Fast Data Delivery for Many-Core Processors

Mohammad Bakhshalipour, Pejman Lotfi-Kamran, Member, IEEE, Abbas Mazloumi, Mahmood Naderan-Tahan, Mehdi Modarresi, and Hamid Sarbazi-Azad

Abstract—Server workloads operate on large volumes of data. As a result, processors executing these workloads encounter frequent L1-D misses. In a many-core processor, an L1-D miss causes a request packet to be sent to an LLC slice and a response packet to be sent back to the L1-D, which results in high overhead. While prior work targeted response packets, this work focuses on accelerating the request packets. Unlike aggressive OoO cores, simpler cores used in many-core processors cannot hide the latency of L1-D request packets. We observe that LLC slices that serve L1-D misses are strongly temporally correlated. Taking advantage of this observation, we design a simple and accurate predictor. Upon the occurrence of an L1-D miss, the predictor identifies the LLC slice that will serve the next L1-D miss and a circuit will be set up for the upcoming miss request to accelerate its transmission. When the upcoming miss occurs, the resulting request can use the already established circuit for transmission to the LLC slice. We show that our proposal outperforms data prefetching mechanisms in a many-core processor due to (1) higher prediction accuracy and (2) not wasting valuable off-chip bandwidth, while requiring significantly less overhead. Using full-system simulation, we show that our proposal accelerates serving data misses by 22% and leads to 10% performance improvement over the state-of-the-art network-on-chip.

Index Terms—Memory System, Network-on-Chip, Circuit Switching, Data Prefetching.

1 INTRODUCTION

Server workloads have large data sets that dwarf on-chip caches. Consequently, such workloads experience many L1-D cache misses, which result in frequent stalls and performance degradation. Data prefetching is a widely-used method to eliminate cache misses or reduce their effect.

Unfortunately, data prefetching techniques encounter many difficulties in many-core processors, which significantly limit their effectiveness. The increase in core count drives designs into memory bandwidth wall [1] due to poor pin count scalability. Many-core chips are already able to utilize and even exceed their bandwidth budgets, hitting the bandwidth wall before the power wall [2]. Prefetchers of a core in a many-core processor can induce significant contention with prefetch and demand accesses of other cores, and lead to notable performance degradation [3]. Another drawback of some prefetching techniques is their large area requirements for storing meta-data that may not be available in many-core processors [4].

While server workloads have vast data sets, their secondary working sets are in the range of few megabytes [2], [5]. As the secondary working set of server workloads has a modest size, processors optimized for execution of such workloads can capture the secondary working set in their last-level caches (LLC). When the secondary working set of workloads fit into the LLC, network-on-chip (NoC) is the main contributor to the L1-D miss penalty [6]. For every L1-D miss, a request must be sent to the LLC and a response needs to be sent back to the L1-D (two network traversals). Several recent studies [6], [7], [8], [9] showed the importance of a fast NoC for increasing the performance.

Recent research proposed an elegant solution to accelerate responses from the LLC to the requesting L1-Ds [8]. Lotfi-Kamran et al. proposed using the time interval between the tag and data lookup in the LLC to reserve a circuit for the response packet. The resulting network, named CIMA, uses standard packet switching for request packets and circuit switching for response packets. This way, response packets pass through the network quickly. This optimization makes request traversal in the network the main bottleneck of fast data delivery in server workloads.

This work proposes accelerating transmission of data requests in the network through a simple-yet-effective predictor. We observe that, LLC slices that serve L1-D misses are strongly temporally correlated. Based on this fundamental observation, we design a predictor that upon an L1-D miss, predicts which LLC slice will serve the next L1-D miss. Consequently, we reserve a circuit in the network to accelerate the transmission of the upcoming L1-D miss request.

Through detailed evaluations targeting a set of server workloads, we show that our proposal outperforms data prefetching techniques, even bandwidth-safe data prefetchers, in accelerating execution of many-core processors. While many data prefetchers cause loss of performance due to wasting off-chip bandwidth, we show that our proposal
improves performance by 10% over the baseline with the state-of-the-art network-on-chip and no prefetcher.

In this paper, we make the following contributions:

- We show that spatial and temporal correlation of data accesses manifest themselves into strong temporal correlation among tiles that serve L1-D misses in a many-core processor. Using this observation, we propose a simple, small and effective predictor for accelerating transmission of L1-D miss requests.
- We evaluate the impact of bandwidth-safe data prefetching in a many-core processor and show that they are more effective than standard data prefetchers. Moreover, we show that our proposal is even more effective than bandwidth-safe data prefetchers.
- It is the first work that shows that optimizations in the network-on-chip are more suitable for addressing L1-D misses in a many-core processor as compared to data prefetchers.
- We use a full-system simulation infrastructure to evaluate our proposal in the context of a 64-core server processor on a set of server workloads. The results show that our proposal offers 10% higher performance on top of a server processor backed by the state-of-the-art network-on-chip.

2 BACKGROUND

Current chip multiprocessors (CMPs) accommodate many processing cores, each with data and instruction caches, and a shared LLC slice. Such processors consist of many tiles, wherein a tile includes a core with its private caches, a slice of the shared LLC, and a router. Routers build an on-chip interconnection fabric for tile-to-tile communications.

While NoC delays negatively influence the performance of CMPs, wire delays constitute only a tiny portion of the whole delay, as significant delays are due to routing, arbitration, virtual channel allocation, and reading from and writing to buffers [8].

2.1 Circuit Switching

One approach to reduce the non-wire delays is to let the routers on the path know, in advance, that a packet is coming (i.e., set up a circuit). As routers are aware of the upcoming packets, they reserve the required resources for the packets, which leads to quicker packet transmission.

