Profiling EEMBC MultiBench Programs in 64-core Machine

Chao Chen[1], Ajay Joshi[1], and Erno Salminen[2]

[1]Electrical and Computer Engineering Department, Boston University, 8 Saint Mary's Street, Boston, MA
[2]Department of Pervasive Computing, Tampere University of Technology, P.O. Box 553, FIN-33101 Tampere, Finland

Abstract

This paper presents the profiling of EEMBC MultiBench programs. We have executed 16 parallel benchmark workloads on M5 simulator. The simulated platform contains 64 dual-issue cores with 16+16KB private L1 caches and distributed 16x1MB L2 cache, running at 2 GHz. The L1-L2 bus runs at 1GHz and is 64 Bytes wide. The measured parameters included application performance as instruction-per-cycle (IPC), traffic on L1-L2 bus, and L1 cache miss penalties. Simulation was very time consuming (about 1 day of CPU time per run) and hence limited to 1 second of application runtime. Measurements varied both the number of parallel workloads and worker threads per workload. Performance peak occurs when there are as many threads as cores, i.e. 64. Running parallel workloads achieves higher performance than using multiple workers for small number of concurrent workloads. The measured IPC varied in the range $0.2 - 16.8$ and bandwidth $0.9 - 49$ GByte/s. The average IPC was surprisingly low, only about 2 instructions per cycle, whereas the average bandwidth in L1-L2 bus was 9.2 GByte/s.

Keywords: network-on-chip, benchmarking, parallel application, profiling, EEMBC MultiBench

CONTENTS

I. INTRODUCTION

This paper presents the profiling of parallel benchmark programs. Modern multiprocessor system-on-chip (MPSoC) includes tens of heterogeneous IP blocks, such as CPUs, memories, input/output devices, and HW accelerators, see for example [13], [24]. Figure 1 shows an example of our 64-core target system. Our target system has 64 cores and each core has private I-cache and D-cache. The cores can access a shared L2 cache space through a L1-to-L2 network. The 16 L2 cache banks are distributed across the entire processor and are located together with 16 memory controllers.

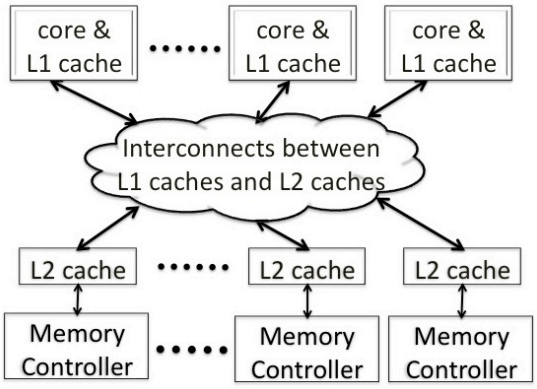


Fig. 1: The NoC in manycore system with distributed L2 cache banks [15].

Network-on-Chip (NoC) design paradigm brings the techniques developed for macro-scale, multi-hop networks into a chip to improve system performance and design. The major goal is to achieve greater design productivity and performance by handling the increasing parallelism, manufacturing complexity, wiring problems, and reliability [1], [16], [17] . Many NoCs have been proposed in literature [3], [19], [20] but comparing and analyzing those remains problematic due to vague documentation and proprietary test cases. Hence, accurate, representative traffic models for benchmarking and designing NoCs are needed.

We at OCP-IP NoC Benchmarking group are currently working towards standardized test case set and methodology [8], [21]. We have published for example a SystemC tool called Transaction Generator (TG) [23] which includes two sets of benchmark applications [18] [14]. In general, test cases can be divided into computation kernels (e.g. IIR, FFT) and full applications (e.g. image and video processing, telecommunications). Both types can be modeled in many different ways. Actual applications give the best accuracy but majority of publications use *synthetic traffic*, such as uniform random traffic [19], [20].

We are aiming at *traffic profiles of real applications* as a trade-off between these extremes. They should provide adequate accuracy (much better than purely synthetic) and also easier portability, scaling, and analysis (much better than applications). Moreover, we encourage designers to systematically evaluate a large set of traffics scenarios and system parameters, see for example [22]. This paper presents the profiling of EEMBC MultiBench 1.0 multicore benchmark suite [7], [9], [10] on 64-core system using M5 and BookSim simulators [2] [6].

