In-Cache Streaming: Morphable Infrastructure for Many-Core Processing Systems

Nuno Neves, Adrien Mussio, Fabien Gonçalves, Pedro Tomás, and Nuno Roma

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
Rua Alves Redol, 9, 1000-029 Lisboa - Portugal

Abstract. Although conventional cache structures often reduce or mitigate the memory wall problem, they often struggle when dealing with memory-bound applications or with arbitrarily complex memory access patterns that are hard (or even impossible) to capture with dynamic prefetching mechanisms. Stream-based communication infrastructures have proved to efficiently tackle such issues in certain application domains, by allowing the programmer to explicitly describe the memory access pattern to achieve increased system throughputs. However, most conventional computing architectures only adopt a single interfacing paradigm, making it difficult to efficiently handle both communication approaches. To circumvent this problem, an efficient unification is herein proposed by means of a seamless adaptation of the communication infrastructure, capable of simultaneously providing both address-based and stream-based models. This newly proposed in-cache streaming infrastructure is able to dynamically adapt memory resources according to runtime application requirements, while mitigating the hardware requirements related to the co-existence of both cache and stream buffers. The presented experimental evaluation considered arithmetic, bioinformatics and image processing applications and it showed that the proposed structure is capable of increasing their performance up to 14x, 5x and 12x, respectively, with a limited amount of additional hardware resources.

1 Introduction

The ever increasing demand for computational processing power at a significantly low-energy consumption has pushed the research for alternative heterogeneous and often specialized many-core processing architectures. However, the design of such architectures is usually mainly focused on the processing blocks, often neglecting the power/performance impact of the inherent data transfers and general data indexing. In fact, a common approach is to rely on conventional cache structures to avoid the usually high memory access latencies. However, although they are well suited for compute-bound applications, they struggle when the application dataset is very large and does not fit in the cache, or when dealing with memory-bound applications, or even with arbitrarily complex memory access patterns, where data locality cannot be efficiently exploited.

Several solutions have been proposed to handle those applications and access patterns, usually relying on efficient prefetching techniques [1,2] and/or stream-based communication systems capable of handling complex data-patterns [3,4]. However, although viable, these techniques can hardly deal with certain application domains (e.g. those based on graphs, on dynamically indexed procedures or on non-deterministic/runtime generated data access patterns), whose implementation is usually more efficient with conventional cache-based approaches.

This duality presents an interesting opportunity to combine both approaches in a single and adaptable communication infrastructure that is capable of in-time switching its paradigm to better suit a running application. Moreover, by combining the advantages of such approaches in a single structure, highly efficient and adaptable communication systems can be deployed, providing the means for exploiting both data-locality and complex data access patterns.

Accordingly, a novel in-cache streaming architecture is herein proposed based on a dynamic adaptation of cache memories at the processing nodes, in order to exploit both stream-based and address-based communication paradigms with the same hardware infrastructure. The proposed architecture is based on a hybrid in-cache stream controller that takes advantage of a conventional n-way set-associative cache organization, by making each way individually usable as a stream buffer, capable of accommodating multiple streams. At the main memory side, the proposed infrastructure relies on a specially devised shared memory controller that combines a conventional address-based memory access controller with an efficient stream generation controller (that deploys the stream-based communication paradigm previously proposed in [4]). The communication between all the system's Processing Elements (PEs) and the main (shared) memory is assured by a high-performance and low-footprint ring-type Network on Chip (NoC), supported by a dedicated message-based protocol.

The envisaged approach contrasts to (and complements) other established strategies based on the sole exploitation of adaptable data-processing structures. Several examples use dynamic reconfiguration capabilities of nowadays Field-Programmable Gate Array (FPGA) devices, where the processing infrastructures can adapt to the target application by reconfiguring its PEs in runtime [5–7]. However, such adaptation is usually only applied to the processing architecture, since the reconfiguration process still results in non-negligible time overheads and power dissipation that can greatly impact the performance and energy consumption of the communication infrastructure. Nonetheless, energy-efficiency has been targeted with the adaptation of the communication subsystem, such as cache architectures with dynamically reconfigurable parameters [8] (such as size and associativity); power-gated hybrid designs built with combinations of different memory technologies [9]; or partial reconfiguration of local scratchpad memories into second level caches, to support implicit and explicit communication [10]. However, although widely adaptable, all these approaches still incur in inevitable delays in the reconfiguration process and struggle when dealing with complex memory access patterns. On the contrary, the efficient and adaptable communication structure that is now proposed is deployed by exploiting a coarser-grained adaptation, that is capable of efficiently and seamlessly switching between address-based and stream-based communication paradigms.

