Data Criticality in Multi-Threaded Applications: An Insight for Many-Core Systems

Abhijit Das, *Student Member, IEEE,* John Jose, *Member, IEEE,* and Prabhat Mishra, *Fellow, IEEE*

***Abstract*—Multi-threaded applications are capable of exploiting the full potential of many-core systems. However, Network-on-Chip (NoC) based inter-core communication in many-core systems is responsible for 60-75% of the miss latency experienced by multi-threaded applications. Delay in the arrival of critical data at the requesting core severely hampers performance. This brief presents some interesting insights about how critical data is requested from the memory by multi-threaded applications. Then it investigates the cause of delay in NoC and how it affects the performance. Finally, this brief shows how NoC-aware memory access optimisations can significantly improve performance. Our experimental evaluation considers *Early Restart* memory access optimisation and demonstrates that by exploiting available NoC resources, critical data can be prioritised to reduce miss penalty by 11% and improve overall system performance by 9%.**

***Index Terms*—Data criticality, multi-threaded applications, many-core systems, network-on-chip (NoC), miss penalty.**

I. Introduction

APPLICATIONS running in any computing device can be classified as either multi-programmed or multi-threaded. Many-core systems have made way for applications with massive processing requirements, something which was not possible earlier. The processing power of many-core systems come from a collection of relatively simpler processing cores, unlike a single powerful core in uni-core systems. Hence, to exploit the full potential of many-core systems, applications need to be parallel, thus multi-threaded. There are promising approaches for automatic program parallelisation [1] and many-core aware mapping of multi-threaded applications [2]. Modern many-core systems employ Network-on-Chip (NoC) based inter-core communication, and it is reported that NoC is responsible for 60–75% of the miss latency in multi-threaded applications [3]. Delay in arrival of the critical data at the requesting core hampers performance of such applications. Hence, it is very important to get an insight into how multi-threaded applications request critical data and how NoC contributes to the miss latency (penalty) of such applications.

While running an application, a core usually requests for a single word from memory, called *critical word* [4]. The critical word is first searched in the cache memory and returned to the core if found. However, if the word is not found in the cache, it is requested from the next level of memory. The smallest unit of data transfer between different levels of memory is in blocks, containing multiple words. So, even though the core requests a single word, a complete block (containing the critical word) is brought from the next level of memory in the form of a packet through the underlying NoC. Nevertheless, data transfer bandwidth in NoC is limited to channel width called *flit*. A data block (packet) is divided into multiple flits (flit << block) and sent in sequence, as shown in Figure 1. The critical word can be in any of the incoming flits and accordingly impact the performance. On their way, flits experience indefinite router delays due to congestion, which impact their arrival on the requesting core, again hampering the performance. Due to the usage of relatively simpler processing cores, modern many-core systems cannot hide the miss penalty by out-of-order (OoO) execution and memory level parallelism (MLP) beyond a point, and thus get stalled.

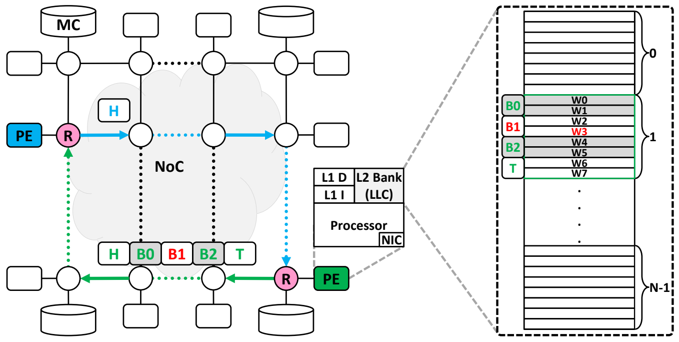

Figure 1: Conceptual view of an NoC based many-core system. Due to limited transfer bandwidth, a packet in NoC is divided into multiple smaller units called flits. A request packet consists of a single head flit (H), whereas a reply packet consists of a head flit followed by multiple body flits and ended with a tail flit (H, B0, B1, B2, T). Flits of a particular packet always travel in order. A critical word (W3) and the flit carrying that critical word (B1) are shown in *red*. The request and reply paths are shown in *blue* and *green*, respectively.