EEMBC benchmark performance is measured while varying the number concurrent workloads and worker threads per workload. The results indicate that increasing the number of concurrent workloads can significantly improve the system performance. We also find the classical L1-to-L2 bus network does not always provide enough bandwidth for EEMBC benchmarks. For example, a more complicated electrical or silicon-photonic NoC that provides higher bandwidth can be beneficial for these benchmark programs. The following chapters present discussion about profiling and the detailed result spreadsheet can be downloaded from the OCP-IP web site[1].

II. EEMBC MULTIBENCH

EEMBC MultiBench 1.0 is a multicore benchmark suite meant for measuring the throughput of multiprocessor systems, including those built with multicore processors [7], [9]. The user may change the number of workers running in parallel, size of the dataset, as well as their binding to processing elements. Software assumes *homogeneous general-purpose* processing elements. Threads are used to express concurrency, and each thread has *symmetric memory visibility*. In EEMBC terminology, *kernel* means the algorithm to be executed (e.g. jpeg decompression). *Work Item* binds a kernel to specific data (e.g. jpeg decompression of 16 images) whereas *workload* consists of one or more work items (e.g. jpeg decompression of 16 images, rotation of the results, and jpeg compression). One or multiple *worker* threads can be assigned to each work item. Figure 2 shows the concept of concurrent workloads and workers.

The suite addresses 3 challenges:

[1]http://www.ocpip.org/white_papers.php

1) Portability - Benchmarks must be portable to many different embedded architectures. Therefore, the kernels are written in C which is the de facto standard in the embedded industry. Moreover, the API related to parallel programming has been minimized to 13 calls and 3 data structures. There is a direct mapping to the more complex *pthreads* interface.
2) Scalability - Arbitrary number of computation contexts is supported and performance is measured as *workloads per second*. Amount of work can be kept constant regardless of the number of contexts used.
3) Flexibility - Benchmarks support many approaches to parallel programming, such as task decomposition (workers running in parallel) and data decomposition (multiple threads processing single piece of data). However, functional pipelining is not considered in version 1.0.

MultiBench 1.0 includes embedded algorithms from earlier EEMBC suites and some new ones. Tasks include for example processing TCP/IP packets, image rotation, MD5 checksums, and video encoding. The suite includes about 30 workload applications written in C and user can create more by instantiating work items in workload creator program GUI. Table I summarizes the 16 profiled workloads. They are sorted according to average IPC in our measurements (see next sections). We chose to limit the working set size to $4M$ per context, e.g. 4 megapixels. EEMBC benchmarks have previously been analyzed for example in [10]–[12].

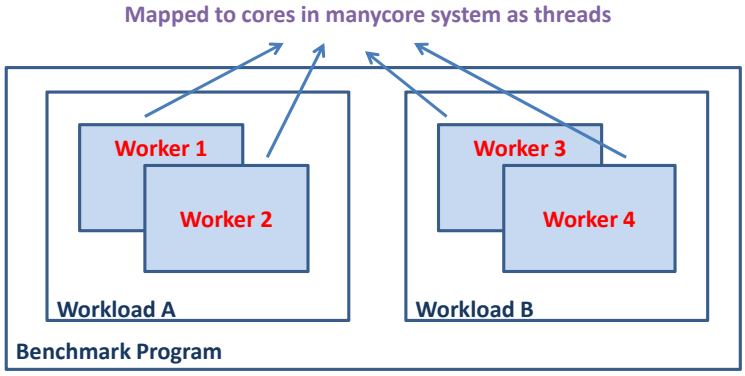


Fig. 2: Example of 2 concurrent workloads A and B and 2 workers per workload. There are 4 threads in total to be mapped on a multiprocessor. This setup would be denoted as $c2w2$.

TABLE I: Benchmarked EEMBC MultiBench programs, in the order of increasing average IPC. Maximum IPC and L1-L2 BW are measured with 64 threads and average IPC for 1-64 threads.