While circuit switching can potentially decrease NoC delays, CMPs impose difficulties for circuit-switching mechanisms. First, there is no dominant communication pattern in CMPs, as all tiles are accessed almost with the same probability. Second, as the communication is usually short, the overhead of circuit setup does not get amortized.

Prior work [8] aims to optimize LLC-to-L1 communications. Lotfi-Kamran et al. proposed using the time interval between the tag and data lookup in the LLC to set up circuits for the LLC-to-L1 packets. The suggested network, named CIMA, employs conventional packet switching for L1-to-LLC communications and circuit switching, which is faster, for LLC-to-L1 communications. With optimizing LLC-to-L1 communications, L1-to-LLC remains as an obstacle to fast data delivery in NoCs.

2.2 Address Interleaving

Data and instruction addresses are interleaved across the LLC slices. Whenever an L1-D (or L1-I) miss occurs, a request for a piece of data (or an instruction) should be sent to the LLC slice of the tile that holds the data. The destination tile will be determined based on hash of the missed block address (e.g., six least-significant bits in a 64-tile CMP).

3 MOTIVATION

Although there are various types of messages passing through the NoC of a cache-coherent CMP, the message types that have meaningful influence on the performance of workloads are (1) requests for pieces of data or instruction arising from cores, as a result of L1 cache misses, and (2) responses originating from the LLC slices, when requests hit in the LLC. Other kinds of messages either constitute a small portion of on-chip traffic (e.g., coherence messages [7]) or are exposed to a much larger delay than the NoC (e.g., off-chip misses), or nothing is waiting for them (e.g., writebacks), and as a consequence, accelerating them would result in a negligible performance improvement.

We observe that a simple next-line prefetcher can cover roughly 65% to nearly 100% of L1-I misses in our workload suite with almost no off-chip bandwidth or area overhead. For workloads in which next-line prefetcher is insufficient, more advanced instruction prefetchers like SHIFT [4] can eliminate nearly all of instruction misses. In the presence of instruction prefetchers for L1-I caches, accelerating instruction requests, which most of them are prefetch requests, would result in a small performance gain. By employing an instruction prefetcher for L1-I caches, L1-D misses, which cannot be covered by data prefetchers as we show, remain the main bottleneck to be addressed in networks-on-chip. While prior work [8] aims to optimize the latency of response packets, this work intends to accelerate request packets.

Whereas highly-speculative and deeply-pipelined out-of-order cores can, to some extent, tolerate the delay of L1-D misses, simpler cores used in many-core processors cannot hide L1-D misses [10]. Tight physical constraints do not permit having many fat cores on a single chip [2]. So, many-core processors usually feature numerous relatively-simple cores for maximizing throughput. Moreover, low memory-level parallelism (MLP) and instruction-level parallelism (ILP) of server workloads [5] exacerbates the impact of L1-D misses on the system performance.

3.1 Miss Latency Breakdown

Figure 1 shows the breakdown of L1-D miss penalty in a 64-core processor with a mesh network (the details of the processor can be found in Section 5.1). The breakdown shows that NoC is the largest contributor to the L1-D miss penalty. The breakdown for the NoC latency further shows that LLC-to-L1 communications are the largest contributor to the NoC latency. Due to the important role of LLC-to-L1 communications in determining L1-D miss penalty, recent work [8] aims to optimize them.

Figure 2 shows the breakdown of L1-D miss penalty in a 64-core processor with the CIMA network. The breakdown
shows that, unlike a mesh, the L1-to-LLC communications are the largest contributor to the NoC delay. With optimizing LLC-to-L1 communications, L1-to-LLC communications remain as an obstacle to fast data delivery in NoCs.

This work aims to improve the L1-to-LLC communications (i.e., requests) by setting up circuits for them. Unfortunately, we do not know the destination of a request packet until the triggering L1-D miss occurs. At this point, there is no time to set up a circuit for the request packet. To overcome this limitation, we use a predictor for predicting the destination of the upcoming request. Our predictor works based on temporal correlation & locality between the sequence of requests’ destinations, in addition to exploiting the temporal correlation and locality of addresses (which was studied by prior work [12], [13]).

Temporal address correlation refers to a sequence of addresses that favor to be accessed together and in the same order, and temporal stream locality points that recently-accessed address streams likely reappear. Temporal address correlation stems fundamentally from data access patterns. Temporal address correlation can be observed in accesses to data structures such as lists and arrays. When data structures are stable, access patterns recur and miss sequences manifest temporal address correlation [12]. Temporal stream locality occurs since recently accessed data structures are expected to be accessed again. Consequently, address sequences that were recently observed are likely to recur [12].

Moreover, the sequence of requests’ destinations has a higher temporal opportunity for prediction than the sequence of addresses. This is because the sequence of requests’ destinations, in addition to exploiting the temporal correlation of addresses, can benefit from the spatial correlation of addresses. Spatial correlation refers to the appearance that memory accesses occur in repetitive spatial patterns (i.e., the same offsets relative to a base address are accessed). Spatial correlation occurs because applications use various objects with a fixed layout, and a traversal is expected to touch the same elements within each object, as it walks the structure [14].

### 3.2 Temporal Correlation Between Tiles

We observe that the sequence of requests’ destinations (i.e., the sequence of destined LLC slices) has high temporal correlations. To identify the opportunity for a predictor based on the temporal repetition in the sequence of requests’ destinations, we use the Sequitur hierarchical data compression algorithm [11]. Figure 3 shows the results of our Sequitur analysis. The opportunity for a temporal predictor ranges from 77% in Web Zeus to 100% in DSS Qry-2.

The sequence of requests’ destinations inherit temporal correlation from the temporal correlation and locality of addresses (which was studied by prior work [12], [13]). Temporal address correlation refers to a sequence of addresses that favor to be accessed together and in the same order, and temporal stream locality points that recently-accessed address streams likely reappear. Temporal address correlation stems fundamentally from data access patterns. Temporal address correlation can be observed in accesses to data structures such as lists and arrays. When data structures are stable, access patterns recur and miss sequences manifest temporal address correlation [12]. Temporal stream locality occurs since recently accessed data structures are expected to be accessed again. Consequently, address sequences that were recently observed are likely to recur [12].