The purpose of this brief is to share some interesting insights about running multi-threaded applications in many-core systems. It specifically presents the pattern of critical words requested from the memory. Then it describes the pattern of delays experienced by the flits while travelling from source to their destination. The brief finally shows how using these insights can significantly improve the performance of existing optimisations on miss penalty reduction. Specifically, this brief makes the following major contributions:

1) In a unique profiling, it presents the position of critical

Manuscript received Month 00, 0000; revised Month 00, 0000; accepted Month 00, 0000. Date of publication Month 00, 0000; date of current version Month 00, 0000. *(Corresponding author: Abhijit Das.)*

A. Das and J. Jose are with the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, Assam 781039, India (e-mail: abhijit.das@iitg.ac.in; johnjose@iitg.ac.in).

P. Mishra is with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611, USA.

This work was partially supported by the NSF grant SaTC-1936040.

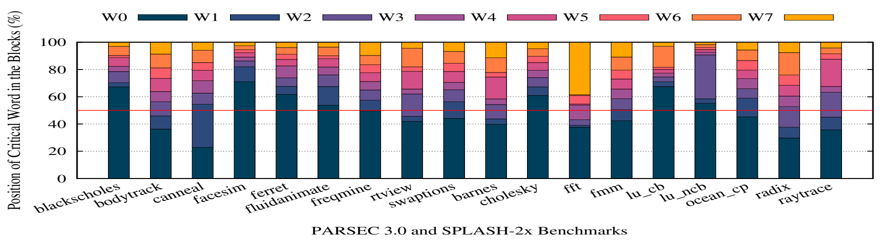

Figure 2: Position of critical word in the requested blocks

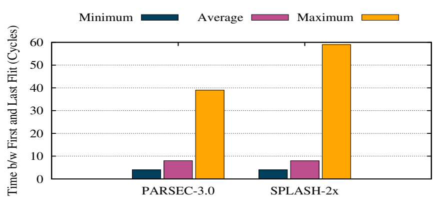

Figure 3: Reply Difference Time

word within the requested block and corresponding flits for PARSEC 3.0 [5] and SPLASH-2x [6] benchmarks.

2) It describes the difference in arrival times of the first and last flit of an incoming data block and how it impacts the performance of the underlying applications.
3) It proposes a modified version of the popular *Early Restart* optimisation [4] by considering the insights from (1) and (2), which performs better than the original.

II. Data Criticality in Applications

In a multi-threaded application, multiple tasks (threads) may run concurrently and independently by using the shared resources. This is the reason why multi-threaded applications utilise the resources of many-core systems better. Most of the modern many-core systems have private L1 caches and a shared L2 cache, which is divided into multiple banks and distributed across all the cores [7][8][9], as shown in Figure 1. When the critical word is not found in the L1 cache (miss), the core requests the word from the corresponding L2 cache bank. The entire block containing the critical word is transferred from L2 to L1 cache (refer Figure 1). In a conventional system, even though the core requires only the critical word to resume its execution, it is made to wait till the arrival of the entire block. We know that a block contains multiple words, so imagine a scenario where the very first word of the block is the critical word. The core could resume its execution after the arrival of the first word, but instead, it needs to wait till the last word of the block. Hence we are motivated to profile state-of-the-art multi-threaded applications to know the pattern in which critical words are spread in their requested blocks. This insight will help us to understand how popular memory access optimisations like *Early Restart* and *Critical Word First* [4] can benefit the corresponding applications (Section III).

We profile PARSEC 3.0 [5] and SPLASH-2x [6], the two most popular suite of multi-threaded benchmarks. We model an equivalent implementation of Intel Xeon Phi Processor 7235 [10], one of the latest many-core systems, on gem5 simulator [11]. Our system configuration is given in Table 1 for reference. We profile those data requests for whom the critical word was not found in the L1 cache. These are the requests that travel through the underlying NoC to reach L2 cache bank and get data, thus suffers NoC related delay [3]. To the best of our knowledge, this critical word based profiling is a first of its kind for any multi-threaded applications. Figure 2 shows the average position of critical word in the corresponding blocks requested from L2 cache. For example, during the entire run of *blackcholes* benchmark from PARSEC 3.0 suite, for 67.20% of the time, the first word (W0) is the critical word, for 3.17% of the time, the second word (W1) is the critical word and so on. There is a very interesting trend: *the first word is the critical word for most of the requested blocks*. This pattern is observed for the majority of the benchmarks of both PARSEC 3.0 and SPLASH-2x suites even though they are from diverse domains, including physics, finance, etc. However, the trend is unusual but not unreasonable, as the existing literature has proof of critical word regularity [12][13]. Literature states that it is reasonable to expect that data in a given region may be accessed in similar order on multiple occasions.