The proposed in-cache streaming architecture capabilities for prefetching and data reutilization through stream-based communication were demonstrated through an experimental evaluation using three benchmark applications. When compared to a baseline conventional cache setup, the obtained result, with a system configuration with 16 PEs, show performance increases of up to 14x for a block matrix multiplication application, 5x for a biological sequence alignment algorithm and 12x for an histogram equalization kernel.

2 Data Streaming with Compiler-Assisted Prefetching

In many common applications (including memory-bound), the PEs are able to perform elementary operations much faster than the main memory accesses, leading to considerable performance losses when off-chip memory modules are accessed. Although a multi-level cache hierarchy can considerably mitigate such overheads, it still presents several drawbacks, namely those resulting from the common utilization of shared communication infrastructures, allied with the inherent main memory access concurrency and bus contention; and also those resulting from the intrinsic characteristics of the executed applications (e.g. memory-bound kernels or complex memory access patterns), which in turn result in reduced data-locality exploitation.

2.1 Dynamic and Static Prefetching

Advanced *static* and *dynamic* prefetching techniques are often considered to hide data transfer overheads behind the PEs computation, by fetching data from memory in advance and storing it in local buffers or caches.

Dynamic prefetching usually relies on complex dedicated modules aggregated to the PEs (or caches), which analyze the recent memory access pattern and try to predict future accesses based on prediction heuristics. The most commonly used techniques are based on stride prefetching, where the prefetcher calculates the difference (or stride) between the most recent requested addresses and issues requests to memory for subsequent addresses based on that difference. However, although such an approach allows a complete abstraction of the prefetching procedure from the application perspective, it can fall short in arbitrarily complex access patterns. Moreover, this technique imposes an increased amount of resources, often related to the adopted level of prefetching aggressiveness [1].

In contrast, static prefetching is usually performed with the aid of compile-time procedures, where the code is pre-analyzed to extract/model the application memory access pattern. Such information is then fed to on-chip prefetching modules, which autonomously generate the required memory address sequence. Such an approach requires far simpler hardware structures, since no on-time analysis is performed, thus resulting in lower-footprint and more energy efficient controllers, at the cost of an increased pre-processing effort. Furthermore, static prefetching

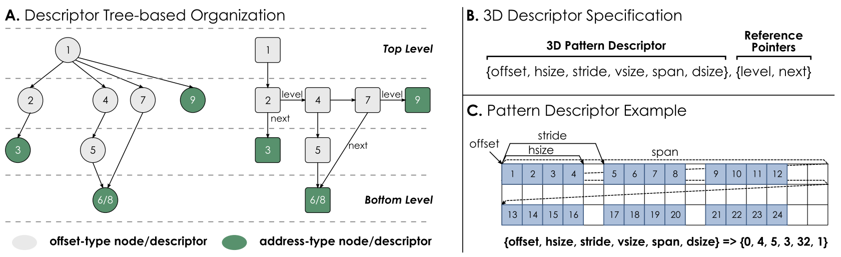
A. Descriptor Tree-based Organization
Top Level
Bottom Level
level
next
offset-type node/descriptor
address-type node/descriptor
B. 3D Descriptor Specification
3D Pattern Descriptor
Reference Pointers
{offset, hsize, stride, vsize, span, dsize}, {level, next}
C. Pattern Descriptor Example
offset
stride
hsize
span
{offset, hsize, stride, vsize, span, dsize} => {0, 4, 5, 3, 32, 1}

Fig. 1. 3D data-pattern descriptor specification, illustrating its (A) tree-based hierarchical organization, (B) the descriptor parameter encoding, and (C) a pattern description example. The numbers in (A) indicate the order in which the descriptors are solved and in (C) the order in which data blocks are accessed.

also promotes the exploitation of highly efficient stream-based communication means, allied to several other approaches to further improve the communication efficiency, such as data reutilization and reorganization, complemented with implicit stream manipulation operations [4].

