DISCO: A Low Overhead In-Network Data Compressor for Energy-Efficient Chip Multi-Processors

Ying Wang, Yinhe Han, Jun Zhou, Huawei Li, and Xiaowei Li
State Key Laboratory of Computer Architecture
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, P.R. China
{wangying2009, yinhes, zhoujun, lihuawei, lxw}@ict.ac.cn

ABSTRACT

Data compression has been proposed to increase the utility of on-chip memory space or Network-on-Chip (NoC) bandwidth in energy-efficient processors. However, such techniques usually add additional compression and decompression latency to the critical path of memory access, which is one of the major factors limiting their application to processors. In contrast to prior work that deals with either cache compression or network compression separately, this study proposes a unified on-chip DIStributed data COmpressor, DISCO, to enable near-zero latency cache/NoC compression for chip multi-processors (CMPs) adopting Non-Uniform Cache Access (NUCA). DISCO integrates data compressors into NoC routers and seeks opportunity to overlap the de/compression latency with the NoC queuing delay through a coordinated NoC scheduling and cache compression mechanism. With the support of DISCO that unifies the solutions of on-chip data compression, it is shown in evaluation that DISCO significantly boosts the efficiency of on-chip data caching and data moving.

1. INTRODUCTION

As multicore scales, the growing number of processor cores are putting more and more pressure on on-chip storage. A large on-chip Last Level Cache (LLC) is essential to bridge the gap between computational strength and off-chip memory bandwidth for current big data or scale-out workloads [1]. However, a large cache footprint has substantial impacts on die area, power consumption, chip yield and opportunity cost to integrate more computation resources. Recently, researchers are looking forward to cache compression to increase the cache space utilization. By exploring the redundancy inside the data value, a variety of data compression techniques have been proposed in these days to reduce cache misses or save memory bandwidth in different scenarios.

There are frequent pattern based compression techniques and delta-based schemes as it categorized in prior work [2] [3] [4] [5]. Although in different implementations, these cache compressors are subject to several critical weaknesses that hinder them from being applied to commodity multi-processors. The first factor is the compression-and-decompression latency caused by the compressor. Sometimes the compression/decompression latency lies in the critical path of cache access and it directly degrades workload performance. Especially in Chip Multi-Processors (CMPs) that connect distributed cache banks to cores via Network-on-Chip (NoC), the additional compression-and-decompression latency and NoC latency will elongate the critical cache access path. The problem is so critical to the latency-sensitive on-chip memory subsystem, that it becomes the greatest factor that limits the application of cache compression techniques [5]. The other factor is the hardware cost. For typical CMPs with Non-Uniform Cache Access (NUCA) as illustrated in Fig. 1, the cost of within-cache compressor rises linearly when cache bank count inevitably increases, especially for compressors with pattern table or dictionary storage [3] [4].

In this work, we propose a DIStributed in-network data COmpressor (DISCO) as a substitute to avoid the weaknesses of typical cache compressor. Instead of attaching compressor to cache or cores, DISCO merges the data compressor into the routers of NoC, and allows data compression to benefit all the data caching and moving activities that occur on chip.

The priories and opportunities of in-network compressor

Conventionally, cache compressors focus on compression ratio, and rarely discuss its integration strategy into the CMPs. We found that merging data compressor into NoC can conspicuously mitigate the overhead of cache compression and also achieve multiple rewarding advantages, making it more likely to be adopted in realistic processors.

First, in-network compressor can substantially reduce the access latency caused by traditional cache compression or even completely remove its performance penalty imposed on cache access. For example in Fig. 1, the timing path of a cache access, which originates from the write-back request from the core or read response to the core, includes three major components: routing latency from cache tile to core tile, decompression-or-compression latency, and cache bank access latency. In DISCO, the de/compression can be conducted during the network routing stage. In other words, DISCO is able to overlap the two important components of cache access latency by re-architecting the router design. In DISCO router, the time wasted on network queuing or congestion-caused stall, are used to compress or decompress the cache blocks in the form of NoC packets, so that the de/compression time overhead is completely hidden. Speed is a key aspect of compression scheme. Some cache compressors offers high compression rate at expense of performance overhead, while some other compression schemes are designed to offer fast compression at a cost of compression ratio [3] [5]. When we could overlap de/compression latency with network queuing delay, the performance of cache compression will become ignorable in some cases so that compression algorithms with high compression ratio will be more practical in energy-efficient CMPs.

