

# An efficient multi-level cache system for geometrically interconnected many-core chip multiprocessor

Tirumale Ramesh, Khalid Abed

Department of Electrical & Computer Engineering and Computer Science, Jackson State University, Mississippi, USA

## Article Info

### Article history:

Received Jul 30, 2021

Revised Sep 11, 2021

Accepted Jan 10, 2022

### Keywords:

Big data

Bus cache

Geometrical

Heterogeneous

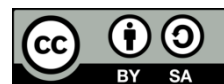
Many-core

Throughput

## ABSTRACT

Many-core chip multiprocessor offers high parallel processing power for big data analytics; however, they require efficient multi-level cache and interconnection to achieve high system throughput. Using on-chip first level L1 and second level L2 per core fast private caches is expensive for large number of cores. In this paper, for moderate number of cores from 16 to 64, we present a cost and performance efficient multi-level cache system with per core L1 and last level shared bus cache on each bus line of a cost-efficient geometrically bus-based interconnection. In our approach, we extracted cache hit and miss concurrencies and applied concurrent average memory access time to more accurately determine the cache system performance. We conducted least recently used cache policy-based simulation for cache system with L1, with L1/L2, and with L1/shared bus cache. Our simulation results show that an average system throughput improvement of 2.5x can be achieved by using system with L1/shared bus cache system compared to using only first level L1 or L1/L2. Further, we show that the throughput degradation for the proposed cache system is only within 5% for a single bus fault, suggesting a good bus fault tolerance.

*This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.*



## Corresponding Author:

Khalid Abed

Department of Electrical & Computer Engineering and Computer Science, Jackson State University

1400 John R Lynch Street (JSU Box 17098), Jackson, MS. 39217, USA

Email: khalid.h.abed@jsums.edu

## 1. INTRODUCTION

In recent years, many cores are trending as a on-chip computing platform [1]–[3] that can provide massive computational power for a heterogenous computing environment for big data [4] and other compute intensive embedded artificial intelligence applications [5]. Some recent work [6]–[9] on high performance computing for big data have focused on processing framework, architecture synthesis and utilization of multiple cores. With increased very large-scale integration (VLSI) density, it may be still manageable to provide heterogeneous computing using cost effective on-chip interconnection and cache memory system. From past research on bus-based interconnection for large parallel processing systems [10], it was determined that regular bus connected multiple-bus interconnection that uses number of buses equal to one-half of the cores or memory modules, gives comparable memory bandwidth. However, the reduced bus interconnection is costly for chip multiprocessor (CMP) due to large number of bus-core/memory connections. In our earlier research, we proposed a cost-effective interconnection using geometrical patterns for bus-core/memory connections [11] with reduced number of buses. The approach in [11] was extended to system level configuration defined with three geometrical system configurations termed as geometrical bus interconnection (GBI) [12] for bus-memory connections using rhombic connection pattern as the base. We achieved cost savings from 1.8x to 2.4x with GBI compared to regular reduced bus interconnection.

However, as the overall throughput of the many-core CMP is also determined by the cache system performance, achieving high overall CMP throughput with cost and performance efficient interconnection and cache system is highly desirable today.

Providing an adequate and sustained many-core CMP throughput becomes more challenging as it also requires efficient cache system solution. Towards this challenge, our focus is to present a cost-effective multi-level cache system to improve the overall many-core CMP throughput using comparable memory bandwidth results from cost-effective GBI [12]. A typical general multi-level cache system hierarchy for multi-core systems as shown in Figure 1 has L1 and L2 private cache per core at levels 1 and 2, and a shared cache L3 as a last level cache (LLC) at level 3. For example, some of the current mainstream commercial multi-core processor such as Intel® Core™ i5 processor has three levels of cache with per core L1 with a separate instruction and data cache, a per core L2 unified (instruction/data) cache and a shared L3 cache as LLC (shared by all cores).

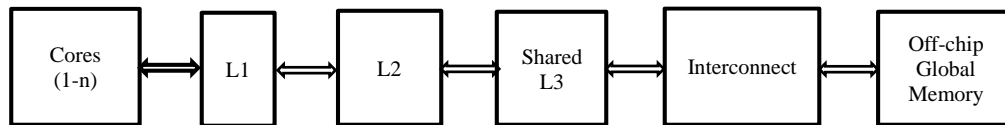


Figure 1. Traditional multi-level cache system with L1, L2 and L3 for multi-core CMP

Adding a large number of per core fast on-chip private L1 and L2 caches with a shared L3 may increase cache system cost. As a result, we propose an alternative solution by combining L1 with a relatively slower shared bus cache (SBC) as LLC added to every bus line of GBI [12] in which the data request of all cores is shared via GBI. In addition, our proposed cache system solution may also provide the ability to increase the cache levels and sizes within the cache hierarchy upon cache reconfiguration in order to optimize the system for cost, performance and power consumption.