2.2 Stream-based Communication and Data Reutilization

Instead of relying on prefetching structures, stream-based communication systems rely on dedicated address generation units to pre-fetch the data, according to pre-determined memory access sequences, and on generating the requested data stream. Such units are commonly devised based on the fact that, independently of their application domain, many algorithms are characterized by memory access patterns represented by an n-dimensional affine function [11], where the memory address (y) is calculated based on an initial *offset*, increment variables x_k and *stride* multiplication factors, as follows:

$$y(x_1,\cdots,x_n) = \mathit{offset} + \sum_{k=1}^{n} x_k \times \text{stride}_k \ , \ x_k \in \{0, \cdots, \text{size}_k\}$$

Since such representation allows indexing many regular access patterns, it is commonly used by Direct Memory Access (DMA) controllers and other similar data-fetch controllers, although typically restricted to 2D patterns (n=2).

Naturally, to describe other arbitrarily complex memory access patterns, affine functions with higher dimensionality can be used, and even allied with hierarchical combinations of several functions, where the affine functions in the higher levels of the hierarchy are used to calculate either the *offset* or the *stride* of the functions in the lower levels. Hence, each complex data stream can be defined by a set of *descriptors*, each encapsulating the set of parameters required to generate the sequence of addresses at a given hierarchy level.

Accordingly, the herein proposed stream-based infrastructure adopts the 3D tree-based descriptor specification, previously proposed in [4] (depicted in Fig. 1). Such memory access pattern is represented by the tuple {OFFSET, HSIZE, STRIDE, VSIZE, SPAN, DSIZE, LEVEL, NEXT}, specifying the starting address of the first

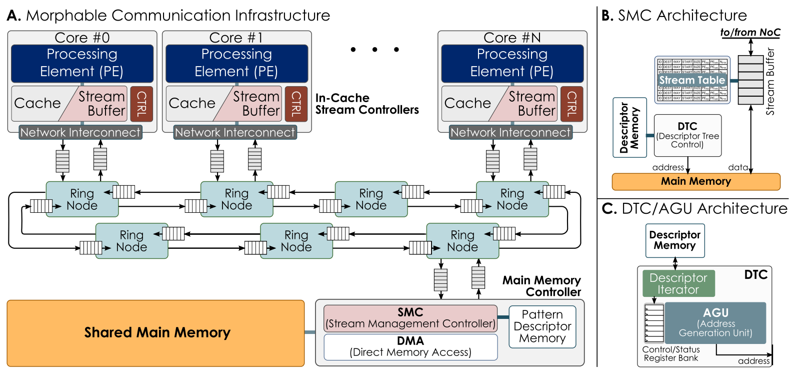

Fig. 2. Morphable communication infrastructure overview (A), comprising the proposed In-Cache Stream controllers at the PEs interface, the main memory controller and a ring-based NoC. The main memory controller is composed of a SMC (B), responsible for generating/storing data stream to/from the main memory, to which the address generation is performed by a dedicated DTC (C).

memory block (OFFSET), the size of each contiguous block (HSIZE), the starting position of the next contiguous block with relation to the previous (STRIDE), the number of repetitions of the two previous parameters (VSIZE), the starting of the next 2D pattern in relation to the previous (SPAN), and the number of repetitions of the four previous parameters (DSIZE). Also, several descriptors can be combined in a tree-based hierarchical scheme (depicted in Fig. 1.A), in which multiple parent-child relations are established between descriptors, representing dependencies between different descriptor levels. Hence, each descriptor has a reference to a child descriptor (NEXT) and a reference to a descriptor that shares the same parent descriptor (LEVEL).

