert Smolinski, Rakesh Komuravelli, Hyojin Sung, and Sarita V. Adve
University of Illinois at Urbana-Champaign, denovo@cs.illinois.edu

***Abstract*—While many techniques have been shown to be successful at reducing the amount of on-chip network traffic, no studies have shown how close a combined approach would come to eliminating all unnecessary data traffic, nor have any studies provided insight into where the remaining challenges are. This paper systematically analyzes the traffic inefficiencies of a directory-based MESI protocol and a more efficient hardware-software co-designed protocol, DeNovo. We categorize data waste into various categories and explore several simple optimizations extending DeNovo with the aim of eliminating all of the on-chip network traffic waste. With all the proposed optimizations, we are able to completely eliminate (100%) on-chip network traffic waste at L2 for some of the applications (93.5% on average) compared to the previous DeNovo protocol.**

I. Introduction

Improving the energy efficiency of the memory hierarchy is an important part of reaching the future computing energy goals [1], [4], [3]. In this paper, we specifically focus on the implications of data that is never used. Transferring data that is never used in the memory hierarchy incurs unnecessary network traffic which directly contributes to wasted energy. Once such data is transferred to a destination, it is stored in on-chip memory organization units (e.g., caches, scratchpads, etc.) resulting in less available space for useful data and hence indirectly contributes to energy wastage.

We begin our analysis by providing a classification of all data into six different categories. One of the categories is for used words and five are for words that are never used before getting evicted or overwritten (wasted). We limit our study to focus on quantifying unnecessary network traffic caused by each of the waste categories and proposing simple optimizations to nearly eliminate each of these wastes. Towards this end, we chose a directory-based MESI protocol on a standard multicore system as a baseline. There are several well-known traffic overheads that make MESI less than ideal for energy-efficiency. Some of these overheads are a result of using fixed cache line sizes for coherence and data transfer while another source of overhead is traffic required for maintaining protocol correctness (e.g. invalidation and "directory unblock" messages).

Given the inherent overheads with the MESI protocol, we study an alternative hardware-software co-designed protocol named DeNovo [2]. DeNovo leverages software-level information (summaries of memory regions that are read or written) to replace writer-initiated invalidation for cached copies with self-invalidation, which eliminates many of the messages used in MESI to maintain coherence and makes it much more efficient than MESI. The coherence granularity is decoupled from the data transfer granularity. Since DeNovo is built on word-level coherence and has no sharers lists, the hardware is able to respond with subsets of a cache line, and it can also prefetch words from different cache lines into the same response message. This optimization is called as the "flexible communication granularity (Flex)". Using Flex, DeNovo already successfully reduces unnecessary network traffic but focuses only on the traffic destined to L1 cache. It did not analyze this or any other sources of remaining waste.

Our goal in this paper (for more details see [5]) is to eliminate every bit of waste in on-chip traffic. We analyze the effectiveness of the DeNovo protocol and its Flex optimization in eliminating waste at L1. We then quantify the amount of waste at L2 and explore several simple optimizations that extend the DeNovo protocol to eliminate different types of waste at L2. These optimizations not only reduce network traffic and execution time, but can also be implemented with little additional protocol complexity.

II. Waste Characterization

To understand the sources of waste, we classify data resident in a cache (we assume a two level cache hierarchy with private L1 and shared L2 data caches) into six categories.

Used: for the L1 cache, a word that is read by its core; for an L2 cache, a word resident in the cache that is sent in response to an L1 request.[1]
Write Waste: a word that is overwritten before being profiled as *Used*.
Fetch Waste: a word brought into a cache where it is already present as either dirty or was brought in by a previous request.
Invalidate Waste: a word that is invalidated by the coherence protocol before being profiled as *Used*.
Evict Waste: a word that is evicted before being profiled as *Used* or *Write*.
Unevicted: a word that is present in a cache at the end of the simulation and has not been classified.