Some earlier research [13]–[16] have addressed various cache system architecture, issues and solutions for improved performance. In [13], the authors addressed analyzing memory performance for tiled many-core CMP. Lin *et al.* [14] suggested hybrid cache systems that included layers for cache architecture from memory to data base to improve performance in specific relational data base query for big data applications. Charles *et al.* [15] looked at cache reconfiguration for network-on-chip (NoC) based many-core CMP. Safayenikoo *et al.* [16] suggested an energy-efficient cache architecture to address the problem of increased leakage power resulting from large area of LLC (as much as 50% of the chip area) due to its increased size. Most of the work reported in [13]–[16] may require complex cache design process. Our proposed cache system solution is simple and do not add any extra or difficult cache design process. Our main contribution in this paper are as: i) Propose a shared bus cache (SBC) within a multi-level cache system; ii) Present a least recently used (LRU) multi-level cache system simulation to extract hit and miss concurrencies; iii) Apply concurrent average memory access time (C-AMAT) [17] to accurately determine the system throughput performance and present our results; and iv) Provide conclusion and present some insight into future research.

## 2. L1-SBC CACHE SYSTEM

Figure 2 shows a system with L1 and share bus cache at every bus line of GBI [12]. We term the memory system using L1 private cache as L1, with L1 and L2 as L12, with L1 and shared bus cache as L1-SBC throughout this paper.

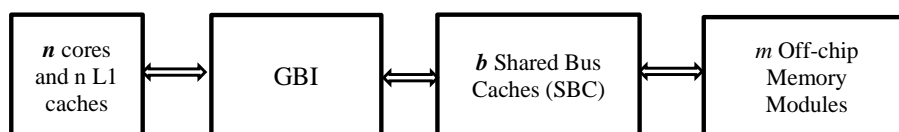


Figure 2. L1-SBC

### 2.1. Concurrent average memory access time (C-AMAT)

Some cache techniques [18]–[20] were suggested earlier for improving traditional average memory access time for multi-level cache systems. In [18], hardware prefetching was considered to exploit spatial and temporal locality of references. In [19], multi-level caches were considered as primary and secondary memories for proxy servers to access web content. In [20], an LRU replacement policy was proposed that makes use of the awareness of the cache miss-penalty to ensure memory access latency is balanced for memory system built with different memory technologies termed as “hybrid” system. The work addressed in [18]–[20] were specific cache techniques attempted to reduce average memory access time without considering any cost implications. Our approach is to optimize cache and interconnection cost across the cache levels and apply C-AMAT for exploitation of parallel concurrency in cache *hit* and *misses* that accurately determine the average memory access time across all levels for data access. An analytical method for determining C-AMAT is briefly provided below. A traditional average memory access time (AMAT) with a multi-level cache system is given in (1) and (2) for L1 and L12 cache systems respectively.

$$AMAT1 = t_1 h_1 + (1 - h_1) t_m \quad (1)$$

$$AMAT2 = t_1 h_1 + (1 - h_1)(t_2 h_2 + (1 - h_2) t_m) \quad (2)$$

Where  $t_1$  and  $t_2$  are the cache access time for level 1 and level 2 caches,  $h_1$  and  $h_2$  are cache hit ratios for level 1 and level 2 caches and  $t_m$  is the global memory access time. In our approach, we exploit parallel concurrency for core and SBC hit and miss concurrency for SBC supported by GBI and apply C-AMAT for performance evaluation. The hit concurrency will improve performance while a cache miss may impact the memory system performance, depending on hit concurrency. Taking advantage of multiple buses with miss concurrency, higher system performance can be achieved. However, the application of C-AMAT need to ensure that the miss concurrency do not exceed the interconnection bandwidth with reduced number of buse. Thus, we re-write (1) and (2) as (3) and (4).

$$C - AMAT1 = \frac{t_1 h_1}{ch_1} + (1 - h_1) t_m / cm \quad (3)$$

$$C - AMAT2 = \frac{t_1 h_1}{ch_1} + (1 - h_1) \left( \frac{t_2 h_2}{ch_2} + (1 - h_2) t_m / cm \right) \quad (4)$$