To allow detaching the PEs computational effort from the memory address generation, and to promote the re-utilization of data streams among multiple PEs, multiple address-generation units can co-exist within a single many-core system. Hence, to maximize the utilization efficiency of the available memory bandwidth, a *stream management* unit, included in the memory controller (see Fig. 2.A), is used to broadcast multiple streams (from the main memory) to one or more PEs, or to organize the writting of data from multiple streams (generated by the PEs) to the main memory. On the other hand, special-purpose stream controllers are located next to the PEs, to manage the flow of data into/out of each PE, effectively allowing data to be directly streamed from one PE to another, or to be broadcasted to multiple PEs or to the main memory.

3 In-Cache Streaming Architecture

The herein proposed in-cache streaming architecture allows each individual PE to seamlessly switch its local communication infrastructure between two distinct paradigms: *i*) conventional *memory-addressed* data access; and *ii*) *packed-*

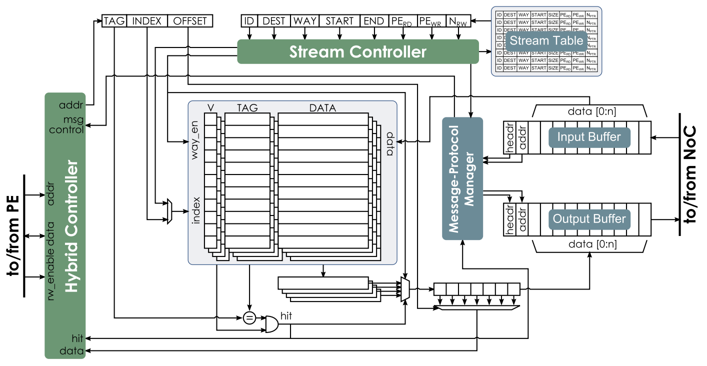

Fig. 3. Hybrid Controller architecture. The *cache controller* and the *stream controller* (supported by the information stored in the stream table) perform an exclusive access to an n-way set-associative cache memory depending on the requests received from the PE and from the communication infrastructure.

stream data access. However, to avoid a complete switching of the two paradigms, which could result in potential performance penalties in non-pure streaming applications, the proposed approach allows morphing a PE n-way set-associative cache memory into a set of n_1 cache ways plus $n-n_1$ stream buffers, each capable of holding multiple streams. Accordingly, not only does the proposed approach support both *memory-addressed* and *packed-stream* data accesses, but it also supports mixed scenarios composed of compile-time predictable and non-predictable/runtime generated memory access patterns. To attain such a morphable infrastructure, the proposed approach relies on an in-cache stream controller to seamless adapt (in runtime) the cache memory according to the instantaneous requirements of the running application (see Fig. 2.A).

3.1 Hybrid Cache/Stream Infrastructure

The proposed in-cache streaming controller, supported by a specially devised main memory controller, comprises two independent modules: a *hybrid cache controller* and a *stream controller* (depicted in Fig. 3), together with an internal n-way set-associative memory that is managed by one of these modules at a time. The adoption of such a switched control structure (instead of relying on dynamic reconfiguration) ensures an immediate switch of the communication paradigm, since no reconfiguration time is imposed.

In-Cache Stream Controller: The default *memory-addressed* communication paradigm can be assured by a conventional *cache controller* (see Fig. 3), using any arbitrarily replacement and write policies. Notwithstanding, the used controller is implemented by means of a simple and efficient hardware structure that deploys a write-through-invalidate, write no-allocate snooping protocol on the local memory, managed by a binary-tree-based Pseudo-Least Recently Used (LRU) replacement policy.