Table 1: System configuration

Processor	64 OoO x86 cores, 1.3GHz
L1 Cache	32KB, 8-way, private, split
L2 Cache (LLC)	512KB$\times$64 cores, 16-way, shared
Memory Bank	4; one located at each corner
NoC	8$\times$8 2D-Mesh topology, 128-bit channel 3 Virtual Networks (VNs), VN0, VN1, VN2 3 Virtual Channels (VCs) per VN 1-flit depth control VC, 4-flit depth data VC
Routing	2-stage routers (1.54ns), XY-DOR algorithm VC based wormhole packet-switching
Packets	1-flit for control packet, 5-flit for data packet
Word/Flit/Block	64-bit/128-bit/64B; 2-words/flit, 8-words/block
Benchmarks	PARSEC 3.0 and SPLASH-2x (multi-threaded)

While explaining the individual memory access patterns for each of the benchmarks is beyond the scope of this brief, we give some common characteristics that justify the pattern on the location of a critical word. Benchmarks that traverse through data arrays exhibit critical words near the beginning of the data block, most often to word 0 (W0). Also, the benchmarks having strided access with the smallest stride length of 0 have W0 as the critical word. Nevertheless, there are also benchmarks like *canneal* and *radix*, where the critical word is somewhat uniformly distributed. Benchmarks whose memory accesses are generated due to pointer chasing exhibits better distribution of the critical word. Based on these observations, specific memory access optimisation can be implemented for a class of applications exhibiting a specific behaviour to reduce the miss penalty of the critical word.

III. Criticality Aware Many-Core Systems

Two of the most popular memory access optimisations to reduce miss penalty of the critical word in modern many-core systems are *Early Restart* and *Critical Word First* [4]. In *Early Restart*, as soon as the critical word is received at the L1 cache, it is forwarded to the processor to resume its execution without waiting for the entire block. In *Critical Word First*, the critical word is forwarded out of order by the L2 cache to be received as the first word in the L1 cache to resume processor execution at the earliest. In NoC based many-core systems, everything

Table 2: Position of critical word in the corresponding flits. Note that head flit (H) does not carry any words of the data block and hence its corresponding column is left empty.

#	Suite	Benchmark	Flits of an incoming requested block				
			H	B0	B1	B2	T
1	PARSEC 3.0	blackscholes		70.37	11.94	7.95	9.74
2		bodytrack		45.90	17.91	17.38	18.81
3		canneal		54.49	17.20	13.38	14.93
4		facesim		82.06	7.06	5.40	5.48
5		ferret		67.60	15.04	8.56	8.80
6		fluidanimate		67.47	14.34	8.23	9.96
7		freqmine		57.50	13.67	12.32	16.51
8		rtview		45.59	20.08	16.26	18.07
9		swaptions		56.33	14.21	14.04	15.42
10	SPLASH-2X	barnes		43.70	14.69	19.40	22.21
11		cholesky		67.19	12.27	10.26	10.28
12		fft		39.10	14.69	7.14	39.07
13		fmm		50.46	15.19	13.94	20.41
14		lu_cb		70.99	6.37	4.32	18.32
15		lu_ncb		58.41	34.25	3.70	3.64
16		ocean_cp		58.96	14.35	13.27	13.42
17		radix		37.68	22.81	15.47	24.04
18		raytrace		45.03	22.51	24.03	8.43
Average				56.60	16.03	11.95	15.42

travel as packets (including data blocks), which are further divided into multiple flits. A head flit (H) carries the packet (message) header containing the routing information and does not carry any part of the data block. Multiple body flits (Bi), ended with a tail flit (T) carries the data block from source to the destination. Hence, a data block (of 8-words, refer Table 1) is transferred as a sequence of head flit, followed by three body flits (of 2-words each) and a tail flit (of 2-words) (H, B0, B1, B2, T). Table 2 presents the percentage of critical words that fall on different flits of a requested data block.