Second, according to our observation, NoC is the best location to conduct data compression by unifying the previously proposed independent data compressors for network traffics and the memory hierarchy. For most CMPs, Miss Status Handling Register (MSHR) in L1 cache of the processor core is responsible for receiving the depacktized cache block ejected out of the local NoC router. The depacktized block has to be decompressed before it enters into a MSHR entry and then reaches the core. Conventional *within-cache* data compressor can only compress and decompress blocks directly in LLC banks, so the cache blocks sent to cores must be decompressed at first, then packetized and injected to NoC in an uncompressed form. In this case, uncompressed blocks will contribute to more data moving power and NoC traffics. Problems also exist in *within-core (L1 cache)* data compressor. Since the main memory sometimes cannot hold blocks compressed for cache due to the problem of misalignment and mapping

DAC '16, June 05-09, 2016, Austin, TX, USA
 ISBN 978-1-4503-4236-0/16/06 $15.00
DOI: http://dx.doi.org/10.1145/2897937.2898007

issue in DRAM devices. If the data compressor is put in cores, the blocks pre-fetched from main memory to last level cache have to reach the core for compression at first, and then goes into LLC in a compressed form, which induces unnecessary data moving. Otherwise, the uncompressed block cannot fit in the LLC bank. The problem reoccurs when a block is evicted from LLC to the main memory. In contrast, NoC router is the gateway to all elements of the on-chip processor including cores, LLC banks and memory controllers. In-network compression guarantees that all these components benefit from compression no matter where the data comes from or goes to. Therefore, morphing cache compressor with NoC router leads to multi-fold benefits including latency and traffic load mitigation for energy-efficient CMPs.

In conclusion, we propose a unified on-chip data compression framework that can goes in parallel with all of the compression proposals that target on high compression ratio or performance. Prior art has successfully leveraged data compression techniques to compress NoC traffics in Network Interface (NI) for energy-efficient communication [9] [8] [7]. However they are independent of cache compression and disregard the opportunities of overhead elimination in combining the two approaches. Specifically, we makes the following contributions in this work:

- We investigate the problem where to conduct data compression, design an in-network data compressor and make it possible to reuse the network queuing time to de/compress cache blocks. It could potentially eliminate the performance penalty of cache compression and so remove the major obstacle that stops the cache compressors from being adopted in practice.
- Based on the observation, we propose a coordinated NoC packet-scheduling policy and de/compression strategy with DISCO compressor, so that cache compression could be conducted in time to improve data moving performance and power-efficiency.
- We examine the potential benefits brought by routing strategies to provide none-blocking selective de/compression for different flow-control. Evident improvement on performance and energy-efficiency is witnessed in the constructed full system simulation environment.

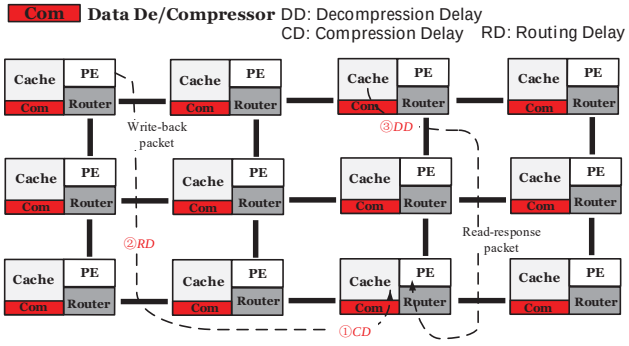


Fig. 1 Path of on-chip data access for CMPs with NUCA cache

2. Related Work

Cache Compression Data compression allows the processor to have better cache utility. Prior work on cache compression exploits frequent data patterns to remove redundancy in it [2] [5]. Hallnor et al. proposed a dictionary-based approach to re-encode the data blocks in the LLC using the registered patterns in dictionary [6]. FPC relies the dominant word patterns in application, like all-"0/1" words, to compress the bits into a sequence of pattern index and a compact header [2]. C-pack utilizes both frequent patterns and dictionaries to support parallel compression of cache lines [4]. Except pattern-based methods, Base-delta-immediate (BΔI) found that the value of words in a block could be represented as difference (delta) from a certain base value [5]. Other solutions seek statistical approach to eliminate value redundancy in cache. For example, SC^2 employs statistical compression and dynamic Huffman coding to increase the compression rate to 3X-4X. In conclusion, prior work proposed a series of cache compression schemes with varied compression rate and overhead as shown in Table. 1 [3]. All of them suffers from performance overhead to some extent.

NoC Compression There are work on packet compression for NoC using the similar compression algorithms targeted on cache or memory. Jin and Zhou use **frequent value compression** to increase the performance and power efficiency of NoCs [7] [8]. Jia et al. uses **delta compression** to achieve better traffic compression effects in NoC for power and latency reduction [9]. Das et al. [10] propose to compress network messages based on zero bits in a word. However, they do not discuss the possibility to unifying the on-chip data compression mechanisms that benefits both NoC and LLC of CMPs or the opportunities to reduce compression overhead with it.

Though there are many prior compression proposals, they mainly focuses on achieving higher compression ratio or less compression complexity. However, fast compression often comes at a cost of compression ratio while efficient compression often means long latency. With DISCO, their greatest concerns on performance penalty will be removed. DISCO utilizes the NoC resources and the opportunities to combine data moving and data compression. It is worth noting that DISCO does not depend on a specific compression method or algorithm to achieve near-zero cost compression.