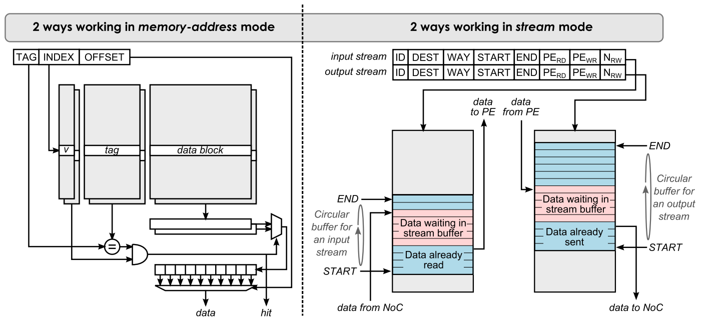
2 ways working in memory-address mode
2 ways working in stream mode
TAG
INDEX
OFFSET
v
tag
data block
data
hit
input stream
output stream
ID
DEST
WAY
START
END
PE_RD
PE_WR
N_RW
data to PE
data from PE
END
Circular buffer for an input stream
START
Data waiting in stream buffer
Data already read
data from NoC
Data waiting in stream buffer
Data already sent
Circular buffer for an output stream
data to NoC

Fig. 4. Configuration example, where a 4-way cache is configured to use 2 ways for conventional memory-address mode and 2 ways for stream mode.

The cache access time is limited to two clock cycles (disregarding cache miss penalties) and hit/miss-related action is taken according to the coherence and consistency protocols in place. PE requests are only answered with a *wait* state when there is a read miss, until the required data is fetched. Upon a write miss scenario, the written data block is immediately sent to the main memory and is followed by an invalidation broadcast, thus minimizing the waiting times and the number of on-the-fly messages in the communication infrastructure.

On the other hand, in order to reuse the resources of the n-way set-associative cache memory for a stream-based communication, its access mechanism has to be conveniently adapted. Hence, each cache way is viewed as an independent buffering structure and it is accessed with a dedicated set of read and write pointers to the memory region where a stream is stored. This transforms the n-way set-associative memory in m independent stream buffers, each capable of storing multiple streams, while allowing the remaining $n-m$ ways to be accessed using traditional *memory-address* load/store operations (see Fig. 4).

Accordingly, the stream-based paradigm requires a set of auxiliary data structures (stored in a programmable *stream table*), including the information and the state of every stream currently stored and handled by the controller. Each table entry (depicted in Fig. 3) comprises: *i)* a unique stream identifier; *ii)* the way used for buffering the stream; *iii)* pointers to the start and end of the buffering region within the way; *iv)* pointers for identifying the PE local read-/write positions in the inbound/outbound stream; *v)* the stream destination (own identification, if it is an incoming stream); and *vi)* a read/write pointer for identifying the current read/write position for a *Message-Protocol Manager*, which transparently handles the communication of the data into/out of the PE.

Hence, whenever a read/write request is performed for a given stream identifier (see Fig. 4), the local memory is accessed according to the information depicted in the stream table, with the consequent update of its read/write pointers. Outgoing streams are automatically sent as soon as they become available and its transmission is granted by the scheduling manager of the processor aggregate. However, the output transmission does not immediately erase the stream data from the local memory, allowing the data to be reused by the PE.

Main Memory Controller: The in-cache stream controllers are served by a remote main memory controller (see Fig. 2.A), composed of: *i)* a low-profile DMA controller, to perform address-based memory operations; and *ii)* a Stream Management Controller (SMC) (depicted in Fig. 2.B), which generates and saves the streams, according to the patterns described by the hierarchical set of descriptors stored in the *pattern descriptor memory*.

The SMC memory access is handled by a special Descriptor Tree Controller (DTC) [4] that deploys the 3D descriptor specification and resolves the procedure described in Section 2.2. Accordingly, the DTC (depicted in Fig. 2.C) is composed of: *i)* a tree iterator, that manages the flow of the descriptor tree; and *ii)* an Address Generation Unit (AGU), that generates the correct sequence of memory addresses, according to a given descriptor. On the other hand, the stream generation/storage is performed by temporarily saving the data in a *stream buffer*, redirecting it (either to the PEs or the main memory) according to a local stream table (as in the in-cache stream controller) (see Fig. 2.B).

3.2 Interface Configuration and Parameterization

To handle both memory-address and stream-based read/write requests at each PE, a generic and parameterizable interface is provided. In particular, each PE request to the cache addressing space is handled by the *cache controller*, whereas requests to the stream addressing space are handled by the *steam controller*, where the stream identifier is encoded in the interface's address and the local memory is accessed according to the stream table.