Where  $ch_1$  and  $ch_2$  are the average hit cycle concurrency at levels 1 and 2 and  $cm$  is the average miss cycle concurrency. In this paper, we evaluate L1, L12 and L1-SBC systems. We selected minimum number of L1 and SBC cache blocks to meet the following criterion for hit and miss concurrency given as (5).

$$c_h \leq n, c_m \leq n/2 \quad (5)$$

Since the GBI interconnection provides a memory bandwidth of  $n/2$ , we can also approximate (4) by miss concurrency supported by the GBI memory bandwidth as (6).

$$C - AMAT = \frac{t_1 h_1}{ch_1} + (1 - h_1) \left( \frac{t_2 h_2}{ch_2} + (1 - h_2) 2t_m / n \right) \quad (6)$$

When the  $cm$  is less than  $n/2$ , the interconnection bandwidth is not fully utilized. The C-AMAT given in (6) is smaller compared to conservative miss concurrency given in (4). The percentage deviation from (4) to (6) varies from 4 to 30 % across all cache systems. We see a higher deviation for L1-SBC system which is attributed to the fact that the miss concurrency decreases as a result of higher hit concurrency using bus cache during read cycle. In this paper, we only include conservative results from (3) and (4) for L1 and L12 cache systems respectively and at the same time ensuring criterion (5).

### 2.2. Geometrical bus interconnection (GBI) [12] cost

Table 1 gives the average normalized interconnection cost of GBI compared to fully reduced multiple bus system [10]. We notice a reduction of about 30 % in cost across the number of cores from 16 to 64.

Table 1. Normalized average GBI cost compared to fully reduced bus system [10]

No. of Cores	Normalized Cost
16	0.69
32	0.66
64	0.64

### 2.3. SBC impact on C-AMAT

In the past, some shared cache techniques [21] have looked at cache sharing of ways based on hash mapping instead of traditional cache set sharing for multi-core platforms. In general, it is known that by increasing the number of processor cores can directly increase LLC (last level cache) hit and miss concurrency giving reduced C-AMAT. As our system uses buses equal to one-half the number of cores, the memory access missed in per core cache is searched in SBC. Since a shared reduced number of buses in our approach naturally captures all core accesses via the bus interconnection, placing an SBC at each bus line of GBI replicates closely to a traditional L3 shared cache normally used in current commercial processor systems. As we used  $\frac{n}{2}$  number of SBC at level 2, any miss in L1 increases the hit concurrency in SBC. In our approach, we accounted only a pure miss concurrency [17] (only if none of the bus cache has a hit in the hit cycle, a miss is accounted).

### 2.4. Cache association impact on C-AMAT

Cache association can also impact our solution. Authors in [22], [23] attributed to the fact that higher cache association normally increases the cache hit rate but at the expense of hardware complexity for the cache controller and additional latency for cache search time with increased association. However, in our approach, the association was selected to ensure that criteria (5) are satisfied. Thus, selecting a direct mapped cache may benefit to achieve reduced C-AMAT. In general, miss concurrencies in LLC can normally be supported by use of multi-ported memory, or multi-bank memory (memory modules) with a single bus. However, for a single bus system, bus contention impacts the throughput performance. The miss concurrency can be facilitated by using a multi-bank memory module with multiple bus interconnection between shared cache and memory modules. The miss concurrency can be supported by multiple buses in GBI yielding lower C-AMAT.

## 3. CACHE SYSTEM SIMULATION

### 3.1. System operation with L1-SBC

Figure 2 shows the operation flowchart for read and write cycles for L1 system. SBC is used only during “read cycle” with a “write through” policy to update on cache miss. In “normal no-fault mode”, during read cycle, the data is first searched in L1. If the L1 read is a “miss,” it is then searched in SBC. If it is “hit,” the data is cached. On read “miss” in SBC, buses in GBI are arbitrated to utilize full memory bandwidth and the data is read from the global memory module and is written to SBC and L1 cache as well. If the current bus that is granted fails, then cache system switches to “bus fault mode” and the interconnection is re-arbitrated to use other  $b-1$  connected buses. After bus re-arbitration, the data is re-searched first in L1 and if “hit”, the data is cached in L1 cache, otherwise searched in SBC. During write cycle, if the L1 cache block is “present”, then data is written into L1 cache. On L1 write “miss”, the L1 cache block is replaced and the data is updated to L1 and consequently the data is written to global memory using arbitrated buses in GBI.

The proposed cache system was simulated using publicly available “lru-cache” libraries in *python* and created multiple objects of a “lru-cache” with indexing to implement L1, L2 and SBC. We iterated cache operation for over  $n \times 1000$  for  $n$  number of cores. Table 2 shows the general parameters used for the simulation. Using as much of insight into today’s memory technologies, we approximately used a relative bit cost for L1, L2 and SBC as given in Table 2.