TABLE. 1 IMPORTANT PARAMETERS OF DIFFERENT COMPRESSION SCHEMES

Method	Comp. Lat.	Decomp. Lat.	Hardware Overhead	Comp. Ratio
FPC	-	*5cycles*	*8%*	*1.5*
SFPC	-	*4cycles*	*8%*	*1.33*
BDI	*1 cycle*	*1~5cycle*	*2.30%*	*1.57*
SC^2	*6cycles*	*8/14cycles*	*1.46%-3.9%*	*2.4*
C-pack		*8 cycles*		

3. Architecture of In-network Compression

3.1 Microarchitecture of DISCO router

To better exploit spatial and temporal locality in on-chip memory subsystem, large-capacity LLCs are often partitioned into multiple distributed compact and fast cache banks, which are defined as NUCA. The cache banks are connected to cores through NoC, so that NoC is responsible for handling the data moving activities including cache block requests and responses. All cache blocks must enter the router of NoC after it departures a core, a cache bank or a memory controller. Fig.2 (a) illustrates the organization of the proposed DISCO router. Except the part added to support in-network de/compression, it is a typical 3-stage router that consists of input buffers, a routing computation unit (RC), a virtual channel allocator (VA), a switch allocator (SA) and a crossbar. When a cache block is read out from a NUCA bank, it is firstly reorganized by the Network Interface (NI) into a packet with multiple independent flits, get attached to a header flit containing the routing information, and then goes into the input buffer of the local router. Afterwards, RC detects the incoming packet and decides its output port according to the destination node ID in packet header and the employed routing algorithm. After that, VA simply receives the header flit and reserves buffer entries (VC) in next router for the packet based on the routing information in header flit. Finally, SA in the router performs arbitration to decide which requesting flit can take over a particular output port in the crossbar when there are multiple packets contending for that direction. Fig. 2 (b) describes the basic stages to route a packet between cache banks and cores. Different from conventional routers, two components are added to DISCO router to enable compression in idle time. The first one is the DISCO compressor connected to the input buffer where cache blocks stay in packet form. Besides that, a DISCO arbitrator is equipped and it cooperates with the RC, SA and VC units to make in-time compression decision.

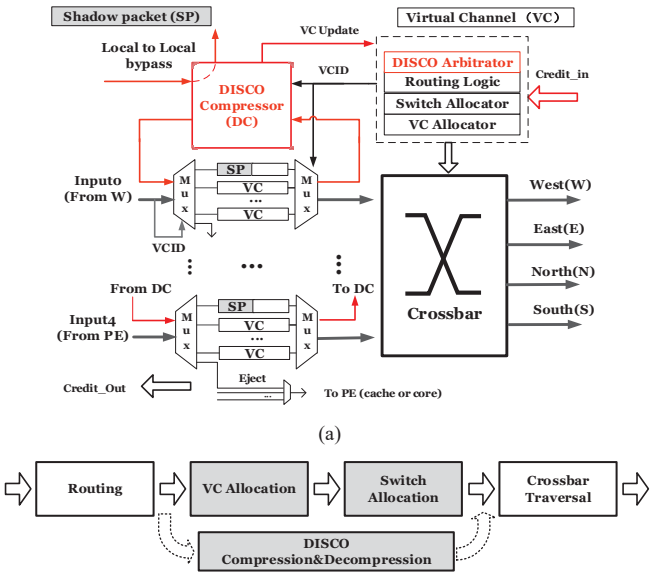


Fig.2 Structure and working mechanism of a DISCO router

3.2 Compression Scheduling with DISCO router

Step-1: Selecting the idling packets in routers

Cache requests and responses are subject to transmission latency when they are packetized and routed across the NoC. The packet latency imposed on cache requests includes two parts: zero-load latency that is exclusively decided by packet size and the hop distance between requesting source and response destination, and contention-induced stall that depends on the traffic intensity and congestion in the network. The former part is predictable with network employing deterministic routing algorithm, while the latter part of latency is the idle time that the packets spend on resource waiting, which is exploitable for cache de/compression.

In details, packets arriving at or injected into a router have to contend for the target direction port or the input buffer (VC entry) of down-stream node if multiple packets are simultaneously heading for the same direction according to routing computation (RC). SW allocator will decide which one is the winner packet depending on flow control information, and also indicate the loser packet with the lowest priority to get the port grant. The aim of DISCO arbitrator is to select one of the idling packets and uses its waiting time for de/compression. To do that, DISCO arbitrator must be informed by VC and Switch Allocator that decide whether a packet loses the resources (VC and crossbar ports) contention. Therefore, the compression arbitration is made at the same time with VC allocation and switch allocation stage as indicated in Fig. 2 (b). The port arbitration results from VA, SA and RC are sent to DISCO arbitrator for compression decision making. For example, as shown in Fig. 2 (a), the VC ID of an idling packet is sent to the DISCO arbitrator if the packet head indicates it's a compressible cache block.