#	Program	Avg IPC	Max IPC	Max BW [GB/s]	Description
1	ipres-4M	0.4	0.8	7.8	Send 4 greyscale images to a printer over the network
2	4M-reassembly	0.6	0.8	7.8	Reassemble fragmented IP packets
3	4M-check-reassembly	0.7	0.9	6.3	Check and reasssemble IP packets
4	4M-check-reassembly-tcp	1.0	3.0	14.3	Pass network to target
5	4M-check-reassembly-tcp--cmykw2-rotate	1.1	2.5	17.3	Send images over network, and print in landscape orientation
6	4M-check	1.2	2.8	15.8	Check IP packet header
7	rotate-34kX128w1	1.5	2.3	15.7	Rotate 128 images by 90 deg clockwise
8	rotate-color1Mp	1.6	5.0	37.7	Rotate 1 MPixel color image by 90 deg
9	4M-cmykw2-rotatew2	1.6	4.0	21.9	Combine rotation and color conversion
10	4M-rotatew2	1.8	5.5	20.0	Rotate image by 90, 180 or 270 degrees, memory intensive
11	4M-tcp-mixed	2.4	15.0	20.9	Most processing-intensive portion of RFC793 protocol
12	md5-4M	2.5	7.9	18.0	Message-digest checksum used in cryptography
13	4M-check-reassembly-tcp-x264w2	2.7	7.7	23.3	Encode video H.264 and send over the network
14	rgbcmyk-4M	3.1	9.2	49.4	Convert RGB to CMYK color
15	iDCT-4M	3.3	16.8	21.0	Inverse discrete cosine transform, from Automotive suite
16	4M-x264w2	6.9	13.7	37.2	Encode H.264 video, computationally intensive.

III. Profiling Setup

Figure 3 shows our profiling approach using M5 full-system simulator [2] that is integrated into BookSim network simulator [6]. Applications are simulated and accurate log files are generated. Since detailed simulation is slow and somewhat tedious,

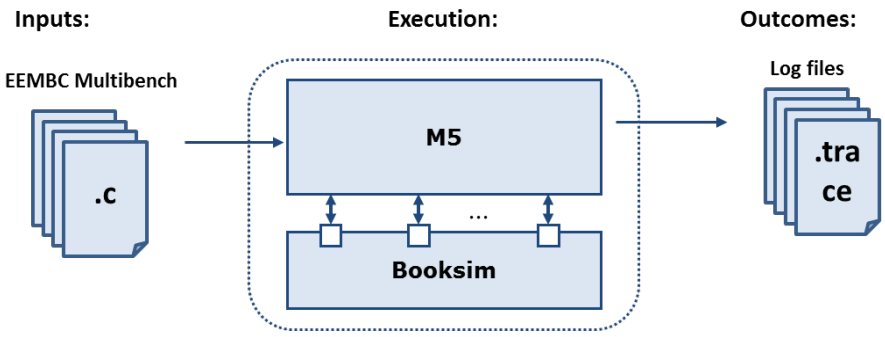


Fig. 3: Simplified view of the profiling and benchmarking steps.

we expect that most NoC benchmarking and design space exploration is carried out with abstract workload models, e.g. using Transaction Generator [23] and processed traffic traces.

A. Simulators

M5, or gem5, is a full system simulator developed in C++/Python by University of Michigan. It is freely available[2] under Berkeley-style license. It can simulate the performance of the entire computer systems. For example, the user can choose various ISA (such as Alpha, AMD and x86), cache architecture (private L2 cache or shared L2 cache), and the dimension of chip components (such as core counts, number of floating point units, and L1/L2 cache sizes).

Since the bus topology does not always provide sufficient network bandwidth, we have integrated the BookSim network simulator into M5. BookSim is an open source[3] network simulator developed in C++ by Stanford University. It can simulate the performance and power of various network configurations. For example, the user can choose various network topologies (such as mesh, clos and crossbar), the use of virtual channel technology and the dimension of network components (such as channel width and router butter size), etc.

Unfortunately, simulating topologies other than bus is significantly lower. For example, the execution of 1 second of EEMBC benchmark in simulator can take $2-4$ days on a 2.3 GHz single-core host machine. It is nearly $3x$ slower than the same benchmark with the default bus topology. Thus in Section IV, we use the default bus in M5. However, some benchmarks with large number of concurrent workloads require very high network bandwidth and the default bus topology saturates.