Moreover, the sequence of requests’ destinations has a higher temporal opportunity for prediction than the sequence of addresses. This is because the sequence of requests’ destinations, in addition to exploiting the temporal correlation of addresses, can benefit from the spatial correlation of addresses. Spatial correlation refers to the appearance that memory accesses occur in repetitive spatial patterns (i.e., the same offsets relative to a base address are accessed). Spatial correlation occurs because applications use various objects with a fixed layout, and a traversal is expected to touch the same elements within each object, as it walks the structure [14]. Figure 4 depicts an example of this phenomenon. Considering 8 KB pages, Page A and Page B embrace 0x0000000-0x001FFF and 0x002000-0x003FFF address ranges, respectively. As shown in the figure, the traversal in Page A recurs in Page B. This causes the sequence of offsets, and consequently the sequence of destination tiles, reappears. In such cases, the sequence of spatially-correlated addresses reshapes as a sequence of temporally-correlated tile sequence. A temporal tile predictor can exploit this behavior for prediction, while a temporal address predictor cannot.

---

1. In this paper, sometimes we use the term temporal correlation to encompass both temporal correlation and temporal locality.
4 The Proposal

Whenever an L1-D miss occurs, our predictor predicts the destination tile of the next L1-D miss. Consequently, we reserve a circuit for the predicted destination. Thus, if the prediction is correct, the next L1-D request will pass through the reserved circuit and will benefit from the delay of circuit switching, which is faster than packet switching.

This mechanism, named Prediction-Based Path Reservation (PBPR), comes within a trade-off with prefetching:

+ Prefetching potentially can increase the off-chip bandwidth (whenever the prefetch candidate is not in the LLC), which is a scarce resource in many-core systems. However, PBPR does not have any notable effect on the off-chip bandwidth.

+ A wrong prefetch can pollute the L1 cache, waste its capacity and bandwidth, and consequently harm the performance. Due to limited capacity and associativity, L1 caches do not tolerate inaccurate prefetches. Nonetheless, a reservation by a circuit does not prevent packet-switched data to use the link bandwidth, as circuit-switched reservations are prioritized when they have packets to forward. A wrong path reservation just wastes a small fraction of NoC’s buffering resources. In a many-core processor, the network is constrained by latency, and not bandwidth or buffering resources, even for workloads with high miss ratio [6]. As the average resource utilization is typically less than 5% in real world applications [15], the negative effect of a wrong path reservation is limited.

+ Storing the meta-data of data prefetchers, especially temporal prefetchers, requires large tables [12], [13]. Nevertheless, recording the correlation between tiles needs significantly less storage.

+ While a correct and timely prefetch can hide the whole latency of an L1-D miss, even a correct and timely reserved path cannot. PBPR can accelerate a significant fraction of miss latency but is unable to hide the entire latency of data misses.

Our predictor relies on temporal correlation between tiles (i.e., the sequence of destined LLC slices). As this predictor predicts just the next L1-D miss tile, streaming schemes may not be effective for our approach. We found that designs similar to classical pair-wise-correlating predictors (e.g., [16]) work much better, in terms of higher accuracy and storage efficiency. These predictors (when have been used as prefetchers), map an address to one (or several) recently-observed succeeding address(es).

One naive approach is to associate each tile to its successor tile. In this manner, upon an L1-D miss, the history table would be searched by current tile and its successor would be the prediction of the predictor. Many prefetchers build upon this simple design. However, we found that this approach considerably suffers from aliasing for tile prediction.

For resolving the aliasing problem, we propose to associate each tile to its \( k \) predecessors. Upon an L1-D miss, we search the history to find the sequence of tiles that served the last \( k \) misses and use the next tile as our prediction. For example, if \( k = 2 \) and the sequence of recent tiles is \( \{ \ldots, T_1, T_2, T_3 \} \), we record that \( T_3 \) follows \( \langle T_1, T_2 \rangle \). We also search the history with \( \langle T_2, T_3 \rangle \) and predict the next tile based on the successor of its last occurrence in the history. Figure 5 represents the sensitivity of coverage and accuracy of the predictor to the number of associated predecessors (i.e., \( k \)), averaged across all workloads. Coverage is the division of covered misses (the data miss requests that we correctly reserve a path for) to the total number of misses. Accuracy is the percentage of correct predictions to all predictions. As we increase \( k \), the accuracy increases because we do not predict unless we find a long sequence in the history. But increasing \( k \) has a different effect on coverage. For \( k \leq 4 \), increasing \( k \) improves the coverage, because of aliasing elimination and distinguishing overlapping streams. For \( k \) larger than 4, increasing \( k \) decreases the coverage because of lack of finding a long sequence in the history. As the figure shows, increasing \( k \) beyond 3 does not yield a notable improvement in the coverage. So for the sake of simplicity, we choose \( k = 3 \) in our experiments.
4.2 Predictor Hardware

Our predictor for determining the LLC slice that will serve the next L1-D cache miss, which is a temporal predictor, is a simple vector indexed by the hash of three previous tile numbers, as shown in Figure 6. The predictor is tagless, so each entry in the vector has few bits (six bits in a 64-core processor) for coding the identifier of the LLC slice that will be accessed next. Upon an L1-D cache miss, we look up the predictor using the three previous tile numbers. The content of the entry determines the next LLC slice. Moreover, we update the content of the entry associated with the three previous tiles with the LLC slice of the current miss address.