Step-2: Selecting the most suitable packet to be compressed

Step-1 gives the possible candidate packets to be compressed. However, in a congested network, there are often more than one loser packets from the five directions of the router (East, West, South, North and local). However, DISCO arbiter cannot choose a random one and send it to the only compression engine in the router. The reason is because these waiting packets are supposed to be stalled for different amount of time in the router. However, if all the waiting packets marked as compressible are immediately compressed, they are likely to suffer from additional performance penalty caused by a hasty decision. The reason is because the NoC queuing time experienced by packets are fluctuating in a wide range as we observed in experiments. If a packet goes into the compressor, it is hold in it for several cycles and then losses the opportunities to be routed if VA or SA is to grant it the contended bandwidth immediately, which contradicts the flow-control or routing result of the NoC. Therefore, we add a *confidence mechanism* in DISCO arbitrator coordinated with VC and SA to avoid such hasty decisions. As shown in Fig. 3, there are two units in arbitrator: packet filter and confidence counter. Packet filter receives the candidate loser packets, and establish a confidence entry in counter for each packet. The stage of confidence calculation is done in parallel with the packet filtering. The confidence counter takes multiple important factors into account: packet priority, local traffic pressure and remote traffic pressure.

As shown in Fig. 3, confidence counter estimates not only the local contention pressure but also the remote pressure in the down-stream node of the compression candidates, so that it ensures that only the low priority packets who have multiple potential competitors with higher priority will be sent to the DISCO compressor. The reason is because such packets are almost impossible to get serviced either in current or next node during the several cycles time of de/compression operation. The credit signals used to estimate the VC occupancy is a good indicator of routing pressure. For local pressure estimation, the *credit_out* signal in local VC allocator, which is sent to the up-stream router for packet scheduling, is reused in the confidence counter. For remote pressure estimation, the *credit_in* signal from the adjacent router in down-stream direction is counted for the candidate packet as another indicator. With these information, the confidence counting mechanism ensures that no hasty compression decision will be made. Therefore, for each candidate packet selected from step-1, its confidence will be calculated and sent to DISCO compressor only if its confidence surpasses the given threshold C_{th}.

For confidence counting, there are two types of candidate packets to distinguish: packets to be compressed and packets to be decompressed. The calculation of C_{th} for these two types of packet are different: CC_{th} is the threshold for uncompressed packets whilst CD_{th} is the threshold for compressed packets.

For compression candidates, the confidence is calculated and compared to threshold CC_{th} as:

$$C(packet_i) = credit\_in\{RC(packet_i)\} + \gamma \times credit\_out\{VA(packet_i)\} > CC_{th} \quad (1)$$

Wherein $RC(packet_i)$ is packet-*i*'s destination port in the next hop, which is decided by RC. $VA(packet_i)$ indicates the local VC entry where packet-*i* stays, which is also associated to certain direction. Such information can be obtained from VA and RC as illustrated in Fig. 3. For decompression candidates, the confidence is calculated and compared to threshold CD_{th} as:

$$C(packet_i) = credit_{in}\{RC(packet_i)\} + \alpha \times credit_{out}\{VA(packet_i)\} - \beta \times RC\_Hop(packet_i) > CD_{th} \quad (2)$$

For decompression arbitration, the confidence counter works in a slightly different way. Besides the local and remote port contention information, it also takes the routing status of the packet into account. Specifically, the confidence counter will consider the packet's current distance away from its destination node, $RC\_Hop(packet_i)$. $RC\_Hop(packet_i)$ is obtainable in RC unit and used to avoid the case of early decompression, which is less desirable because early decompression deprives the NoC of the opportunity the reduce traffics through compression. We use the real workload traces from the NoC simulator to train the empirical parameters: local coefficient α and distance coefficient β in the counter. In practice, the values of CC_{th} and

CD_{th} is also determined based on the experimental observation. In experiments, the performance of the two parameters are proved to be dependent on the NoC congestion condition and the configuration of NoC as well, i.e the stage number, VC depth and flow-control method. However in this work, these two parameters are assumed deterministic in NoC for simplicity so that they will not induce extra overhead of congestion-aware mechanism or the overhead of on-line threshold calculation.

Fig. 3 depicts the working mechanism of DISCO arbitrator, which includes a filter that collects the candidate packets header from SA and then selects the final de/compression packet depending on *credit_out* in VA, *RC_Hop* in RC and *credit_in* from adjacent nodes.