B. Metrics

The measured properties include network bandwidth, benchmark performance demands, and bus latency, as listed in Table II. Instrcution per cycle (IPC) is the primary performance metric in this study as it measures the amount of computation performed. Bandwidth does go hand in hand with IPC, however small bandwidth is preferred to keep bus non-saturated. Cache miss latency denotes the time that CPU is stalled upon a cache miss, and of course smaller cache miss latency is better for performance.

The two last columns list the measured typical values. The minimum and maximum values shows the difference between benchmark programs. Column $w1$ includes 4 cases $w1c1, w1c4, w1c16, w1c64$, whereas $w16$ includes only 2 cases $w16c1, w16c4$. Columns $c1$ and $c16$ are collected similarly. We also notice parallelism (many concurrent workloads and workers) makes all the numbers larger.

TABLE II: Measured properties on a 64-core system running at most 64 threads.

Metric	Unit	Min, Avg, Max		Min, Avg, Max	
		$w1$	$w16$	$c1$	$c16$
Instruction per cycle (IPC)	1/cycle	$0.2, 2.3, 16.8$	$0.2, 1.8, 11.2$	$0.2, 1.0, 3.9$	$0.2, 3.2, 13.7$
L1-L2 bandwidth (BW)	GByte/s	$1.1, 10.3, 49.4$	$1.0, 8.5, 27.7$	$0.9, 5.3, 14.5$	$1.2, 13.5, 37.2$
L1 miss penalty	cycle	$94, 268, 1957$	$92, 196, 413$	$92, 204, 422$	$89, 257, 1037$

[2][Online] http://www.m5sim.org/Main_Page
[3][Online] https://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/BookSim

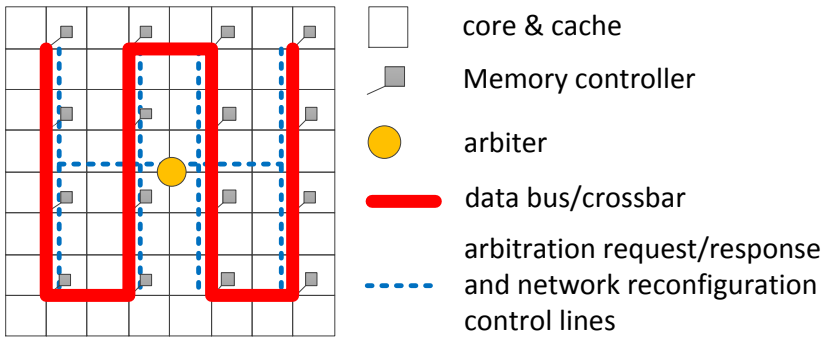


Fig. 4: Bus layout between L1s and L2s in a 64-core manycore processor [15]

TABLE III: Micro-architectural parameters of the target system in M5.

Micro-architecture Configuration	
Core Frequency	2.0 GHz
Issue	2-way Out-of-order
Reorder Buffer	128 entries
Functional Units	2 IntAlu, 1 IntMult 1 FPALU, 1 FPMult
Physical Regs	128 Int, 128 FP
Instruction Queue	64 entries
Branch Predictor	Tournament predictor
L1 ICache	16 KB @ 2 ns each
L1 DCache	16 KB @ 2 ns each
L2 Cache	4-way set-associative, 64 B block Distributed 16 x 1 MB @ 6 ns
NoC	Split bus, 1.0 GHz, 64 Bytes
Main memory	1GB, 50 ns access time (100 cycles)

C. Settings

The target system parameters are listed in Table III. The system is composed of 64 cores, 128 L1 caches (1 I-Cache and 1 D-Cache per core), distributed L2 cache (16 banks), and 16 memory banks (enough size to hold the working set). The 128 L1 caches are connected to a L2 cache through a L1-to-L2 bus. Each L2 bank cache is connected to memory through L2-to-mem bus. The address space is interleaved among the L2 cache banks and memory banks. The bus bandwidth demands and bus penalty shown in our plots are those measured on L1-to-L2 bus.