To understand the observation in Table 2, when a requested data block in *blackscholes* benchmark is transferred through flits, 70.37% of the time, critical words are in flit B0, 11.94% in flit B1, 7.95% in flit B2 and 9.74% in the tail flit T. It can be seen that the first body flit (B0) contains the critical words most of the time. It was evident from the fact that most of the critical words are the first word of a data block (refer Figure 2), and B0 carries the first two words of the block. Hence, *Early Restart* can be very effective in these kinds of applications. *Critical Word First* involves sending the critical word in the first flit by allowing out of order travel. Since by their very nature, the majority of the applications have their critical word in the first flit itself, *Critical Word First* might not be required. Avoiding *Critical Word First* also brings in the advantage of avoiding the complexity of sending the critical word out of order and then reordering the words at their destination.

Both *Early Restart* and *Critical Word First* are oblivious of the underlying on-chip communication infrastructure. These optimisations were introduced in the era of bus based on-chip communication, where a data block is transferred as a continuous stream of words. So, the block is transferred within a fixed time, and the core could resume execution at the earliest as per the optimisation. However, modern many-core systems use NoC based on-chip communication where the data block is transferred as multiple flits in a discrete fashion. On their way to the destination, flits experience indefinite router delay due to on-chip congestion. Hence, the effectiveness of *Early Restart* and *Critical Word First* reduces in many-core systems.

To understand about the delay experienced by incoming flits, we conduct an experimental analysis for all the PARSEC 3.0 and SPLASH-2x benchmarks, as given in Figure 3. We consider a metric called *Reply Difference Time (RDT)* [14], which calculates the difference between the arrival of the first flit and last flit of a data block in the requesting core. The minimum RDT remains 4 for both PARSEC 3.0 and SPLASH-2x benchmarks. It implies that all the flits reach back to back without any delay (ideal case). However, the maximum RDT is surprisingly high (during congestion): 39 for PARSEC 3.0 and 59 for SPLASH-2x benchmarks. Even the average RDT is 8.01 and 7.98, more than the minimum RDT meaning, flits are generally getting delayed. Any of the flits (including the one carrying the critical word) may be indefinitely delayed and hamper the performance. If memory access optimisations are made aware of the pros and cons of the underlying on-chip communication infrastructure, they may yield even better benefits. The purpose of this brief is not to provide NoC-aware implementations of all the existing critical word based memory access optimisations. Instead, we take just one of the most popular memory access optimisations, *Early Restart*, and demonstrate that an NoC-aware *Early Restart* is effective in significantly improving the overall system performance.

IV. NoC-Aware Early Restart Optimisation

This section proposes an NoC-aware *Early Restart* optimisation in many-core systems based on the observations from Figure 2, Table 2 and Figure 3. For the ease of illustration, we call the original *Early Restart* optimisation as **ER** and our proposed version of NoC aware *Early Restart* as **ER-NoC**.

A. *Critical Flit Identification*

When the core (processor) needs a critical word, it sends a request to the L1 cache controller (L1 CTLR), giving the address of the data block which contains the word. L1 CTLR maps into the corresponding set using the set index bits of the requested address, as shown in Figure 4. After the set is identified, L1 CTLR checks the tag bits for hit/miss detection. The corresponding data lookup is also done in parallel to reduce memory access latency. If the tag checker returns true (cache hit), the requested block is present in the L1 cache. L1 CTLR identifies the critical word using the offset bits of the address and sends it to the processor. If the tag checker returns false (cache miss), the block needs to be brought from the corresponding L2 cache bank through the underlying NoC. While the block is being fetched, ER optimisation uses the offset bits to check for the arrival of the critical word. As soon as the critical word is fetched at L1 cache, it is sent to the processor to resume its execution, without waiting for the arrival of the entire block. While the execution continues, remaining words of the block are fetched in the background. Since data transfer is still performed at block-level granularity, the underlying cache coherence remains unaffected.