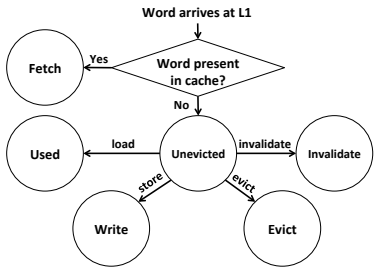


Fig. 1: Finite state machine for waste profiling at the L1 cache.

Figure 1 shows the decision diagram for how data sent to the L1 is categorized. The decision diagram at the L2 is analogous to that of the L1. A network data flit is categorized as above depending on how the data it carries is categorized.

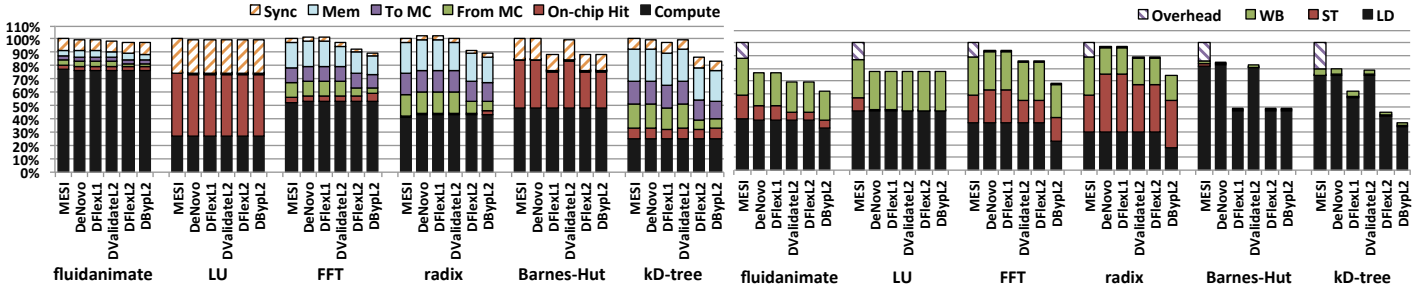
III. Simple Optimizations to Reduce Network Traffic Waste

This section discusses several simple optimizations for the DeNovo line protocol [2] that mitigate the network traffic waste related to the L2 cache.

L2 Write-Validate: The original DeNovo protocol chose to implement a fetch-on-write write policy at the L2 (a write

This work is supported in part by Intel and Microsoft through the Universal Parallel Computing Research Center at Illinois, by Intel through the Illinois/Intel Parallelism Center at Illinois, by the National Science Foundation under grants CCF-1018796 and CCF-1302641, and by the Center for Future Architectures Research, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA. Rakesh Komuravelli was also supported by a Qualcomm Innovation Fellowship.

[1]For the L2, words brought in response to a miss are considered *Used* only if they will be accessed again at the L2.



(a) Execution time (b) Overall network traffic

Fig. 2: Network traffic and execution time of simulated protocols. All bars are normalized to MESI.

miss at the L2 would fetch the entire line from main memory). This introduces *Fetch* waste for the critical word and may also introduce *Write* and *Evict* waste for the remaining words in the line. This optimization changes the L2 write policy to be write-validate reducing the amount of on-chip (and off-chip) traffic.

L2 Dirty-Words-Only Writeback: This optimization uses the L2 per-word dirty bit information so that writebacks to memory from the L2 will only include dirty data, saving the unnecessary traffic required for data that is unchanged.[2]

Memory Controller to L1 Transfer: For L2 misses, the original DeNovo protocol sends data from the memory controller to the L2, and then the L2 forwards the data to the requesting L1. This optimization modifies the protocol so that the memory controller sends the data to both the L1 and the L2 in parallel. This reduces the memory hit latency (and not network waste).

L2 Flex: We extend the L1 Flex optimization to the L2-main memory interface to prevent useless data from being returned from memory, thereby reducing *Evict* waste traffic. We conservatively use a conventional DRAM protocol that only allows for cache line sized reads but the memory controller uses the Flex information on the cache line(s) received from DRAM and only forwards the needed words in the response message to the on-chip cache(s).