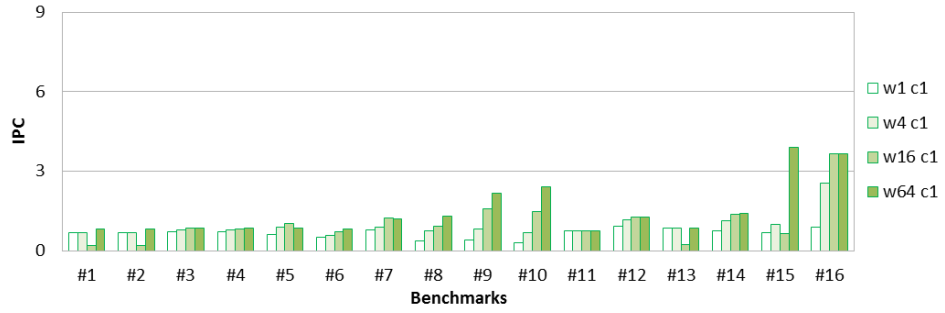
Figure 4 shows an example of L1-to-L2 networks. This network uses split-bus topology and the bus arbitration block is located in the center of the chip. Our simulations use this L1-to-L2 bus network to show the effect of network demands of EEMBC benchmarks.

The core works at 2 GHz and the rest of the system works at 1 GHz. A typical L1 miss penalty include L1-to-L2 bus round trip latencies (20+ core cycles) and L2 access time (6 $ns = 12$ core cycles) at minimum, whereas L2 miss requires additional L2-to-mem bus round trip latency (more than 10 core cycles) and memory access time (50 $ns = 100$ core cycles). These example bus round trip latencies are the so called *zero-load latencies*, which only happen in the ideal case when there is no bus contention. In a real system, the bus has slightly higher latencies, but they increase very rapidly if the traffic load increases beyond the saturation threshold.

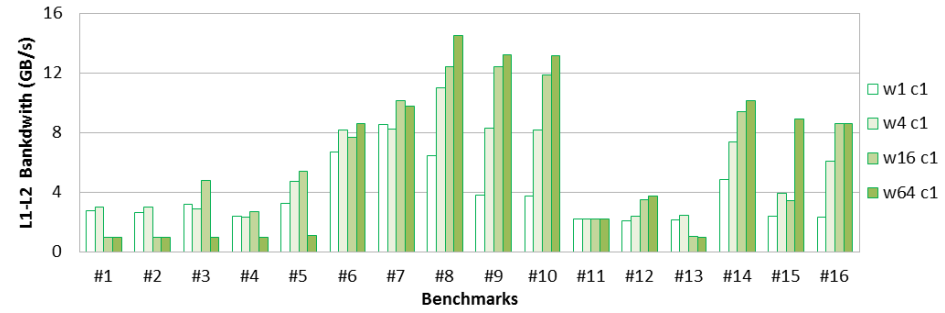
For each benchmark, we compare 1, 4, 16, 64 concurrent workloads (shown as 'c') and 1, 4, 16, 64 workers for each workload (shown as 'w'). For example, $w64c64$ means 64 workloads with 64*64 workers (threads) working on it. Thus $4 \cdot 4 = 16$ simulations were needed for each of the 16 workloads (256 runs in total).

We didn't control the mapping. Since we use the full system mode of Gem5 simulator, the Linux OS running on the target machine determines the thread mapping according the available core count. For 64 threads running on 64 cores, we could see 1 thread mapped onto one core. We didn't do thread binding, thus there might be thread migrations.

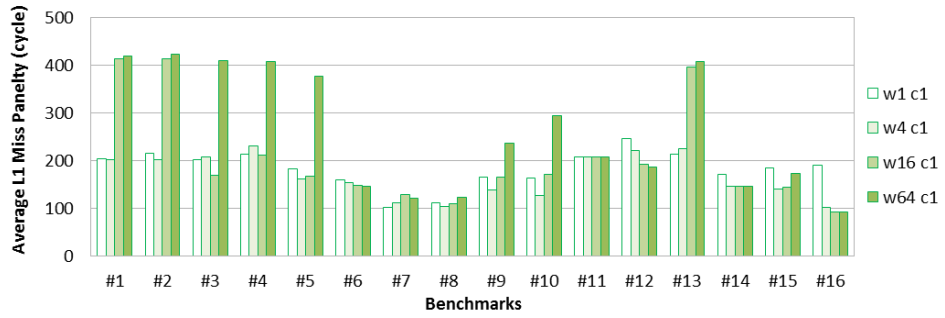
Please note we simulate at most $1s$ due to limited simulation speed. Some of applications didn't complete in that time and we only consider the average value and tracing within $1s$.