The hybrid controller interfaces with the communication infrastructure by means of two input/output register-based buffers. Such an approach not only allows contention mitigation through intermediate buffering, but it also provides isolation between the PEs and the interconnection operating frequencies, allowing them to operate with different clock frequencies. Each buffer accommodates a complete message to/from the NoC. Hence, depending on the assigned message type (see protocol definition in Section 3.3), incoming messages are handled either by the *cache controller* or the *stream controller*. Outgoing messages are generated by one of the controllers, depending on which is activated at the time.

3.3 Unified Message-Passing Protocol

To abstract the underlying ring-based NoC infrastructure from the PEs morphable interface perspective, and to keep the impact on the performance of the inter-communication between the system components as low as possible, a simple message-passing protocol was adopted, which consists on a 32-bit *header*, an optional memory *address* and a number of *data* words that, at most, add up to the size of a cache line. The *header* is composed of: *i)* a message identification; *ii)* flags for invalidate, read/write and data access mode (*memory-addressed* or *packed-stream*); *iii)* message size; and *iv)* identification of the message sender.

The bidirectional ring-based NoC infrastructure itself was devised to deploy a very efficient and low-profile interconnection. Hence, each node routes the

Table 1. Resource usage of the Morphable Communication Infrastructure

	Available Resources	Baseline Cache Ctrl.	In-Cache Stream Ctrl.	Main Memory Stream Ctrl.	Ring Node
Slices	75,900	1896	2370	852	155
LUTs	303,600	3602	4367	1666	297
Registers	607,200	365	1176	991	164
BRAM	3,090	0	0	2	2
Max. Freq. [MHz]	-	238	210	232	278

incoming messages to/from its two adjacent nodes (right and left) and to/from its connected component. To overcome the contention caused by simultaneously arriving packets, a simple round-robin priority function was devised that rotates the priority between channels upon the completion of a message transmission.

4 Experimental Evaluation

To validate the proposed infrastructure, a complete prototype was implemented in a Xilinx VC707 board, equipped with a XC7VX485T Virtex-7 FPGA and a 1GB DDR3 SODIMM 800MHz/1600Mbps memory module. The proposed infrastructure was evaluated against a conventional cache-based system, using three representative benchmarks from the computational algebra, image processing and bioinformatics domains. For such purpose, both computing infrastructures are composed of multiple PEs, each one comprising an adapted MB-LITE [12] processor, a private scratchpad for program data, and a memory-mapped interface to the proposed in-cache stream controller.

To guarantee a fair and realistic comparison, the cache configuration of the baseline system was made identical to a typical ARM Cortex A7 configuration. Hence, each PE is associated with a 8KB 4-way set-associative cache memory with 64-Byte cache lines. According to the considered cache line size, each message of the proposed communication protocol is composed of (at most) 16 32-bit data words plus the header and the address fields, totaling an 18-word message.

4.1 Hardware Resource Overhead

The FPGA implementation results are presented in Table 1. Despite the added versatility of the offered streaming capabilities, the results obtained for the devised in-cache stream controller represent a very low increase of the hardware resources, with an impact as small as 28 MHz in the maximum operating frequency. In fact, each of the devised components requires less than 2% of the FPGA resources. Moreover, due to the inherent scalability of the adopted ring-based NoC interconnection, it can be efficiently used to support a very large number of processing elements, being the only limiting factor the increased communication latency between nodes. The presented BRAM utilization refers to the buffering structures that are present at each component, except for the in-cache stream controller where they are implemented with registers.

4.2 Performance Evaluation

To evaluate and demonstrate the data-transfer and communication capabilities of the proposed infrastructure, three different benchmarks were considered.