Figure 7 shows how the coverage of predictor changes by varying its storage. As shown, beyond 4 kilo-entry there is a minor improvement in the coverage of predictor. We choose 4 kilo-entry (3 KB) table, as it is near-optimal in performance and has low area overhead. Table 1 shows the latency and energy parameters of the history table with three different transistor types. The area overhead of the history table is 0.0039 mm², independent of transistor type. With Low Operative Power transistors, the table responds within one clock cycle in 2 GHZ frequency and consumes less than 0.5 mW power and occupies a negligible area. Nevertheless, each tile in our configured system dissipates in excess of 3.5 W and occupies more than 4 mm². The results indicate the insignificant overhead of the predictor hardware.

**TABLE 1**: Design parameters of the history table.

<table>
<thead>
<tr>
<th>Transistory Type</th>
<th>Latency (ns)</th>
<th>Access Energy (pJ)</th>
<th>Leakage Power (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>High Performance</td>
<td>0.16</td>
<td>1.27</td>
<td>1.65</td>
</tr>
<tr>
<td>Low Operative Power</td>
<td>0.33</td>
<td>0.65</td>
<td>0.34</td>
</tr>
<tr>
<td>Low Standby Power</td>
<td>0.66</td>
<td>1.44</td>
<td>0.83 x 10⁻¹</td>
</tr>
</tbody>
</table>

2. For this experiment, we use CACTI [17] and model a cache which its tag array is the same as our history table.

4.3 Prediction-Based Path Reservation

Knowing the next LLC slice, we plan to build a circuit for the subsequent miss address to accelerate its transmission in the network. Our main contribution is the predictor, which is orthogonal to, and can be used along with, any existing circuit setup mechanism. As such, we only briefly describe the circuit setup mechanism used in our design.

Just like prior work [8], [18], the circuit setup mechanism relies on a narrow dedicated control network. To establish a circuit for a request packet, we send a control packet to the destination tile of the next LLC slice using the control network to reserve a VC in each router of the data network along the path.

In this design, we require the data network to have multiple virtual channels (VCs) for the *request* message class. One VC is reserved for standard packet switching and the rest of them are used for building circuits. Note that a chip multiprocessor has three message classes to guarantee protocol deadlock avoidance: *request*, *response*, and *coherence*. We do not require any changes in the number of VCs for *response* and *coherence* message classes. Moreover, as request packets are 1-flit long, the area overhead of having multiple VCs for the *request* message class is small. Figure 8 shows the sensitivity of performance to the number of request VCs in the data network. Beyond three, there is negligible improvement in system performance. So, we dedicate three VCs to the *request* message class.

On receiving a control packet in a router of the control network, the packet is passed through route computation unit and circuit reservation logic in parallel (i.e., lookahead routing). The circuit reservation logic, which is equivalent of VC allocation logic in a conventional network, assigns a data network VC at the output port specified by the lookahead routing to the upcoming request packet. After reserving the circuit at a node (by assigning the output port and downstream VC to the input VC), the control packet will go through the crossbar and the link in the following cycle to continue reserving the circuit in the downstream node.

Control network is bufferless and drops control packets in case of path conflict. If multiple control packets from different input ports are competing for the same output at the same cycle, they are statically prioritized based on their input port: the winner control packet continues setting up...
Table 2 summarizes the key elements of our methodology, with the following sections detailing the specifics of the evaluated designs, workloads, prefetchers’ configurations, and simulation infrastructure.

5.1 CMP Parameters

Our target is a 64-core processor with 32 MB of last-level cache and six DDR4-2400 memory channels, which is modeled after Intel Xeon Phi™ [19]. Core microarchitecture includes 32 KB L1-I and L1-D caches. The Next-Line prefetcher of L1-I is enabled. Using CACTI [17], we estimated the tag and data lookup delays of a 512 KB LLC slice to be one and four cycle(s), respectively. Cache line size is 64 bytes. Six DDR4-2400 memory channels provide up to 115.2 GB/s of off-chip bandwidth. The request packet length is one flit, while the length of the response packet is five flits.

We study three NoC designs, as follows:

**Mesh:** Our baseline is an 8-by-8 mesh-based tiled processor. A mesh hop is composed of a two-stage router pipeline plus a single-cycle link traversal, resulting in three cycles per hop delay at zero-load. Lookahead route-computation, virtual channel assignment, and speculative switch allocation are performed in the first cycle, while the switch-traversal is done in the subsequent cycle. For covering round-trip credit time, each VC is five flits wide.

**CIMA:** It is implemented on top of the mesh baseline. VCT switching is used for request packets. Control packets are injected into the network right after the end of tag lookups to set up circuits for response packets. A control packet passes each hop in two cycles. For part of the path to the destination on which a circuit is established, the header flit of the response packet passes routers in just one cycle. For the rest of the path, baseline VCT switching will be used. Each VC is five flits deep.

**PBPR:** Prediction Based Path Reservation (PBPR) is implemented on top of baseline. Using predictor, control packets are injected into the network after an L1-D miss to set up circuits for the next miss. While response and coherence message class each has one 5-flit VC, the request message class has a 5-flit VC as in the baseline and two 1-flit VCs for circuit switching.

5.2 Workloads

We include a variety of server workloads from competing vendors, including online transaction processing, decision
5.3 Simulation Infrastructure

We estimate the performance of various methods using Flexus full-system simulator [22]. Flexus provides the detailed timing models for cores, caches, and interconnects, extending the Virtutech Simics functional simulator. Flexus models the SPARC v9 ISA and is able to run unmodified operating systems and applications. Flexus models the network using Booksim network simulator [23]. To model off-chip DRAM performance, we use configured instances of DRAMSim2 [24], parametrized with data borrowed from commercial DDR4 device specifications.