(a) Instructions-per-cycle



(b) Bandwidth



(c) Average L1 miss penalty

Fig. 5: Profiling results when running the benchmark programs alone (fixed $c = 1$) and varying the number of workers per workload ($w = 1, 4, 16, 64$). The benchmark names corresponding to the IDs can be found in in Table I.

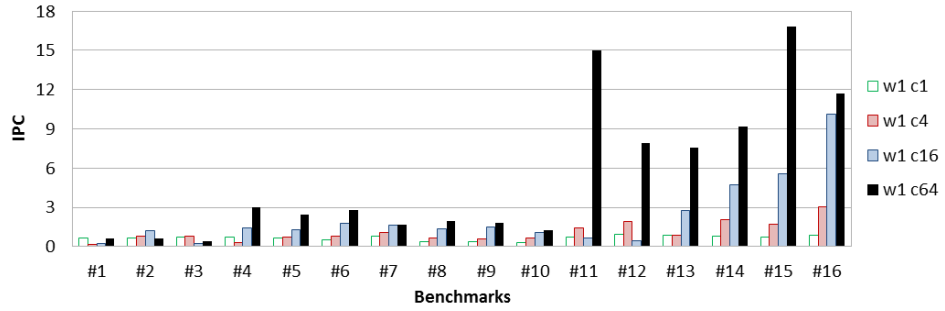
IV. Findings

We run the benchmark programs with various number of parallel workloads c with various number of workers per workload w. Figure 5 shows the simulation results while fixing the number of workloads ($c = 1$) and changing the number of workers per workload ($w = 1, 4, 16, 64$). On the other hand, Figure 6 shows the simulation results while changing the number of workloads ($c = 1, 4, 16, 64$) and fixing the number of workers per workload ($w = 1$). The benchmarks names corresponding to the benchmark ID can be found in Table I. The complete measurement results can be found in Appendix and the associated Excel spreadsheet.

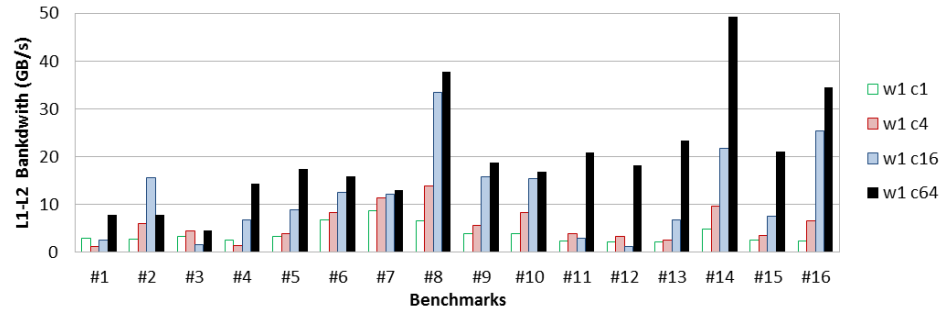
A. Workers (threads)

The first method of improving the system performance is to increase the number of workers (threads) of each workload. Figure 5 shows that the system performance, measured in instructions per cycle (IPC), increases when we increase the number of workers from 1 to 64 for all benchmarks. The results are sorted in asceding order of average IPC. However, measured IPC is surprisingly low on average, $IPC_{w=1} = 0.7$ and $IPC_{w=64} = 1.5$.

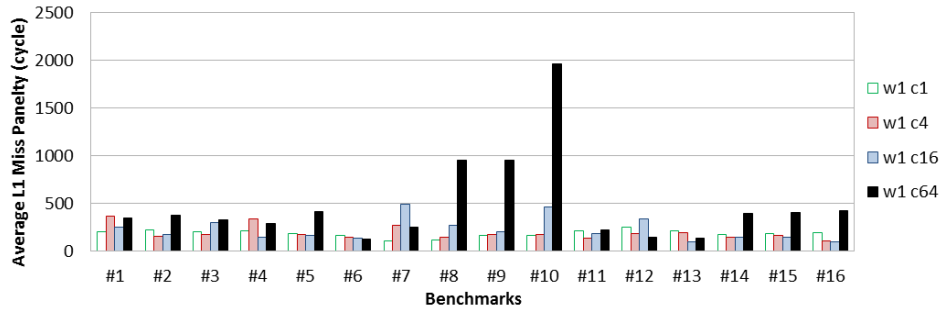
There is, of course, variation between applications. The ratio between max and min IPCs is $3 - 20x$, and between max and avg about $1.5 - 3x$. There are some peculiar results. For example, a spike at 64 workers with workload #15, iDCT. In some