- A Block-based Matrix Multiplication kernel that performs the $C=C+AB$ operation, where A, B and C are 128×128 matrices, divided in 8×8 sub-blocks, in order to maximize the cache usage. Since the matrices do not entirely fit in the cache memory, each row of matrix A is fetched once from memory (and maintained in the cache memory for as long as it is required), while matrix B is fetched once for each sub-block of matrix C.
- A Biological Sequence Alignment application that performs the computation of the alignment score between a reference and several query sequences (all randomly generated with a size of 1024 symbols). Two steps are considered, namely: (i) a pre-processing stage, where sequence data is reorganized to generate a query profile; and (ii) the computation of the alignment score matrix, by using the algorithm proposed in [13].
- A Histogram Equalization application to enhance the contrast by adjusting the intensities of a 256×256 pixels image. Two steps are required: (i) computation of the 8-bit image intensity histogram and corresponding cumulative distribution function (CDF), and (ii) scaling of the image intensities according to the obtained CDF. The first step is applied by evenly distributing the original image to the different PEs, such that multiple partial histograms are firstly obtained and then reduced and accumulated in a single PE, in order to generate the CDF. In the second step, each PE reads the CDF and applies the image intensity scaling to an individual block of the original image.

The first benchmark highlights the prefetching and broadcasting capabilities of the proposed system, the second one illustrates the proposed system capabilities when dealing with complex memory access patterns and data reorganization and the third demonstrates the advantages of deploying a morphable communication infrastructure that can adapt itself to the requirements of a running application. The obtained results for the three evaluation benchmarks are depicted in the graphs of Fig. 5, by considering a variable number of PEs.

In particular, the bar plots present the data transfer clock cycle reduction attained by the proposed framework due to the offered streaming and broadcasting capabilities. As it can be observed, the proposed infrastructure provides a significant reduction of the data transfer overheads in all benchmarks, which results from an efficient data prefetching and reutilization, allowing not only a mitigation of the shared memory latency, but also a reduction of the total number of memory accesses, therefore decreasing the contention in the shared interconnections. Naturally, these offered advantages are directly reflected in the resulting performance, as presented in the line plots, representing: the system performance scalability (n-PEs *vs* 1-PE) when relying on traditional pure cache-based approaches (orange); the system performance scalability (n-PEs *vs* 1-PE) when relying on the proposed morphable infrastructure (blue); and the

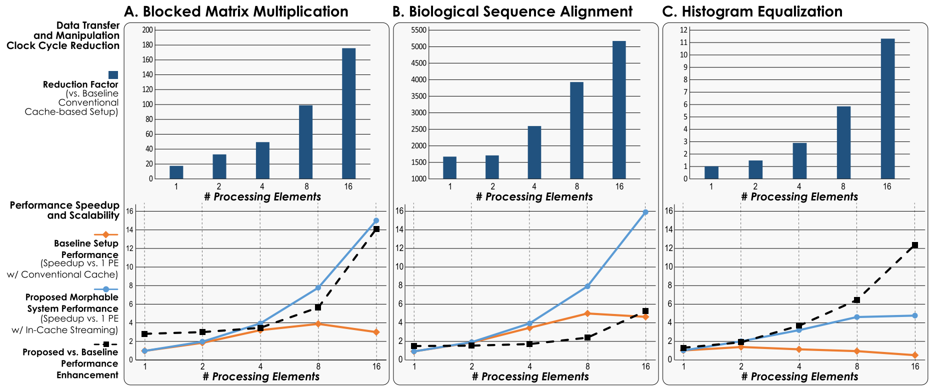

Fig. 5. Comparison of the proposed morphable infrastructure with the considered baseline conventional cache-based system, in what concerns data transfer and manipulation latency (top graphs) and performance scalability (bottom graphs).

speedup offered by an n-PE processing system using the proposed morphable infrastructure, regarding a traditional n-PE based system (black).

A careful analysis of the presented results evidences a poor scalability of the conventional cache-based system (orange), which even leads to a performance degradation when a higher number of PEs is used. On the other hand, the proposed morphable infrastructure is characterized by data transfer overheads that are mostly mitigated by its prefetching capabilities, partially aided by the broadcast capabilities of the supporting ring interconnection. As a result, a performance speedup of up to 15.03x, 15.9x and 4.7x is observed in the block matrix multiplication (Fig. 5.A), biological sequence alignment (Fig. 5.B) and histogram equalization (Fig. 5.C) benchmarks, respectively, with a 16-PE configuration.