5.4 Prefetchers Configurations

We compare our design with some prefetching techniques. While there are a wide variety of prefetching strategies, we compare PBPR with the following prefetchers:

5.4.1 Variable Length Delta Prefetcher

VLDP [25] is a state-of-the-art prefetcher and was shown to outperform a wide variety of other prefetching mechanisms. VLDP relies on spatial locality and benefits from multiple previous deltas (the difference between two successive miss addresses in a page) for prediction. We equip VLDP with a 16-entry DHB, a 64-entry OPT, and three fully-associative 64-entry DPTs based on the original proposal.

5.4.2 Best-Offset Prefetcher

BO [26] is the winner of the Second Data Prefetching Championship (DPC-2) [27]. BO seeks to determine automatically an offset value (the distance of a prefetch address from the demand address) that yields timely prefetches (i.e., to have the prefetched blocks ready before the actual access). We used the author’s released code. BO is configured with a 128-entry RR and a 15-entry delay queue based on the original proposal.

5.4.3 Global Delta Correlation

G/DC [28] is a data prefetcher that relies on delta correlation between global addresses. We include G/DC because, just like our proposal, it relies on temporal correlation. We provide G/DC with a 512-entry index table and a 512-entry global history buffer (GHB) based on the original proposal.

5.4.4 Stride Prefetcher

We include a stride prefetcher [29] as it is common in commercial processors available today. We implement PC/CS [28] as it overapproximates the original stride prefetcher. We equip PC/CS with a 256-entry index table and a 256-entry GHB based on the original proposal.

5.4.5 Next-Line Prefetcher

Next-line prefetcher is also common in today’s commercial processors. Next-line prefetcher has simple logic and does not impose any storage overhead.

Prefetchers that store their meta-data off-the-chip in addition to erroneous prefetches, incur extra off-chip bandwidth overhead due to fetching and updating the meta-data. Based on published results, these prefetching methods increase the off-chip bandwidth by a factor of $1 \times 4.5 \times$ for server workloads and may not be effective in many-core processors where off-chip bandwidth is a scarce resource. As such, we do not evaluate them.

5.5 Bandwidth-Safe Prefetching

Off-chip bandwidth limitations of many-core processors limit the effectiveness of prefetching techniques as they can potentially increase the off-chip bandwidth and harm the performance. One naive solution in such cases may be dropping the off-chip prefetch requests. In this way, when the prefetch candidate is not in the LLC, the prefetch request from the L1 cache is safely dropped without incurring off-chip bandwidth overhead, which is crucial for many-core processors. For every prefetching technique, we implement one Bandwidth-Safe variant of that prefetcher, named “[Method]-BS” (e.g., VLDP-BS, G/DC-BS, . . . ).

6 Evaluation Results

We run trace-based simulations for profiling and miss coverage studies and detailed cycle-accurate full-system timing simulations for performance experiments.

6.1 Miss Coverage & Overprediction

To demonstrate the effectiveness of the proposed predictor, Figure 9 shows the coverage and overprediction of PBPR as compared to prefetching methods for various workloads. Coverage is the division of covered misses to the total number of misses. Overprediction is the number of incorrect predictions which are normalized against the number of L1-D cache misses in the baseline processor without prefetcher. Incorrect predictions are reservations/prefetches which get evicted before use in PBPR/prefetchers. The coverage of PBPR ranges from 38% in Web Zeus to 100% in DSS Qry-2. The average coverage is 79% across all workloads. As compared to evaluated prefetchers, PBPR increases the coverage of best-performing prefetcher (VLDP) by 1.9× on average across all workloads. Moreover, across all workloads, PBPR covers more misses than the evaluated prefetching methods. As a result of high coverage, the predictor accelerates transmission of many request packets in the network and boosts performance.

The average overprediction of PBPR is 18% across all workloads. The overprediction of PBPR is lower than all prefetching techniques except for the stride prefetcher, which very infrequently issues prefetch requests. As incorrect predictions of PBPR just waste circuit-switched VC buffers (and not cache or off-chip bandwidth), its negative impact is limited.
As most of the applications (i.e., server applications) do not exhibit significant next-line pattern [5], the next-line prefetcher works poorly and pollutes the cache by wrong prefetches. Stride prefetcher rarely prefetches, because most of the applications do not manifest high strided access patterns [5], [12]. Consequently, it has low coverage and low overprediction. Corroborating prior work [12], G/DC is not useful in the context of server workloads. G/DC’s efficiency is limited by its low accuracy and consequently high overprediction rate, wasting off-chip and cache bandwidth and diminishing the performance.

VLDP works better than other prefetching methods but offers significantly less coverage as compared to PBPR. This is because server workloads exhibit less spatial locality at higher levels of memory hierarchy due to the short residency of data [30]. For instance, an 8 KB page would typically linger in a 32 MB last-level cache for tens of milliseconds, unleashing much further time for different data pieces to be accessed within the page in contrast to a 32 KB L1 cache. Consequently, spatial prefetchers are more suited for prefetching into lower levels of the memory hierarchy (e.g., LLC).

Just like VLDP, BO is a spatial prefetching technique, and hence, is not effective for L1 caches. Whereas VLDP detects complex overlapped access patterns by maintaining multiple previously-observed deltas, BO relies on a single delta, and hence, offers lower coverage.

Figure 10 compares the coverage and overprediction of PBPR against the bandwidth-safe prefetching techniques. As compared to normal prefetchers, the overprediction is significantly reduced in bandwidth-safe prefetchers. On average, bandwidth-safe prefetching reduces the overprediction by 25% in the next-line prefetcher to 100% in G/DC. Unfortunately, bandwidth-safe prefetching also reduces the coverage except for the next-line prefetcher in which the coverage is improved by 2%. On average, the coverage is reduced by 9% in VLDP to 61% in the stride prefetcher with bandwidth-safe prefetching. As compared to the best-performing bandwidth-safe prefetcher (VLDP-BS), PBPR offers 2.2× higher coverage on average across all workloads. The overprediction of PBPR is at the same level as that of bandwidth-safe prefetching techniques.