(a) Instructions-per-cycle



(b) Bandwidth



(c) Average L1 miss penalty

Fig. 6: Profiling results when running the benchmark programs with various number of concurrent workloads ($c = 1, 4, 16, 64$) and just a single worker per workload (fixed $w = 1$). The benchmark names corresponding to the IDs can be found in in Table I.

cases, the IPC increases otherwise but drops with $w16$, e.g. #1 and #13. The exact reason is yet unknown.

Fig. 5(b) shows the offered bandwidth in L1-L2 bus. On average, $BW_{w=1} = 1.8\ GB/s$ and $BW_{w=64} = 6.1\ GB/s$. Most benchmarks show that the offered bandwidth also increases while the number of workers increases. However, some benchmarks (such as #1, #3) show that the offered bandwidth can decrease by $2-3x$ when running the workload with 32 or 64 workers. This is because the increasing number of parallel threads requires more synchronization and larger IPC causes more contents to be processed in a given period. Measured bandwidth drops when the system suffers very high L1 miss penalties, as seen in cases #1-#5 and #13 in Figs. 5(b) and 5(c).

A significant portion of L1 miss penalties is due to the round-trip latencies in L1-to-L2 bus. While increasing the number of workers, the L1-to-L2 arbitration time increases and the increasing bandwidth demand causes contentions in the L1-to-L2, which therefore causes the decrease in offered bandwidth and the increase in L1 miss penalties. In many cases, miss penalty is rather constant when w increases and sometimes it even decreases a little. However, in 5 cases setting $w = 64$ nearly doubles the miss latency, and in 4 cases this happens on both $w = 32$ and $w = 64$. On average, $t_{L1miss,w=1} = 180$ and $t_{L1miss,w=64} = 239$ cycles.

In Figure 5, the offered bandwidth in the L1-to-L2 bus varies in the range of $0.94-14.5\ GByte/s$ ($0.47-7.26\ Byte/cycle$

w.r.t. CPU clock) and the average offered bandwidth is 5.3 GB/s. The system performance (IPC) varies in the range of $0.19 - 3.92$, and the average system performance (IPC) is 1.04. The system performance scales, but not well, while increasing the number of workers for a single workload.

B. Concurrent Workloads

We have shown that the EEMBC benchmarks show some performance improvements while we scale up the number of workers for each workload. Another method of increasing the system performance is to execute multiple concurrent workloads and it seems more powerful. In Figure 6, we fix the number of workers for each workload ($w = 1$), while increasing the number of concurrent workloads. We find that both the system performance and offered bandwidth increases more than in previous experiment. For some benchmarks (such as #8, #9 and #10), the L1-to-L2 bus reaches the saturation region while running 64 concurrent workloads, and therefore the system suffers extremely high L1 miss penalties (about $1\ 000 - 2\ 000$ cycles).

Figure 6 shows that the system performance, IPC, varies now in the range of $0.2 - 16.82$, and the average IPC is 2.3. The offered bandwidth in the L1-to-L2 bus varies in the range of $1 - 50$ GByte/s ($0.5 - 25\ Byte/cycle$), and the average offered bandwidth is 9.2 GByte/s. Average cache miss penalty increases by 30% from previous case to 260 cycles. By comparing Figure 5 and Figure 6, we find that the system performance improves more significantly while we increase the number of concurrent workloads.

C. Increasing both the workers and workloads

Fig. 7 shows how IPC and bandwidth demand rise when the number of threads increases. However, since there are 64 cores, both values peak at 64 threads and either saturate or drop after that. Both average and maximum values behave similarly. Therefore, the detailed results in appendices do not include cases where number of threads exceeds 64, i.e. cases $w4c64$, $w16c16$, $w6, 64$, $w64c4$, $w64c16$, and $w64c64$.