Table 4. Cache hit and miss concurrency with 50% read requests

System size	16			32			64		
	ch1	ch2	cm	ch1	ch2	cm	ch1	ch2	cm
L1	0.4		8.3	0.4		16.1	0.5		31.9
L12	4.5	4.2	8	8.5	8.2	15.7	16.8	16.1	31.2
L1-SBC	4.4	5.2	7.4	8.5	11.9	12.8	16.7	31.5	16.1

Table 5. Cache hit and miss concurrency with 80% read requests

System size	16			32			64		
	ch1	ch2	cm	ch1	ch2	cm	ch1	ch2	cm
L1	0.4		8.3	0.4		16.1	0.5		31.9
L12	1.9	6.8	7.1	3.7	13.1	14.3	7.3	25.9	29.2
L1-SBC	2.0	8.2	6.4	3.7	19.3	10.6	6.9	50.9	16.1

Figures 5 and 6 show the hit and miss concurrency for L1-SBC for 50 % and 80 % *read* requests respectively. For the same number of cores, the miss concurrency decreases for L1-SBC as compared to L1 due to higher hit concurrency in SBC. The miss concurrency utilization in L1-SBC is about 50 % for larger number of cores. This is attributed to the fact that SBC offers higher hit concurrency yielding reduced memory traffic over the interconnection. Even though the low miss concurrency utilization may suggest that the number of buses for higher number of cores may be reduced further, it may invariably decrease the hit rate for SBC due to lower bandwidth availability thus nullifying any overall advantage. As the data read are more than data writes, SBC hit concurrency increases by approximately 1.5x for the same system size.

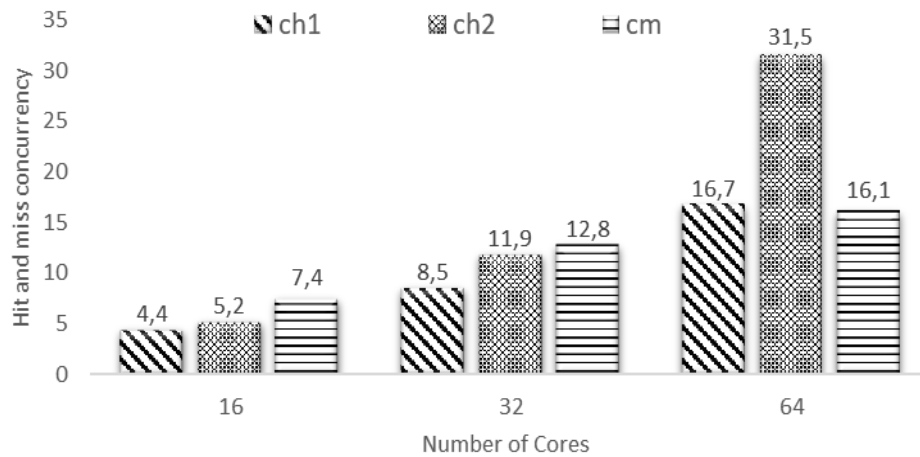


Figure 5. Cache hit and miss concurrency for L1 with 50 % read requests

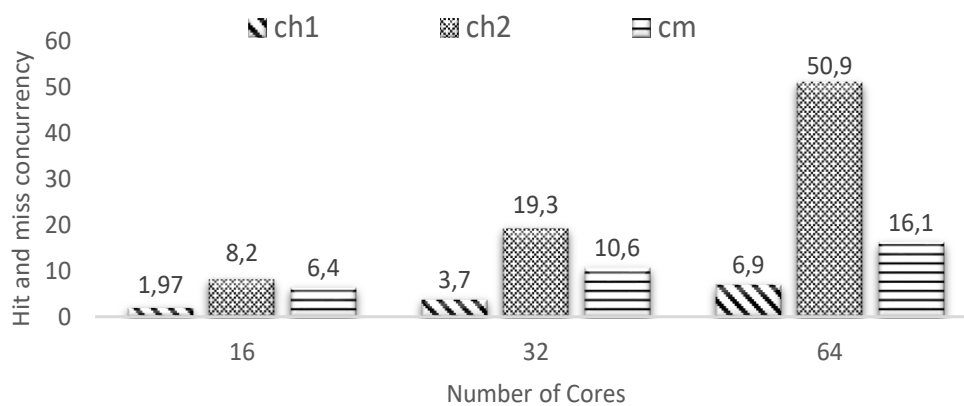


Figure 6. Cache hit and miss concurrency for L1 with 80 % read requests

### 3.5. Concurrent average memory access time (C-AMAT) cycles

We evaluated the concurrent average memory access time (C-AMAT) cycles from (3) and (4). Tables 6 and 7 show the C-AMAT for 50 % and 80 % read requests respectively.