6.2 System Performance

We evaluate system performance given a fixed 128-bit links for all NoC configurations. Figure 11 and Figure 12 show full-system performance, normalized to Mesh (without prefetching), for normal and bandwidth-safe prefetching methods, respectively. The two extra VCs that are dedicated to circuit switching in PBPR can be assigned for packet
Switching in the baseline network. As a point of reference, we include Mesh-7-VCs that represents a network similar to the baseline mesh but with seven 1-flit VCs for request packets (instead of one 5-flit VC).

As shown, normal prefetching methods are unable to considerably boost performance. In many cases, prefetchers degrade the performance due to increasing the off-chip bandwidth, putting excessive pressure on DRAM and increasing shared resource contention.

Bandwidth-safe prefetching techniques can potentially increase the performance by hiding the entire or partial latency of data misses. Meanwhile, evaluated bandwidth-safe prefetching methods fall short of efficiency due to one or several of these obstacles: (1) some of the prefetching methods suffer from cache pollution problem (e.g., next-line prefetcher). (2) The meta-data storage of some of the evaluated prefetchers is too small to capture the address-correlation history of server workloads. Capturing the history of address-correlation methods requires multi-megabyte storage for meta-data which cannot be accommodated on-the-chip and should be located off-the-chip. Unfortunately, locating the meta-data off-the-chip dramatically increases the off-chip bandwidth, which makes such schemes ineffective for many-core processors. (3) The assumptions and consequently the mechanism of some prefetchers are not suitable for server workloads (e.g., strided access patterns). As a result, these methods are not effective in the context of server workloads.

The Mesh-7-VCs network offers only a negligible performance improvement over the baseline mesh (less than 1% on average). In general, a VC is a throughput-oriented knob in packet-switched networks that helps increasing the performance by reducing the head-of-line (HOL) blocking. As network traffic of server workloads is moderate, and consequently, the HOL blocking rate is minimal, increasing the number of VCs does not significantly improve the performance.

While evaluated prefetchers are unable to boost the performance significantly, PBPR improves the performance across all workloads. The performance improvement of PBPR ranges from 2% in Media Streaming to 19% in DSS Qry-1. The geometric mean performance improvement of PBPR is 10%. The second best method is VLDP-BS with geometric mean performance improvement of 6%.

### 6.3 Fast Data Delivery

As PBPR and CIMA accelerate different types of packets in the network, they can be used orthogonally. Figure 13 shows the performance improvement of Fast Data Delivery.
CIMA improves the performance by 4% on average. The main deficiency of CIMA is the limited run ahead of its control packets. CIMA leverages the time interval between the tag and data lookup in the LLC to reserve a circuit for the response packets. As there is not much time at this point, CIMA can reserve resources at a few routers, and in most cases, is unable to reserve the entire path for the response packets. Meanwhile, PBPR benefits from enough run ahead of its control packets. PBPR can benefit from the whole time interval between two consecutive L1-D misses to reserve a circuit for the upcoming request. Figure 14 shows the cumulative distribution of clock cycles between consecutive L1-D misses. As depicted, most of the time there is enough slack for reserving a circuit for the next data request. As a consequence, PBPR often reserves the entire path for request packets and results in higher performance improvement.

6.4 Timeliness of Circuit Reservations

While PBPR usually has enough time to reserve the entire path, as implied by Figure 14, request packets do not always benefit from circuit switching for the whole path down to the destination. This is mainly due to conflicts when there is no free VC in a hop, in which a victim is inevitably chosen and the corresponding circuit is torn down.

Figure 15 evaluates the timeliness of circuit reservations for requests with correct predictions. One bar shows the fraction of requests that benefit from circuit switching in their entire path. The other bar shows the fraction of hops in which circuit switching is used for request transmission. The number on each bar shows percentage of miss-penalty reduction of PBPR. While circuit teardown is possible, Figure 15 shows that requests pass most of the hops with fast circuit switching, thanks to the two dedicated VCs that minimize the circuit teardowns. On average, 44% of requests use already-established circuits for their entire paths and more than 89% of the hops are traversed via fast circuit switching.

As a result of fast transmission and high coverage, PBPR greatly accelerates data misses. The miss-penalty reduction of PBPR ranges from 9% in Zeus to 25% in Qry-16. The average miss-penalty reduction is 22% across all workloads.

6.5 Out-of-Order Execution

PBPR is not tightly coupled to in-order cores. It can be used with out-of-order (OoO) cores as well. When using OoO cores, for preserving the order between tiles, the history table is updated based on the retirement order of instructions. To facilitate logging, reorder buffer (ROB) entries are marked when the data of the load instruction misses in the L1-D cache.

Figure 16 compares the performance improvement of PBPR on in-order and OoO cores. For OoO execution, we use 2-wide Decode/Rename/Retire cores, each equipped with 72-entry ROB and 12-entry LSQ (similar to Intel’s Knights Landing [31]). As depicted in the results, the effectiveness of PBPR varies for different types of cores.
depending on the workload. Whenever an application exhibits high MLP (e.g., blackscholes), OoO cores can hide the latency of L1-D misses to a certain extent. In these workloads, the performance improvement of PBPR on OoO cores lowers because L1-D misses become less important. When an application has low MLP (e.g., DSS Qry-2), OoO cores cannot effectively hide the latency of L1-D misses. In such situations, the performance improvement of PBPR on OoO cores becomes higher, as there is a larger gap between the performance of cores and the memory system.

### 6.6 Three-Level Cache Hierarchy

While the evaluated processors have a two-level cache hierarchy, there are many processors with three levels of on-chip caches. In such processors, PBPR trains and predicts on L2 data misses. Whenever a data miss occurs in the private L2 cache, PBPR predicts the destination of the next data miss and then updates its history table. For evaluating the proposed method in a three-level cache hierarchy, we add 256 KB 8-way L2 caches between L1-I/Ds and the LLC in our configured system.