L2 Response Bypass: We explore an optimization that prevents lines with poor reuse from being sent to the L2 cache. We found that this optimization was useful for targeting two types of access patterns: (1) the region is read and then overwritten by the same core, or (2) the amount of data in a region exceeds the L2 cache size and is read only once in the current phase of the application. To identify these access patterns, we rely on the programmer or compiler to specify which regions of data should bypass the L2.

The above optimizations do not introduce additional protocol complexity (e.g., transient states) to DeNovo but do require small hardware changes. We did not incorporate any of the optimizations in MESI due to the significant added complexity to the protocol. If we were able to add these optimizations, we expect their improvements would be analogous to those for DeNovo.

IV. Methodology and Results

A. Protocols Studied

MESI: The MESI protocol included with GEMS.
DeNovo: The baseline DeNovo line protocol.
DFlexL1: $DeNovo$ with the Flex optimization at L1.
DValidateL2: $DeNovo$ with support for "L2 Write-Validate" and "Dirty-Words-Only Writebacks."
DFlexL2: $DValidateL2$ with support for "Memory Controller to L1 Transfer" and "L2 Flex."
DBypL2: $DFlexL2$ with "L2 Response Bypass" support.

[2]We assume a DRAM [6] that supports writing subsets of a cache line.

B. Simulation Infrastructure

We used the SIMICS full-system simulator to model each core, the GEMS memory system simulator to model the on-chip memory hierarchy and coherence protocols, Garnet to model the on-chip network, and DRAMSim2 to model DRAM timing. We model a tiled processor with 16 tiles. Each tile has a single core, a 32KB private L1 cache, and a 256 KB slice of the shared L2. The tiles are connected by an on-chip mesh networkwith a link width of 16 bytes. We use all of the benchmarks used in [2] except bodytrack.

C. Results

Figure 2 shows the execution time (part (a)) and the network traffic (part (b)), normalized to $MESI$, for the protocols studied. The individual categorizations in the graphs are explained in [5]. The results show that, relative to the previous state-of-the-art of $DFlexL1$, the optimizations explored in this paper together are effective at reducing network traffic with the same or improved execution times. $DBypL2$ shows an average 18.2% (range of 0% to 39%) reduction in network traffic relative to $DFlexL1$. The execution time reduction ranges from 0% to 14.5% (6.8% on average). These optimizations also eliminated up to 100% of the on-chip network traffic waste at L2 (for two applications), and 93.5% on average compared to $DFlexL1$. The remaining network traffic waste is attributed to unpredictable data access patterns and is hard to eliminate without either increasing the complexity of hardware or having a negative impact on the execution time.

V. Conclusion

This paper focuses on reducing the on-chip network traffic by analyzing sources of wasted data movement and identifying optimizations that reduce such waste in the network. Our evaluation showed that the optimizations explored in this paper can significantly reduce the wasted network traffic for the applications with predictable access patterns. We believe that this study can work as a useful limits analysis and guideline that the potential techniques for waste reduction can rely on.

References

[1] Shekhar Borkar and Andrew A. Chien. The Future of Microprocessors. *Communications of the ACM*, 54(5), 2011.

[2] Byn Choi et al. DeNovo: Rethinking the Memory Hierarchy for Disciplined Parallelism. In *PACT*, 2011.

[3] Stephen Keckler et al. GPUs and the Future of Parallel Computing. *IEEE Micro*, September 2011.

[4] Peter Kogge et al. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems. 2008.

[5] Rob Smolinski et al. Eliminating on-chip traffic waste: Are we there yet? Extended version of ISPASS 2015 paper available at http://denovo.cs.illinois.edu/Pubs/15-ISPASS-extended.pdf.

[6] Doe Hyun Yoon et al. The Dynamic Granularity Memory System. In *ISCA*, 2011.