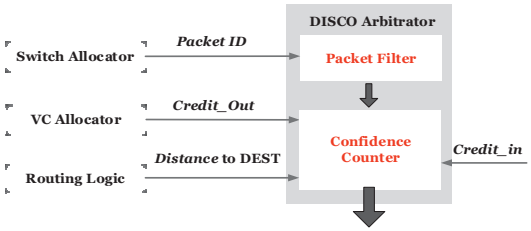


Fig. 3 Working mechanism and organization of DISCO arbitrator

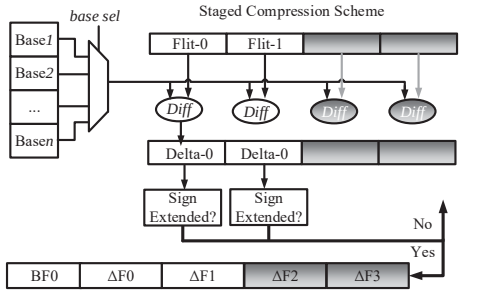
Step-3: **Entering into DISCO compressor for data compression**

When compression arbitrator has determined the ideal packet to be compressed, it copies the packets from input buffer into the DISCO compressor as demonstrated in Fig. 2 (a), but still keeps the packet duplicate in the virtual channel as a shadow packet (SP). SP has a shadow invalid bit indicating whether it is locked in case of being illegally reassigned to a port before compression completes. Only after de/compression completes, the shadow flits will be replaced by the new flits. The buffer capacity saved by compression is also released and become immediately available for the incoming flits.

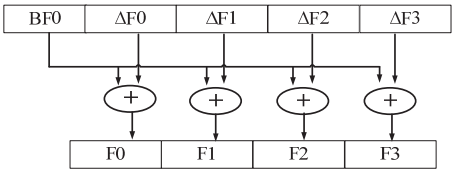
The other reason to keep the shadow packet in buffer is that SP is useful for enabling **non-blocking compression**. Although we have confidence mechanism to avoid the case of hasty decision of packet compression or decompression, there are still possible cases when the intended port is released during packet compression or decompression. Especially the latency of decompression makes the network performance become sensitive to mis-prediction made by confidence counter. After all, for some workloads, the traffic changes are both spatially and temporally elusive in NoCs, and precise prediction of stall time cannot be guaranteed. In this case of mis-prediction, SP is schedulable by SA and immediately sent to the released port with non-blocking de/compression. Meanwhile, the flits under process are invalidated by the scheduler.

The DISCO compressor unit could be implemented to support different compression algorithms like FPC/BDI [5] [2]. Though they differ in details, different compression algorithm are mostly based on exploitation of frequent patterns to eliminate redundancy in data encoding and achieve high compression rate. In this paper, we use a representative delta-based compression and modify it to suit the structure of router as an illustration of this procedure. Fig. 4 depicts the basic unit of delta-based compressor in DISCO router. Its organization is simple: for a compressed block that consists of a base value BF0 and subsequent delta values ΔF0, ΔF1, ΔF2 and ΔF3. Assuming that the first 8-byte flit is the base, if the difference between each 8-byte flit and the base is less than 255, so each flit will be represented by a single byte. The compressed version of this packet load would be an 8-byte word, 8 1-byte words and a few bits in the packet header including the fields of base size and delta size. In addition, the compressor allows the usage of multiple base sizes and multiple base values, for example, zero flit, zero half-flit (4 bytes) or the first flit. In this paper, the first base is always the first partition (the first flit in this work), and the second base is zero flit. For each time in compression, DISCO compressor compares all flit to the two bases, obtain the smaller differences, and use them as delta values in the compressed packet as illustrated in Fig. 4 (a).

The illustrated DISCO compressor consists of multiple compressor units. Fig. 4(a) shows only one unit that takes four flits as an input. This unit outputs the compressed flit if the extended sign indicates it is compressible. The compressor selection logic is used to choose the smallest compressed flit size from the multiple compression units. The DISCO compressor has an additional component that re-organizes the compressed flits into new flits to decrease the flit count in a packet, which is beneficial to buffer and link utilization. Meanwhile, the reserved space will not be fully occupied because the packet shrinks after compression. Thus, the VC allocator and *credit_out* should be updated to retrieve the saved space in the input buffer.



(a) Delta-based compression unit in DISCO compressor



(b) Delta-based decompression unit

Fig. 4 Compression and Decompression units in DISCO compressor

As shown in Fig. 4 (b), the decompressor is very simple. It adds the base to the differences to obtain the uncompressed packet. It also support non-blocking operation similar to the compressor.

3.3 The implication on all aspects of NoC

A. Restriction on Flow control policy

As we have described in previous sections, one important constraint on in-network compression is that the whole packet has to be compressed within one node, which has a direct impact on flow control policy. This requirement goes smooth with two conventional flow control methods: *store-and-forward* and *virtual cut-through*, which both support whole packet transmission between two nodes and make sure one whole packet stays within one node for compression. However, *worm-hole* flow control is trickier to deal with since it separates flits of one packet when no enough space in the downstream buffer is allocated.