Table 6. C-AMAT with 50 % read requests

Cache system	16	32	64
L1	12.4	6.4	3.2
L12	4.1	2.2	1.1
L1-SBC	3.7	1.6	0.3

Table 7. C-AMAT with 80 % read requests

Cache system	16	32	64
L1	12.4	6.4	3.2
L12	6.5	3.3	1.6
L1-SBC	5.7	2.4	0.3

As a result of increased SBC hit concurrency, the C-AMAT decreases with the number of cores. Figure 7 shows the C-AMAT for 50% and 80% *read* requests respectively. Further reduction in C-AMAT is seen for 80% *read* requests due to increase in SBC hit concurrency.

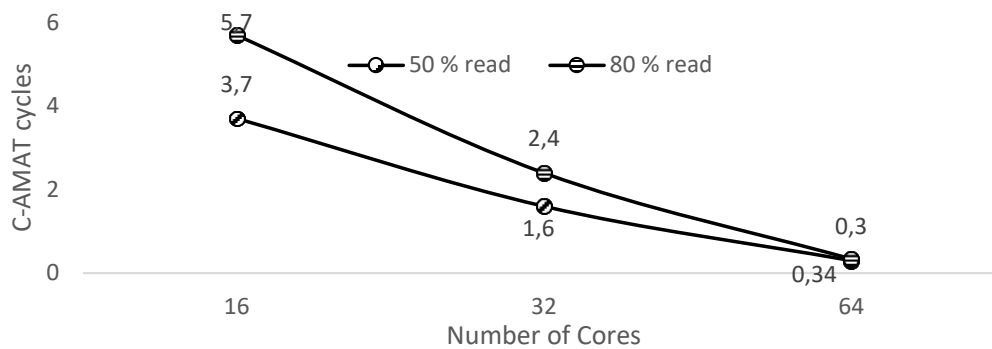


Figure 7. C-AMAT with 50 % and 80 % read data requests

### 3.6. Cache system through

The throughput in GB/sec ( $g$ ) given as (6).

$$g = \frac{2 \cdot b}{C-AMAT + tr} \quad (6)$$

Where  $b$  is the number of buses with 2 bytes bus data width. We assumed GBI bus arbitration and bus allocation reconfiguration time ( $tr$ ) of 1 cycle and a clock cycle time of 0.5 ns. Table 8 summarizes our results for throughput in GB per sec. We used normalized unit cost from Table 3 and C-AMAT using (3) and (4). As shown in Table 8, the throughput increases with the number of cores and read request percentage suggesting a good advantage.

Table 8. Throughput in GB/sec for L1-SBC for 50 % and 80 % read requests

		No. of Cores					
		16		32		64	
		50 %	80 %	50 %	80 %	50 %	80 %
		6.8	4.8	24.6	18.8	98.5	95.5

Figure 8 shows the throughput for 50% and 80% *read* requests respectively. Figure 9 shows the average throughput improvement factor for L12 and L1-SBC cache systems over L1 cache system. We found that the average throughput improvement factor of L12 cache system across all system sizes is 1.5x for 50 % *read* requests and 1.8x for 80 % *read* requests compared to L1. We determined that the average throughput improvement for L1-SBC memory system is 2.5x for 50 % *read* requests and 2.4x with 80 % *read* requests compared to L1 system. As there is very negligible cost increase for L1-SBC (0.5%) over L1, we conclude that L1-SBC cache is both cost and performance efficient compared to L1 or L12 cache system, L1-SBC offers 30 to 60% increase in throughput improvement factor compared to L12 improvement factor over L1.