Figure 17 shows the opportunity and miss coverage of PBPR on L2 caches, as compared to L1 caches. In the three-level hierarchy, the opportunity and coverage of the predictor fall by 17% and 20%, respectively. The main reason is that as we get away from the core, the temporal correlation of accesses decreases [30], [32]. Memory requests that their data is available at higher-level caches are filtered, resulting in less temporal predictability at lower-level caches. However, while the coverage of the predictor is reduced, it is still significant. With a three-level hierarchy, PBPR, on average, covers 60% of data misses (up to 80%) and manifests the potential for providing a substantial performance improvement.

### 6.7 Sensitivity to Last-Level Cache Access Latency

We considered Low Operative Power for the transistor type of the LLC, as advised by prior work [33]. In this section, we analyze the performance improvement of PBPR, when other transistor types are used for building LLC slices. Among transistor types, High Performance (HP) transistors offer the lowest access latency, but at the cost of high power consumption. Low Standby Power (LSTP) transistors, on the other hand, trade performance for power reduction, and enable building low-power caches, but with higher access latency. Low Operative Power (LOP) designs also have a performance and power characteristics that lie in between the HP and LSTP designs.

Using CACTI [17], we estimate the access latency of a 512 KB LLC slice to be 3/5/8 cycles with HP/LOP/LSTP transistors. Figure 18 compares the performance improvement that PBPR provides with various LLC designs. The maximum performance improvement is achieved for HP design and the minimum for LSTP. With increasing LLC access latency, the performance improvement of PBPR slightly decreases. The main reason is that, by increasing the LLC access latency, the contribution of NoC to the total miss penalty decreases (cf. Figures 1 and 2). However, the reduction is insignificant because NoC is still the major contributor to the miss latency. The performance improvement of PBPR with HP and LSTP transistors is within 1% of the performance improvement with LOP transistors.

3. The configured processor has 32 MB LLC which is distributed to sixty-four 512 KB slices.
6.8 Concentrated Mesh
Concentrated Mesh (CMesh) [34] was proposed for reducing zero-load latency, area and wiring costs of NoCs. In CMesh, several tiles share the same router for communications, which reduces the total number of routers. With the reduction of routers, the number of bits that should be used for determining the destination router also decreases. For example, if every two tiles in a 64-tile CMP share the same router, the number of required bits for identifying the target router reduces from 6 to 5. As such, using PBPR in the context of CMesh requires predicting fewer bits.

For evaluating PBPR in concentrated topologies, we consider two CMesh cases and re-measure the prediction opportunity. One case devotes a router to two tiles (C2Mesh), and the other one shares each router among four tiles (C4Mesh). Figure 19 shows the prediction opportunity in concentrated topologies, as compared to the baseline Mesh. Repeatedly, Mesh/C2Mesh/C4Mesh design has 64/32/16 routers, and 6/5/4 bits are used for determining the destination router. The figure shows that as the topology becomes more concentrated, the prediction opportunity increases. The main reason for this phenomenon is the increased spatial locality when fewer bits are predicted (cf. Section 3.2).

6.9 Power Analysis
Corroborating prior work [6], [7], [8], our investigation confirms that NoC is not a considerable consumer of power at the chip level4. For all organizations and all workloads, the NoC power is below 2 W. In contrast, the chip consumes in excess of 200 W [19]. Moreover, as most of the NoC power is dissipated in the links [7], there is no much difference in the power consumption of Mesh, CIMA, and PBPR networks.

7 RELATED WORK
Data prefetching is an active research area. Thread-based prefetching methods (e.g., [36]) exploit idle thread contexts or dedicated pre-execution hardware to run threads that prefetch for the main program thread. However, the extra resources the prefetcher threads need may not be available when the processor is fully utilized.

Software-based prefetching methods (e.g., [3]) use compiler or programmer hints to issue prefetch operations. However, the complicated access patterns and changes in the data set of server applications make prefetching difficult for static and profile-based approaches. Moreover, these techniques need application modifications or recompilation.

Alternatively, many pieces of prior work tried to compensate part of the memory access latency by reducing the on-chip network latency, since a noticeable fraction of LLC access latency is due to on-chip communications. Reducing the communication latency is mainly achieved by reducing either the average hop count or per-hop latency of the on-chip network.

Adopting low-diameter topologies is the most effective approach to reduce packet hop count. While inherently low-diameter topologies, such as hypercube, are not suitable for on-chip circuit-level implementation, prior work has shown that several high-radix topologies [37] can provide shorter hop count with easier circuit-level implementation.

Circuit-switching is often the most basic way to reduce per-hop latency. This latency reduction is achieved by forwarding data on pre-allocated paths, effectively eliminating the need for buffering, arbitration, flow control and virtual channel allocation. However, the long setup time of conventional circuit-switching is often prohibitive. To eliminate the circuit setup time from the packet’s end-to-end delay, several pieces of prior work proactively reserve a path well before the actual packet transfer [8], [9], [18], [38]. In flit-reservation flow control method [38], a control flit pre-allocates buffers and channel bandwidth for one or multiple flits of a data packet. This method can effectively reduce packet latency, but at the cost of using a faster control network (with higher frequency) for the control flits to keep pace with their corresponding data packets.

CIMA is a method that uses the period between the end of tag and data lookup in the LLC to proactively allocate resources for data packets [8]. PRA, in addition to cache-induced resource pre-allocation of CIMA, leverages the packet in-network blocking time for resource pre-allocation in a single-cycle multiple-hop traversal NoC [9]. Proactive Circuit Allocation (PCA) [18] is another proactive circuit setup method that tries to reduce memory access time. In this method, request packets, while traveling down to their destination, reserve a circuit in the reverse direction for their corresponding data packets. These methods only accelerate response packet transmissions, and hence, are orthogonal to our proposal that targets request packet latency reduction.