In worm-hole routing, a packet is often separated. In this case, an incomplete packet in one node cannot not be compressed with conventional compression method. There are two methods to enable compression in worm-hole routing. The first one is to impose constraint on both compression and router configuration simultaneously, *i. e.*, increasing the depth of input/output buffers of router to ensure the flits of a packet stay together and forwarding a whole packet to the compressor only. Otherwise, adding the support like Whole Packet Forwarding mechanism (WPF) also make sure that no packet flits are separated in NoCs [11]. It is easy to do that by disallowing a non-empty VC to be re-allocated to new flits. This approach increases the overhead of buffer entries.

The other method is to support **separate compression** with independent flits from a packet, which is adopted in DISCO. For example in Fig. 4 (a), once the flit-0 and flit-1 occupy the empty VC of input before flit-2 and flit-3 come, they are sent to compressor without the left flits in upstream router. In compressor, there are multiple base registers that store different bases. The two flits are compared to zero base and the head base, i.e. flit-0, to obtain the deltas. However, the same bases will be kept in base registers for the remaining flits of the packet, i.e. flit-2 to flit-7, until all of flits arrived and get compressed continuously. Separate compression does not need the constraint of whole packet transfer or deep input buffers. However, separate compression sacrifices the compression rate. For example, two 32-bit flits are compressed but only into an 8byte base+1byte offset, however, the compressed result with 7 bytes zero bubbles behind still occupies two whole 32-bit flit entries in buffer, and no space will be saved. However, if the complete packet of 4 flits are compressed at one time, the result will be 8byte-base+8bytes offset, the whole packet takes only two buffer entries. Therefore, we need hardware support in DISCO compressor to merge the separated flits into a continuous packet without zero bubbles between them. As the example in Fig. 4 (a) shows, the 8+1 bytes leaves a bubble space in the buffer entry, and it attaches a new tag at the end of offset bit, so that the ending offset in the first part of compressed flits can be directly concatenated with the beginning offset bytes in the second part of the separately-compressed packet. In this way, zero bubbles in buffer entries can be eventually eliminated.

B. On packet scheduling

Packet scheduling decides what kind of packets should be prioritized in bandwidth contention. Generally, the read requests and responses are thought in the critical path so that they receive a higher priority in scheduling. Except this rule, other packets are often treated equally according to conventional round-robin or FIFO scheduling. However, with the intuition that it might increase the bandwidth utility, we add a second rule to the scheduling policy to let the compressible but uncompressed packets to have a lower priority in routing. In this way, such packets could be de/compressed at a higher probability, and also save the routing resources for other packets without idle time to exploit.

C. Cost-effective Compression for CMPs

For cache-coherent multi-core processors, there are a variety of packet types that are recognizable to the NoC. For example in a cache-sharing CMP with MOESI protocol, there are mainly three types of packets: request packet, response packet and coherent packet. Request packet carries operation command to a cache banks, on-chip directory or to the on-chip memory controller. Response packet carries the data blocks written-back to or sent from a LLC bank, a core (L1), or the memory controller. Coherent packet carries the data synchronization signals that help invalidate a data copy or consent a cache-modifying operation, and etc. Among them, response packet that carries the payload of cache blocks occupies the majority of on-chip bandwidth because of its size, which is consistent with a cache block size. Therefore, it is more profitable to ignore coherent/request packet and compress response packets only to save both bandwidth and power with DISCO.

4. EVALUATION

4.1 Experimental Setup

We use a full system simulator GEM5 and a cycle-accurate NoC simulator Booksim to faithfully model the CMP architecture with DISCO [12] [13]. In the CMP, 16 tiles are connected via the 4×4 NoC. Each tile consists of an x86 core and a SRAM cache bank in 45nm process technology. The detailed system configurations are described in Table 2. The last level NUCA cache is 4MB, and shared by the 16 cores. To evaluate DISCO with real applications, PARSEC-2.1 benchmark suite is used in GEM5 [14].

As we have mentioned, there are multiple packet types in the cache-coherent system. For the data packet, which originated as the write-back block or the response to data transfer request, the compressor uses 8-byte-base 1-byte-Δ unit for the 64 bytes cache line so that the payload of such data packets will be compressed in a 1BF+7ΔF form.

For Comparison Our unified in-network compression scheme is also compared with state-of-the-art solutions: cache compression (CC), cache and NoC compression (CNC). For CC that only considers cache compression, a de/compression unit is equipped in every cache bank to compress all input and output data in cache controller. In contrast, CNC not only enables cache compression as CC, and also equips each Network Interface with a de/compression unit that decompress all ejected packet and compress all injected packets as in [9]. Besides, in each experiment, the same compression algorithm with identical compression rate, speed and overhead is employed in CC, CNC and DISCO for fair comparison, because DISCO is not focused on or restricted to a particular compression algorithm. This feature enables the designers to trade-off between the compression rate, overhead and speed of various compression methods. In this work, four representative compression policies are used in CNC, CC and DISCO for investigation.