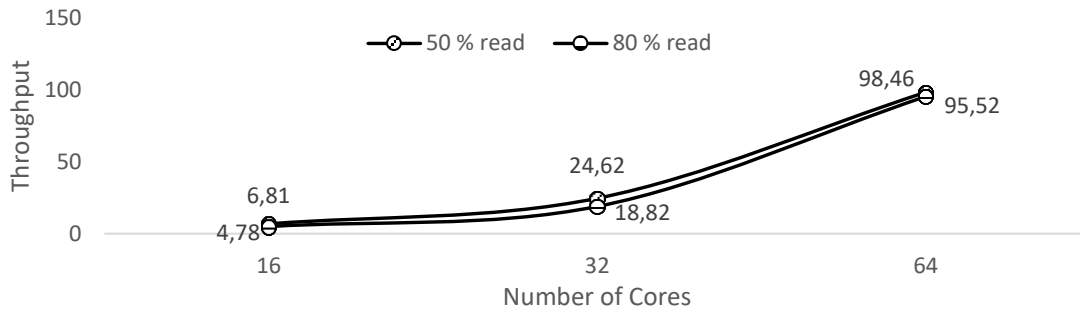


Figure 8. L1-SBC throughput in GB/sec for 50 % and 80 % read data requests

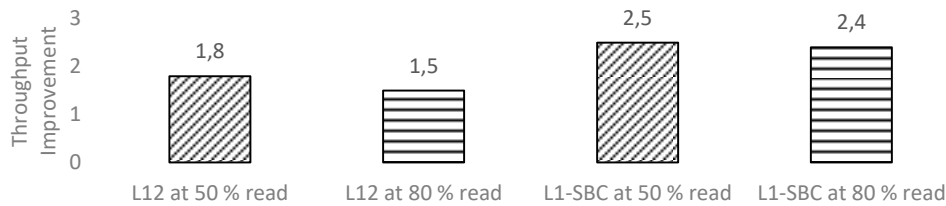


Figure 9. Average improvement in throughput for L12 and L1-SBC compared to L1

### 3.7. Cache system throughput with single bus fault

We also ran simulation for L1-SBC with a single bus fault in the system. We used both critical and non-critical bus for assigning faulty bus. A bus is a “critical bus” if a memory is only connected to that bus. Typically, rhombic interconnection [11] has a single “critical bus”. However, with GBI [12], we provided redundant bus paths yielding all buses “non-critical”. Figure 10 shows the percentage degradation of a single bus faulted system compared to normal L1-SBC system with 50% *read* requests. We noticed that the percentage degradation in throughput for a single bus fault is less than 5% across all system sizes and decreases with higher number of cores. This suggests a good fault tolerance for L1-SBC for increased number of cores.

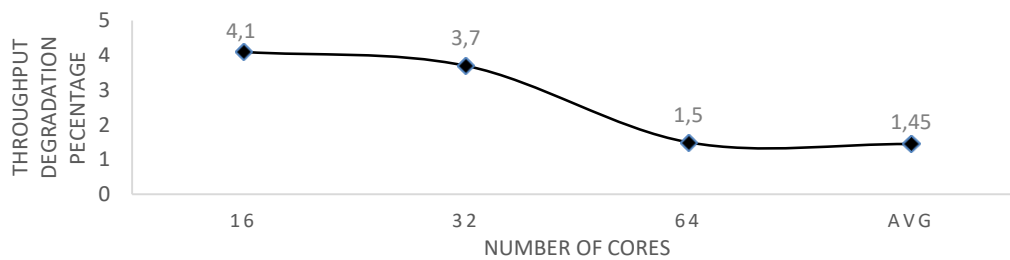


Figure 10. Throughput degradation with a single bus fault for 50 % read requests



#### 4. CONCLUSION AND FUTURE RESEARCH

Many-core based heterogeneous system demands high system throughput for big data applications and other compute intensive embedded applications. By adding a less expensive SBC in association with expensive per core L1 private cache within a multi-level cache hierarchy, we can achieve higher system throughput. For better accuracy, we extracted cache hit and miss concurrencies at each level and applied concurrent average memory access time for L1, L12 and L1-SBC systems. We conducted simulation of L1, L12 and L1-SBC cache systems. Our simulation results indicate that by using L1-SBC, we can achieve 2.5x throughput improvement compared to using only L1 private cache and we see that L1-SBC offers higher increase in throughput improvement factor compared to L12 improvement factor at a very negligible increase in SBC cost over L1. We also determined that the throughput degradation using L1-SBC with a single bus fault is less than 5 % across all system sizes and this degradation reduces as the system size increases suggesting a good advantage for higher number of cores. As we used the SBC only during read request, in the future, we hope to develop some additional novel SBC cache protocols using *exclusive* and *shared* modes and include SBC in both read and write cycles. We also hope to perform some heterogenous computing big data application benchmarks with LRU L1-SBC system and assess the overall system performance.

#### ACKNOWLEDGEMENTS

This work was supported in part by Army Research Office HBCU/MSI contract number W911NF-13-1-0133 entitled: "Exploring High Performance Heterogeneous Computing via Hardware/Software Co-Design".