5 Conclusion

A novel in-cache streaming architecture for many-core systems was proposed. Depending of the PE data request, the devised controller is able to deploy both conventional memory-addressing and stream-based communication paradigms and offers a rather convenient set of streaming capabilities, such as prefetching, complex memory access generation and stream manipulation, supporting a seamlessly switching between these communication paradigms without any significant impact in the data-transfer performance. The underlying communication is supported on a ring-based NoC interconnection, able to deploy a low-contention and broadcast-capable communication through a very low resource and scalable structure. When compared to a baseline conventional cache, with system configurations of up to 16 PEs, the obtained results show performance increases of up to 14x for a block matrix multiplication application, 5x for a biological sequence alignment algorithm and 12x for an histogram equalization kernel.

Acknowledgment

This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) under project UID/CEC/50021/2013 and research grant SFRH/BD/100697/2014.

References

[1] Y. Guo, P. Narayanan, M. A. Bennaser, S. Chheda, and C. A. Moritz, "Energy-efficient hardware data prefetching," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 19, no. 2, pp. 250–263, 2011.

[2] A. Jain and C. Lin, "Linearizing irregular memory accesses for improved correlated prefetching," in *IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*. ACM, 2013, pp. 247–259.

[3] T. Hussain, M. Shafiq, M. Pericàs, N. Navarro, and E. Ayguadé, "PPMC: A Programmable Pattern Based Memory Controller," in *Reconfigurable Computing: Architectures, Tools and Applications*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7199, pp. 89–101.

[4] N. Neves, P. Tomás, and N. Roma, "Efficient data-stream management for shared-memory many-core systems," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2015, pp. 508–515.

[5] T. Chau, X. Niu, A. Eele, W. Luk, P. Cheung, and J. Maciejowski, "Heterogeneous reconfigurable system for adaptive particle filters in real-time applications," in *Reconfigurable Computing: Architectures, Tools and Applications*, ser. Lecture Notes in Computer Science. Springer, 2013, vol. 7806, pp. 1–12.

[6] M. Modarressi, A. Tavakkol, and H. Sarbazi-Azad, "Application-aware topology reconfiguration for on-chip networks," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 19, no. 11, pp. 2010–2022, 2011.

[7] R. Pal, K. Paul, and S. Prasad, "Rekonf: A reconfigurable adaptive manycore architecture," in *IEEE Int. Symposium on Parallel and Distributed Processing with Applications (ISPA)*, 2012, pp. 182–191.

[8] K. T. Sundararajan, T. M. Jones, and N. P. Topham, "The smart cache: An energy-efficient cache architecture through dynamic adaptation," *International Journal of Parallel Programming*, vol. 41, no. 2, pp. 305–330, 2013.

[9] Y.-T. Chen, J. Cong, H. Huang, B. Liu, C. Liu, M. Potkonjak, and G. Reinman, "Dynamically reconfigurable hybrid cache: An energy-efficient last-level cache design," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*. IEEE, 2012, pp. 45–50.

[10] G. Kalokerinos, V. Papaefstathiou, G. Nikiforos, S. Kavadias, M. Katevenis, D. Pnevmatikatos, and X. Yang, "Fpga implementation of a configurable cache/scratchpad memory with virtualized user-level rdma capability," in *International Symposium on Systems, Architectures, Modeling, and Simulation, 2009 (SAMOS'09)*. IEEE, 2009, pp. 149–156.

[11] S. Ghosh, M. Martonosi *et al.*, "Cache miss equations: An analytical representation of cache misses," in *ACM International Conference on Supercomputing*. ACM Press, 1997, pp. 317–324.

[12] T. Kranenburg and R. van Leuken, "MB-LITE: A robust, light-weight soft-core implementation of the MicroBlaze architecture," *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp. 997–1000, March 2010.

[13] M. Farrar, "Striped Smith-Waterman speeds database searches six times over other SIMD implementations," *Bioinformatics*, vol. 23, no. 2, p. 156, 2007.