We define *critical flit* as the flit carrying the critical word of the block through the underlying NoC. The proposed ER-NoC

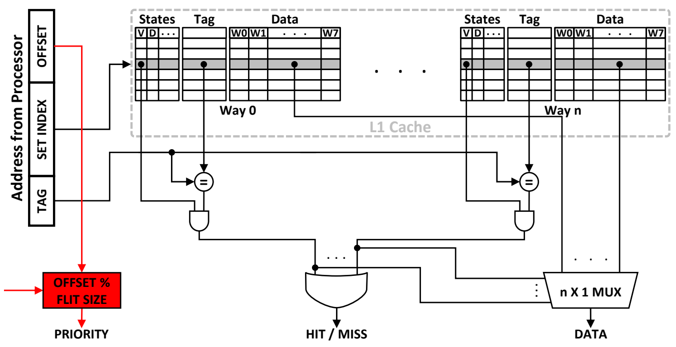

Figure 4: Modified L1 cache controller

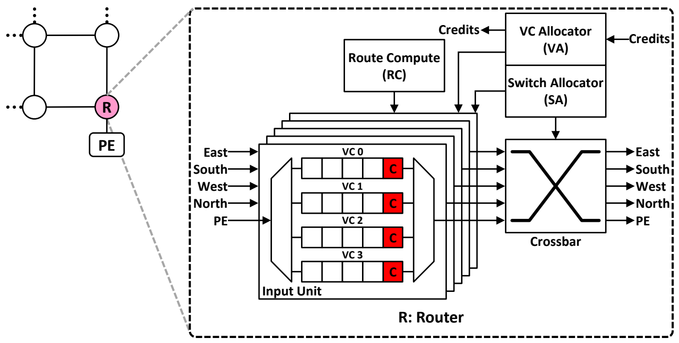

Figure 5: Modified router microarchitecture

optimisation adds a tiny module at L1 CTLR to identify the critical flit, as shown in *red* in Figure 4. This module takes as input the offset bits and flit size to return a 2-bit value (00/01/10/11) called *critical flit identifier (CFI)*. If the CFI value is 00, it means that the critical word will be transferred in the body flit B0, if 01 then B1, if 10 then B2 and if 11 then in the tail flit T. The proposed module added in L1 CTLR to get CFI runs in parallel with tag checker and data lookup modules and hence does not incur any additional delay in memory access latency. If the tag checker returns true, the CFI value is ignored as the requested block is already in L1 cache, and there will be no flit transfer. However, if the tag checker returns false, the CFI value is added to the message/packet header when the block request is sent to the L2 cache bank.

B. Critical Flit Prioritisation

Upon receiving the request, the L2 cache bank controller (L2 CTLR) replies with the data block as multiple flits through the underlying NoC. For the proposed ER-NoC optimisation, L2 CTLR puts the corresponding CFI value back into the packet header (H) of the reply. When the head flit traverse through the NoC routers, the CFI values are stored in a counter C in each router, as shown in Figure 5. ER-NoC modifies the traditional round-robin based priority policy to use C for priority during routing and arbitration. The motive behind this move is to reduce the router delay for critical flits to reach the destination at the earliest. It is employed after learning about the experienced RDT due to on-chip network congestion (refer Figure 3). However, with priority, there is always a risk of starvation for lower priority flits. To minimise such a risk, our policy does not prioritise all the flits of a data block; rather, it just prioritises up to the critical flit. For example, if the critical flit is B1 for a data block, our proposed policy just prioritises B0 and B1. This is done as all the flits preceding the critical flit of the block should reach L1 cache at the earliest, then only the critical flit will reach. Once the critical flit is prioritised, the succeeding flits (B2 and T) can reach L1 cache at their own pace as they are not required to resume execution.

When two flits of two different data blocks compete for the same output port, one with the lower CFI counter C is prioritised. When a flit wins the arbitration, the corresponding C is decremented by 1. This way, when the critical flit reaches a router, the counter C becomes 0, meaning the highest priority. Hence the proposed policy prioritises only and until critical flit of a data block. Exploiting NoC infrastructure to prioritise based on data criticality reduces miss penalty and improves overall system performance. This brief does not claim that the proposed priority policy gives the lowest starvation and best performance. The purpose is to show that it is beneficial to make existing ER like memory access optimisations aware of the NoC infrastructure in NoC-based many-core systems.

C. Experimental Evaluation

We consider the following architectures for evaluation:

- **Baseline:** Without any optimisation.
- **ER:** Original *Early Restart* optimisation.
- **ER-NoC:** Proposed *Early Restart* optimisation.