#### REFERENCES

- [1] S. Le Beux, P. V. Gratz, and I. O'Connor, "Guest editorial: emerging technologies and architectures for manycore computing part 1: hardware techniques," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 2, pp. 97–98, Apr. 2018, doi: 10.1109/TMSCS.2018.2826758.
- [2] S. Savas, Z. Ul-Abdin, and T. Nordström, "A framework to generate domain-specific manycore architectures from dataflow programs," *Microprocessors and Microsystems*, vol. 72, p. 102908, Feb. 2020, doi: 10.1016/j.micpro.2019.102908.
- [3] J. Ax *et al.*, "CoreVA-MPSoC: a many-core architecture with tightly coupled shared and local data memories," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 5, pp. 1030–1043, May 2018, doi: 10.1109/TPDS.2017.2785799.
- [4] H. Homayoun, "Heterogeneous chip multiprocessor architectures for big data applications," in *Proceedings of the ACM International Conference on Computing Frontiers*, May 2016, pp. 400–405, doi: 10.1145/2903150.2908078.
- [5] A. Parashar, A. Abraham, D. Chaudhary, and V. N. Rajendiran, "Processor pipelining method for efficient deep neural network inference on embedded devices," in *Proceedings - 2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics, HiPC 2020*, Dec. 2020, pp. 82–90, doi: 10.1109/HiPC50609.2020.00022.
- [6] L. Cheng *et al.*, "A tensor processing framework for CPU-manycore heterogeneous systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2021, doi: 10.1109/tcad.2021.3103825.
- [7] M. Goudarzi, "Heterogeneous architectures for Big Data batch processing in MapReduce paradigm," *IEEE Transactions on Big Data*, vol. 5, no. 1, pp. 18–33, Mar. 2019, doi: 10.1109/TBDDATA.2017.2736557.
- [8] E. Alareqi, T. Ramesh, and K. Abed, "Functional heterogeneous processor affinity characterization to Big Data: towards machine learning approach," in *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, Dec. 2017, pp. 1432–1436, doi: 10.1109/CSCI.2017.250.
- [9] C. Lai, X. Shi, and M. Huang, "Efficient utilization of multi-core processors and many-core co-processors on supercomputer beacon for scalable geocomputation and geo-simulation over big earth data," *Big Earth Data*, vol. 2, no. 1, pp. 65–85, Jan. 2018, doi: 10.1080/20964471.2018.1434265.
- [10] T. N. Mudge, J. P. Hayes, G. D. Buzzard, and D. C. Winsor, "Analysis of multiple-bus interconnection networks," *Journal of Parallel and Distributed Computing*, vol. 3, no. 3, pp. 328–343, 1986, doi: 10.1016/0743-7315(86)90019-5.
- [11] T. Ramesh and K. Abed, "Reconfigurable many-core embedded computing platform with Geometrical bus interconnection," in *Proceedings - 2020 International Conference on Computational Science and Computational Intelligence, CSCI 2020*, Dec. 2020, pp. 1256–1259, doi: 10.1109/CSCI51800.2020.00234.
- [12] T. Ramesh and K. Abed, "Cost-efficient reconfigurable geometrical bus interconnection system for many-core platforms," *International Journal of Reconfigurable and Embedded Systems (IJRES)*, vol. 10, no. 2, pp. 77–89, Jul. 2021, doi: 10.11591/ijres.v10.i2.pp77-89.
- [13] Y. Liu, S. Kato, and M. Edahiro, "Analysis of Memory System of Tiled Many-Core Processors," *IEEE Access*, vol. 7, pp. 18964–18974, 2019, doi: 10.1109/ACCESS.2019.2895701.
- [14] Y. Te Lin, Y. H. Hsiao, F. P. Lin, and C. M. Wang, "A hybrid cache architecture of shared memory and meta-table used in big multimedia query," in *2016 IEEE/ACIS 15th International Conference on Computer and Information Science, ICIS 2016 - Proceedings*, Jun. 2016, pp. 1–6, doi: 10.1109/ICIS.2016.7550809.
- [15] S. Charles, A. Ahmed, U. Y. Ogras, and P. Mishra, "Efficient cache reconfiguration using machine learning in NoC-based many-core CMPs," *ACM Transactions on Design Automation of Electronic Systems*, 2019, doi: <https://doi.org/10.1145/1122445.1122456>.
- [16] P. Safayenikoo, A. Asad, and F. Mohammadi, "An Energy-Efficient Cache Architecture for Chip-Multiprocessors Based on Non-Uniformity Accesses," in *2018 IEEE Canadian Conference on Electrical & Computer Engineering (CCECE)*, May 2018, pp. 1–4, doi: 10.1109/CCECE.2018.8447736.
- [17] X. H. Sun and D. Wang, "Concurrent average memory access time," *Computer*, vol. 47, no. 5, pp. 74–80, May 2014, doi: 10.1109/MC.2013.227.
- [18] J. H. Lee, S. W. Jeong, S. D. Kim, and C. C. Weems, "An intelligent cache system with hardware prefetching for high performance," *IEEE Transactions on Computers*, vol. 52, no. 5, pp. 607–616, May 2003, doi: 10.1109/TC.2003.1197127.