In Runahead NoC, a bufferless network is stacked on top of the main packet-switched NoC [15]. Runahead NoC carries a single flit of every critical packet, which are already traversing the main network, using single-cycle routers that drop packets when contention occurs. To avoid multiple packet delivery, the packets received from Runahead NoC are registered at destination nodes to prevent ejecting their copies transmitted over the main NoC. Bufferless NoCs have been shown to be useful only under light traffic loads. The performance of a bufferless NoC starts to degrade in moderate traffic loads [39], which is the case for many server applications. They are also not scalable because the proba-
bility that a packet can survive up to its destination node reduces when NoC size grows. In our bufferless control network, however, dropping a packet in an intermediate router does not nullify all the effort, as a partial circuit up to that point is reserved, and the upcoming request packet can use it for faster transmission in a part of the path to the destination.

Designing low-latency routers is another approach to reduce packet latency. Bufferless routers [39] and express virtual channels [40] eliminate different pipeline stages of a router to reduce zero-load latency. These low-latency NoC designs can be integrated into the proposed prediction method to further reduce latency.

8 Conclusion
L1-D misses often stall the processor for the data to arrive at the L1-D, so workloads lose performance due to inefficient data delivery. As prior work accelerated the data delivery from the LLC to the L1-D, this work focused on speeding up the request transmission from the L1-D to the LLC. We observed that LLC slices that are serving L1-D misses are strongly temporally correlated. Based on this observation, we proposed a simple predictor for identifying the LLC slice that will serve the next miss. Knowing the future, a circuit was set up for fast transmission of the request packet of the upcoming L1-D miss. We showed that our proposal improves system performance by 10% over the state-of-the-art NoC.

Acknowledgments
We thank the anonymous reviewers for their valuable comments. The authors would like to thank members of the HPC center of IPM for managing the cluster that is used to conduct the experiments. This work is supported in part by a grant from Iran National Science Foundation (INSE) and a grant from Research and Technology Deputy of the Sharif University of Technology.

References
9 Biography

Mohammad Bakhshalipour received his M.Sc. and B.Sc. in computer engineering from SUT in 2017 and 2015, respectively. Currently, he is a graduate research assistant in computer science and a member of Iran’s national compute-grid project at Institute for Research in Fundamental Sciences (IPM). His research interest is computer architecture with an emphasis on memory systems.

Pejman Lotfi-Kamran is an associate professor of computer science and the manager of Iran’s national compute-grid project at Institute for Research in Fundamental Sciences (IPM). His research interests include computer architecture, computer systems, approximate computing, and cloud computing. His recent work on scale-out server processor design lays the foundation for Cavium ThunderX. Lotfi-Kamran has a Ph.D. in computer science from the Ecole Polytechnique Federale de Lausanne (EPFL). He received his MS and BS in computer engineering from the University of Tehran. He is a member of IEEE and the ACM.

Abbas Mazloumi received the B.S. degree in Computer Engineering from University of Mazandaran, Babol, Iran, in 2009 and M.S. degree in Computer Architecture from University of Tehran, Tehran, Iran, in 2014. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, University of California, Riverside, CA, USA. His research interests include Computer Architecture in general, Networks on Chip, GPU Micro-Architecture, High-Performance Computing and Graph Processing Algorithms.

Farid Samandi received his B.Sc. degree in Computer Engineering from Amirkabir University of Technology, Tehran, Iran, in 2015 and his M.Sc. degree in Computer Engineering from Sharif University of Technology, Tehran, Iran, in 2017. Currently, he is working as a research assistant in Institute for Research in Fundamental Sciences (IPM). His research interests expand in the area of computer architecture including memory systems, multicores architectures, and processor microarchitecture.

Mahmood Naderan received the B.Sc. degree from Amirkabir University of Technology, Tehran, Iran, in computer engineering in 2005, the M.Sc. and Ph.D. degrees in the computer architecture in 2007 and 2016, respectively from Sharif University of Technology, Tehran, Iran. He is currently an assistant professor in computer engineering faculty of Shahid Chamran University of Ahwaz, Iran. He is also a non-resident researcher at School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran. His research interests are in the areas of multicores and accelerated architectures, cache and memory systems, storage systems, approximation and stochastic computing.

Mehdi Modarresi received the B.Sc. degree in computer engineering from Amirkabir University of Technology, Tehran, Iran, in 2003, and the M.Sc. and Ph.D. degrees in computer engineering from Sharif University of Technology, Tehran, Iran, in 2005 and 2010, respectively. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of Tehran, Tehran, Iran. His research focuses on different aspects of high-performance computer architecture and parallel processing with a particular emphasis on networks-on-chip and neuromorphic architectures.

Hamid Sarbazi-Azad received his BSc in electrical and computer engineering from Shahid Beheshti University, Tehran, Iran, in 1992, his MSc in computer engineering from Sharif University of Technology, Tehran, Iran, in 1994, and his Ph.D. in computing science from the University of Glasgow, Glasgow, UK, in 2002. He is currently a professor of computer engineering at Sharif University of Technology and heads the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran. His research interests include high-performance computer/memory architectures, NoCs and SoCs, parallel and distributed systems, performance modeling/evaluation, and storage systems, on which he has published over 300 refereed conference and journal papers. He received Khwarizmi International Award in 2006, TWAS Young Scientist Award in engineering sciences in 2007, and Sharif University Distinguished Researcher awards in years 2004, 2007, 2008, 2010 and 2013. He is now an associate editor of ACM Computing Surveys, Elsevier Computers and Electrical Engineering, and CSI Journal on Computer Science and Engineering.