All the architectures are modelled on event-driven gem5 simulator and the system configuration is already given in Table 1. We modify MOESI_CMP_directory protocol in Ruby inside gem5 to implement the proposed cache controllers. We modify GARNET inside gem5 to implement the proposed router microarchitecture. We run gem5 in full-system (FS) mode to collect the results. Each multi-threaded benchmark runs 64 threads in 64 different cores of the system. We use *sim-medium* [5] input set and report the results for the run of entire region-of-interest. We consider miss penalty and system speedup as the performance metrics for evaluation. All the results are normalised with respect to the Baseline architecture. **L1 Cache Miss Penalty:** For Baseline architecture, it is the number of cycles required to bring a requested data block (containing the critical word) in L1 cache. In case of ER and ER-NoC architectures, it is the number of cycles required to receive the critical word while the requested data block is brought in L1 cache. Miss penalty directly reflects the effectiveness of the proposed ER-NoC. Figure 6 shows the normalised L1 cache miss penalty for PARSEC 3.0 and SPLASH-2x benchmarks. Since Baseline resumes execution only after the entire data block is received, and average RDT is quite high (refer Figure 3), miss penalty is more. The experienced RDT is due to on-chip congestion, which also affects the effectiveness of the existing ER. By exploiting the insights about the position of critical word (Figure 2 and Table 2) and RDT due to congestion (Figure 3), our prioritisation scheme in the proposed ER-NoC significantly reduces miss penalty by 11%. Our prioritisation scheme is not optimal as we can see it performs poorly for *blackscholes* when compared to ER. For more than 70% of the time, critical words are in the first body flit B0 for *blackscholes* (refer Table 2). Also, it is one of the simplest benchmarks with a very

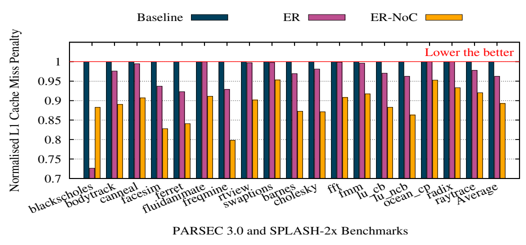

Figure 6: L1 Cache Miss Penalty

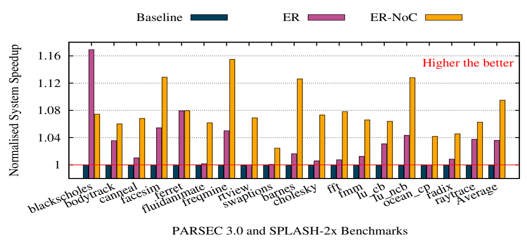

Figure 7: System Speedup

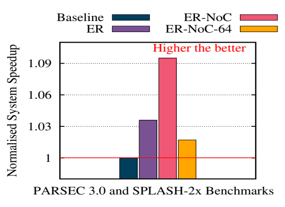

Figure 8: Sensitivity

small working set and negligible communication. Hence, the over-ambitious scheme to prioritise critical words during on-chip congestion is unnecessary. The presence of the scheme in ER-NoC delays routing and arbitration, thus performing poorly with respect to ER for *blackscholes*. However, the focus is not to propose an optimal prioritisation scheme, rather highlighting the optimisation opportunities by exploiting the insights.

System Speedup: System speedup (S) is given by, $S = \frac{ExecTime_{baseline}}{ExecTime_{proposed}}$, where $ExecTime_{baseline}$ and $ExecTime_{proposed}$ are the execution time of Baseline and proposed architectures, respectively. Figure 7 shows the normalised system speedup for PARSEC 3.0 and SPLASH-2x benchmarks. As expected, a reduction in miss penalty directly translates into the improvement of overall system performance. Our proposed ER-NoC architecture achieves a maximum and an average system speedup of 15% and 9%, respectively.

Sensitivity: One of the key NoC parameters that can have a significant impact on the proposed ER-NoC architecture is the *channel width (flit size)*. We perform a sensitivity analysis by changing the channel width in ER-NoC from 128-bit (refer Table 1) to 64-bit. The new architecture, *ER-NoC-64* has 9-flit data packets and hence should ideally take more time to transfer a block from L2 cache bank to L1 cache. Surprisingly, as shown in Figure 8, ER-NoC-64 is still able to beat the performance of Baseline that has only 5-flit data packets.