TABLE. 2 BASELINE SYSTEM PARAMETERS

Processor Core	16/64 core, 2GHz OoO, 4 issue, x86, 32KB 4-way I-cache, 32KB 4-way D-cache,
NoC Topology	$4\times4/8\times8$, Mesh, XY routing
Router	3 pipeline stages, wormhole flow control, 8-flit depth buffer, 2 Virtual Channels
Coherence	MOESI protocol
L2 Cache	Shared, NUCA, 8 way, 64B lines, 16-banked, LRU replacement, 4-cycle hit (NoC delay not included)
Memory	4G DRAM, 1 rank, 1 channel, 8 banks,
DISCO	Non-blocking compression, Delta-based 1 cycle compression, 3-cycle decompression [5]

4.2 Performance and Energy Consumption

On-chip data access latency The NUCA cache and the NoC altogether form as the on-chip memory subsystem of a CMP, so its performance could be directly estimated by recording the *average NUCA data access latency* triggered by L1 cache misses, which includes both NoC delay and cache bank access delay for a certain workload. The results are normalized to the same system with cache compression but without the de/compression overhead as the ideal case. Fig. 5 shows the performance of CC, CNC and DISCO. CC, CNC and DISCO in Fig. 5 uses the delta-based compression method as described in section-3.2. The performance of DISCO surpasses that of CC by 12%, and beats CNC by 10.1%.

We reevaluate the results in scenario when other methods like FPC and SC^2 [2] [3] are also equipped with DISCO, CC and CNC. The two most advanced compression algorithms and hardware are assumed to be added to cache or NI or router in experiments. Their performance and overhead are also described in Table. 1. Fig. 6 shows the performance of DISCO, CC and CNC with different compression algorithms other than the delta-based one. It is seen the average on-chip data access performance of DISCO is even higher. DISCO achieves 11%~16% performance boost on average over the other two counterparts. In general, for the three algorithms evaluated, DISCO achieve the best performance boost with SC^2, and it reduces 16.7% average latency over CNC, and 15.5% over CC because the de/compression delay of SC^2 is longer than both delta-based and FPC. However, with the optimization of DISCO, SC^2, which achieves the highest compression rate, can also significantly reduce its performance penalty added to the cache access path. In Fig. 6 the average performance of CNC lags behind CC because the two-level compression in cache and NI sometimes causes excessive latency to

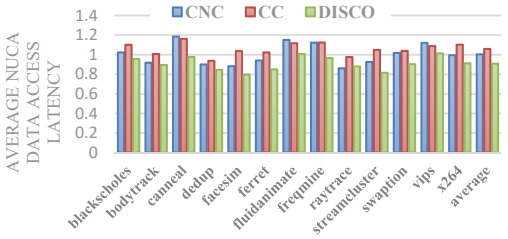


Fig. 5 Performance Comparison with delta-based compression

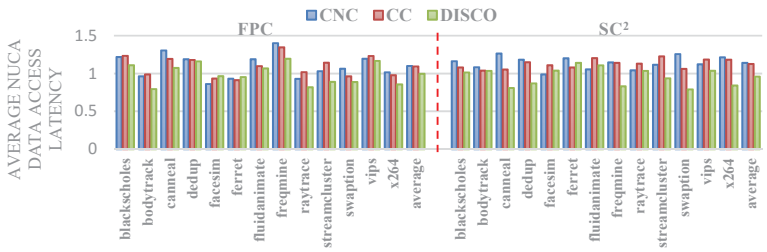


Fig. 6 Performance Comparison with FPC and SC²

routing in order to exploit the left redundancy in cache blocks.

Energy Study As a unified on-chip data compression architecture, DISCO saves considerable data transmission energy and system energy by increasing the utility of LLC. However, it also increases energy by adding a DISCO compressor and arbitrator. We implemented DISCO hardware in 45nm process technology with the synthesis library of FreePDK45, and use Design Compiler to create the static/dynamic power profile, which is used in simulator for energy evaluation. The NoC energy is obtained using Orion-2.0 [15], while the power parameters of NUCA cache are obtained from CACTI [16]. Fig. 7 shows the energy comparison of the on-chip memory subsystem including the NoC and NUCA cache under realistic Parsec-2.0 workloads. The power statistics are normalized to the baseline case **without** cache or NoC compression. It is seen from Fig. 7 that DISCO reduces about 9.1% total energy consumption over CNC and 8.3% over CC on average for the Parsec-2.0 benchmarks. The major energy saving comes from the reduced communication activities and accelerated performance. Compared to the baseline without data compression, DISCO memory subsystem consumes only 73.3% of the total energy consumed by baseline to complete the workloads on average.