- [19] Y. Niranjana, S. Tiwari, and R. Gupta, "Average memory access time reduction in multilevel cache of proxy server," in *Proceedings of the 2013 3rd IEEE International Advance Computing Conference, IACC 2013*, Feb. 2013, vol. 2013-Febru, pp. 44–47, doi: 10.1109/IAAdCC.2013.6506813.
- [20] D. Chen, H. Jin, X. Liao, H. Liu, R. Guo, and D. Liu, "MALRU: Miss-penalty aware LRU-based cache replacement for hybrid memory systems," in *Proceedings of the 2017 Design, Automation and Test in Europe, DATE 2017*, Mar. 2017, pp. 1086–1091, doi: 10.23919/DATE.2017.7927151.
- [21] A. K. Singh, K. Geetha, S. Vullala, and N. Ramasubramanian, "Efficient Utilization of Shared Caches in Multicore Architectures," *Arabian Journal for Science and Engineering*, vol. 41, no. 12, pp. 5169–5179, Dec. 2016, doi: 10.1007/s13369-016-2197-0.
- [22] M. D. Hill and A. J. Smith, "Evaluating Associativity in CPU Caches," *IEEE Transactions on Computers*, vol. 38, no. 12, pp. 1612–1630, 1989, doi: 10.1109/12.40842.
- [23] D. Ramtane, N. Singh, S. Kumar, and V. K. Patle, "Cache Associativity Analysis of Multicore Systems," in *2020 International Conference on Computer Science, Engineering and Applications, ICCSEA 2020*, Mar. 2020, pp. 1–4, doi: 10.1109/ICCSEA49143.2020.9132884.
- [24] S. Khan, A. R. Alameldeen, C. Wilkerson, O. Mutluy, and D. A. Jimenez, "Improving cache performance using read-write partitioning," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2014, pp. 452–463, doi: 10.1109/HPCA.2014.6835954.
- [25] S. Chandak *et al.*, "Improved read/write cost tradeoff in DNA-based data storage using LDPC codes," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2019*, Sep. 2019, pp. 147–156, doi: 10.1109/ALLERTON.2019.8919890.

## BIOGRAPHIES OF AUTHORS



**Tirumale Ramesh**    is currently supporting Jackson State University as an advanced computing research consultant where he previously served as a Senior Research Associate. His current research interests include heterogeneous computing, network-on-chip, cache systems, artificial intelligence (AI)/machine learning. He received his BE degree in electrical engineering from Bangalore University, India in 1975, an MSEE in VLSI area from Mississippi State University in 1983 and the Ph.D degree in computer engineering from Oakland University, Michigan in 1993. Ramesh has a long-standing career. Previously he served as a tenured professor of computer engineering at Saginaw Valley State University in Michigan. He was a corporate fellow for advanced computing at Boeing and provided technical leadership for several research projects funded by Boeing Corporate Research. He was a senior engineer at IBM. He also served as a professorial lecturer in the department of electrical and computer engineering at George Washington University in DC. Ramesh has numerous US and foreign patents and published widely. He is a senior member of IEEE and has served in leadership roles for IEEE conferences and IEEE computer society and received several professional awards. He can be contacted at email: rjfeb35@gmail.com.



**Khalid Abed**    is a Tenured Professor in the Department of Electrical & Computer Engineering and Computer Science at Jackson State University (JSU). His research interests include high performance heterogeneous/reconfigurable computing (HPRC/HPHC), edge computing, artificial intelligence (AI), machine learning (ML), and deep learning (DL). He received his B.S., M.S., and Ph.D. in Electrical Engineering from Wright State University in 1995, 1996, 2000, respectively. He has published extensively in IEEE journals and conferences and is a technical reviewer for several IEEE journals and conferences. Dr. Abed is a Senior Member of IEEE, IEEE Computer Society. He co-authored several patent submissions in the HPHC/HPRC, areas. He has received funding from sources including the NSF, the DoD, and the Army Research Office. He has received about \$3M in grants for HPHC/HPRC education and research. He can be contacted at email: khalid.abed@jsums.edu.