Overhead: McPAT [15] is run at 22nm processor technology to get the overheads due to the additional units. While the ER-NoC architecture has a negligible area and static (leakage) power overhead of 1.34% and 3.60%, respectively, dynamic power reduces by 5.85% due to performance improvement.

V. Related Works

There are specific works on efficient cache [16] and NoC [17] organisation to improve performance of many-core systems. The existence of critical word and memory access optimisations to prioritise critical words are available in the classic textbook by Hennessy and Patterson [4]. One of the first attempts to understand criticality for data requests from L1 to L2 cache is by Gieske [12]. He reported that for multi-programmed benchmark suites SPEC CPU 2000 and 2006, sufficient regularity in critical word exists. Exploiting this criticality information, quite a few optimisations are proposed in the recent past [13][18][19][20]. However, none of them explicitly studied the behaviour of critical words for multi-threaded benchmark suites like PARSEC 3.0 and SPLASH-2x.

VI. Conclusion and Future Work

In this brief, we shared two crucial insights about running multi-threaded applications in NoC based many-core systems. Taking an existing memory access optimisation as a case study, we demonstrated that an NoC-aware implementation could effectively utilise the two observed insights to significantly improve the overall system performance. In future, we will utilise the observed insights to design an optimal critical word based memory access optimisation. We also plan to consider recent AI and ML applications for analysis and comparison.

References

[1] U. Banerjee, R. Eigenmann, A. Nicolau, and D. A. Padua, "Automatic program parallelization," *Proceedings of the IEEE*, vol. 81, no. 2, pp. 211–243, 1993.
[2] B. K. Reddy, A. K. Singh, D. Biswas, G. V. Merrett, and B. M. Al-Hashimi, "Inter-cluster thread-to-core mapping and dvfs on heterogeneous multi-cores," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 3, pp. 369–382, 2017.
[3] D. Sanchez, G. Michelogiannakis, and C. Kozyrakis, "An analysis of on-chip interconnection networks for large-scale chip multiprocessors," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 7, no. 1, pp. 1–28, 2010.
[4] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.
[5] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *PACT*, 2008, pp. 72–81.
[6] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodological considerations," *ACM SIGARCH computer architecture news*, vol. 23, no. 2, pp. 24–36, 1995.
[7] A. Sodani, *et al.*, "Knights landing: Second-generation intel xeon phi product," *IEEE MICRO*, vol. 36, no. 2, pp. 34–46, 2016.
[8] J. Balkind *et al.*, "Openpiton: An open source manycore research framework," in *ASPLOS*, 2016, pp. 217–232.
[9] B. Daya *et al.*, "Scorpio: a 36-core research chip demonstrating snoopy coherence on a scalable mesh noc with in-network ordering," in *International Symposium on Computer Architecture (ISCA)*, 2014, pp. 25–36.
[10] (2017) Intel Xeon Phi Processor 7235. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/128694/intel-xeon-phi-processor-7235-16gb-1-3-ghz-64-core.html
[11] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.
[12] E. J. Gieske, "Critical words cache memory," Ph.D. dissertation, University of Cincinnati, 2008.
[13] N. Chatterjee *et al.*, "Leveraging heterogeneity in dram main memories to accelerate critical word access," in *MICRO*, 2012, pp. 13–24.
[14] A. Das *et al.*, "Critical packet prioritisation by slack-aware re-routing in on-chip networks," in *NOCS*, 2018, pp. 1–8.
[15] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, 2009, pp. 469–480.
[16] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler, "A nuca substrate for flexible cmp cache sharing," *IEEE transactions on parallel and distributed systems*, vol. 18, no. 8, pp. 1028–1040, 2007.
[17] Y. Xue and P. Bogdan, "User cooperation network coding approach for noc performance improvement," in *Proceedings of the 9th International Symposium on Networks-on-Chip (NOCS)*, 2015, pp. 1–8.
[18] B. P. Lilly, J. M. Kassoff, and H. Chen, "Critical word forwarding with adaptive prediction," Apr. 29 2014, uS Patent 8,713,277.
[19] C.-C. Huang and V. Nagarajan, "Increasing cache capacity via critical-words-only cache," in *ICCD*. IEEE, 2014, pp. 125–132.
[20] Z. Li, J. San Miguel, and N. E. Jerger, "The runahead network-on-chip," in *HPCA*. IEEE, 2016, pp. 333–344.