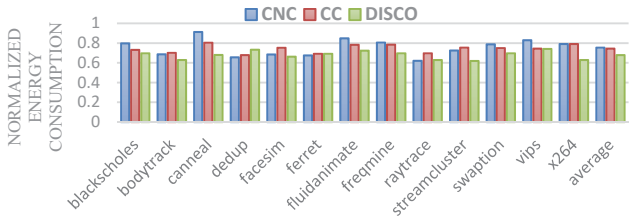


Fig. 7 Energy Comparison with delta-based compression scheme

Previous experiments show the impacts of DISCO on performance and energy consumption in a 16-banked NUCA cache. Fig. 8 shows the performance scaling of DISCO with CMPs of varying scale. The performance metric is still the average on-chip data access latency normalized to the ideal case. When the NUCA cache only comprises 4 distributed banks, the performance improvement of DISCO is not significant. However, with the CMP increases to an 8x8 Mesh topology with 64 NUCA banks, the performance gain of DISCO increases from 10% to 22% on average when it is compared to CC, which means DISCO has a very good scalability.

4.3 Overhead Estimation

The hardware components of DISCO and the assumed 3-stage router are implemented and synthesized with 45nm FreePDK45 library. For area overhead, the delta-based DISCO de/compressor and the arbitrator for 64b flits increases 17.2% more area of the original router in total. Considering the total area of 4MB NUCA cache, it adds less than 1% area. However, for CNC with both cache compression and within-NI compression, DISCO can saves about half of the area overhead. Another aspect of overhead brought by DISCO is the added power consumption to the bare system without data compression, which is already included in the energy estimation results in Fig. 7.

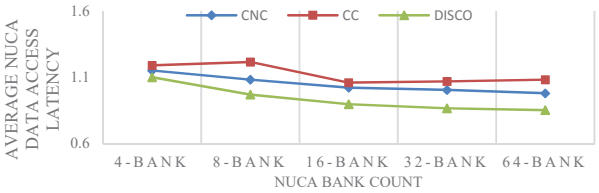


Fig. 8 Scalability Comparison of DISCO compression

5. Conclusion

We propose a unified in-network data compression solution for CMPs, DISCO, to mitigate the performance overhead that limits the application of cache compression. It is proved that DISCO can effectively hide the latency of cache de/compression by exploiting the packet idling time across the NoC, and finally increase the cache space and NoC bandwidth utility with one single solution. The experiments shows that DISCO can significantly boost the performance of the whole on-chip memory subsystem by 11%~16%, decrease the energy consumption by 21% on average. Besides, DISCO is in parallel with different compression algorithms and methods with different compression ratio and overhead. It help make the controversial cache compression methods, which have high-compression ratio but high overhead, more practical in energy-efficient CMPs.

6. ACKNOWLEDGMENTS

This work was supported in part by National Natural Science Foundation of China under Grant No. 61432017, 61176040, 61504153, 61402146 and 61521092. The corresponding author is Huawei Li.

7. References

[1] P. Lotfi-Kamran et. al., “Scale-out processors,” in Proc. *ISCA*, 2012
[2] A. Alameldeen et. al., “Adaptive cache compression for high-performance processors,” in Proc. *ISCA*, 2004.
[3] A. Arelakis et. al., “SC2: a statistical compression cache scheme,” in Proc. *ISCA*, 2014.
[4] X. Chen et. al., “C-pack: a high-performance microprocessor cache compression algorithm,” in *IEEE Trans. on VLSI Systems*, 2010.
[5] G. Pekhimenko et. al., “Base-delta-immediate compression: practical data compression for on-chip caches,” in Proc. *PACT*, 2012.
[6] E. Hallnor and S. Reinhardt, “A Unified Compressed Memory Hierarchy,” in Proc. *HPCA*, 2005.
[7] Y. Jin et. al., “Adaptive data compression for high-performance low-power on-chip networks,” in Proc. *MICRO*, 2008.
[8] P. Zhou et al., “Frequent value compression in packet-based NoC architectures,” in Proc. *ASPDAC*, 2009.
[9] J. Zhan et al., "NoΔ: Leveraging delta compression for end-to-end memory access in NoC based multicores," in Proc. *ASPDAC*, 2014.
[10] R. Das et al., “Performance and power optimization through data compression in Network-on-Chip architectures,” in Proc. *HPCA*, 2008.
[11] S. Ma et al., “Whole packet forwarding: Efficient design of fully adaptive routing algorithms for networks-on-chip,” in Proc. *HPCA*, 2012.
[12] C. Bienia et. al., “The PARSEC Benchmark Suite: Characterization and Architectural Implications,” in Proc. *PACT*, 2008.
[13] D. Wang et al., “DRAMsim: a memory system simulator,” in SIGARCH Comput. Archit. News , 2005.
[14] N. Binkert et. al., “The gem5 simulator,” in SIGARCH Comput. Archit. News 39(2):1-7, 2011.
[15] A. Kahng et. al., “ORION 2.0: a fast and accurate NoC power and area model for early-stage design space exploration,” in Proc. *DATE*, 2009.
[16] N. Muralimanohar et al., “Architecting Efficient Interconnects for Large Caches with CACTI 6.0,” in IEEE *Micro* 28(1): 69-79, 2008.