Fig. 7(c) shows the IPC with 64 threads on different applications. It is evident that larger c benefits more than large w. However, average IPC is lower than expected, and should be investigated more thoroughly. For example, our previous simulations show that PARSEC and NAS benchmarks could achieve an IPC of $10 - 30$ while running 64 threads on same 64-core manycore platform [4], [5], [15] (the core frequency might differ, though).

V. CONCLUSIONS

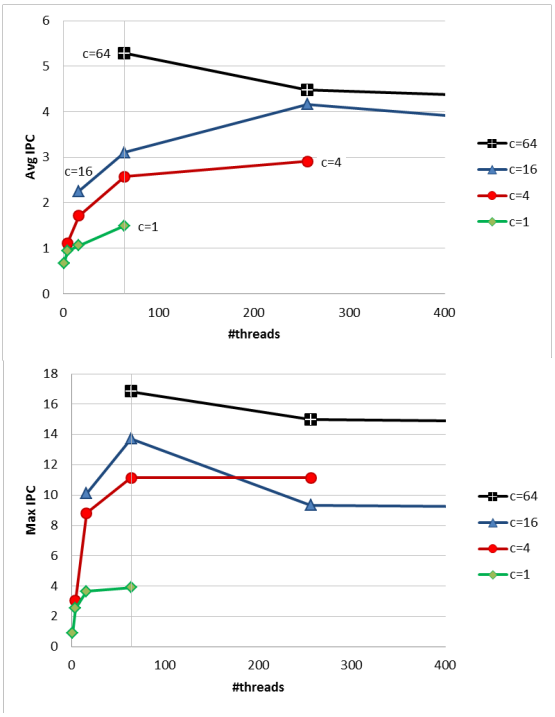
This paper gives an overview of profiling the EEMBC benchmark suit which is summarized in Table IV. Our methodology integrates M5 and BookSim simulators in order to evaluate EEMBC and other parallel benchmarks with various NoC configurations. The collected traffic traces can be later utilized in NoC benchmarking with traffic generators. Our simulation results show a maximum offered bandwidth of $50\ GByte/s$, and a maximum system performance (IPC) of 16.82. The average values are about $9\ GByte/s$ and 2 IPC. We also find that given a 64-core manycore system, both increasing number of concurrent workload and increasing number of workers per workload can help improve the system performance. But the number of concurrent workload shows a more significant impact on the system performance. Detailed simulation results are collected in the accompanied spreadsheet *chen_eembc_profiling_2013.xlsx*.

ACKNOWLEDGEMENT

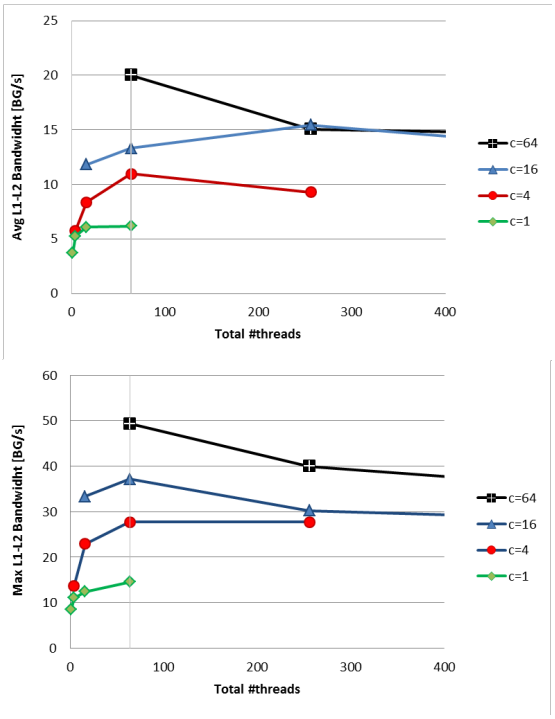
Authors wish to thank Markus Levy and Shay Gal-On from EEMBC for providing the benchmarks and valuable help. Moreover, Ian Mackintosh and Joe Basquez from OCP-IP have also helped a lot in coordination which is gratefully acknowledged.

REFERENCES

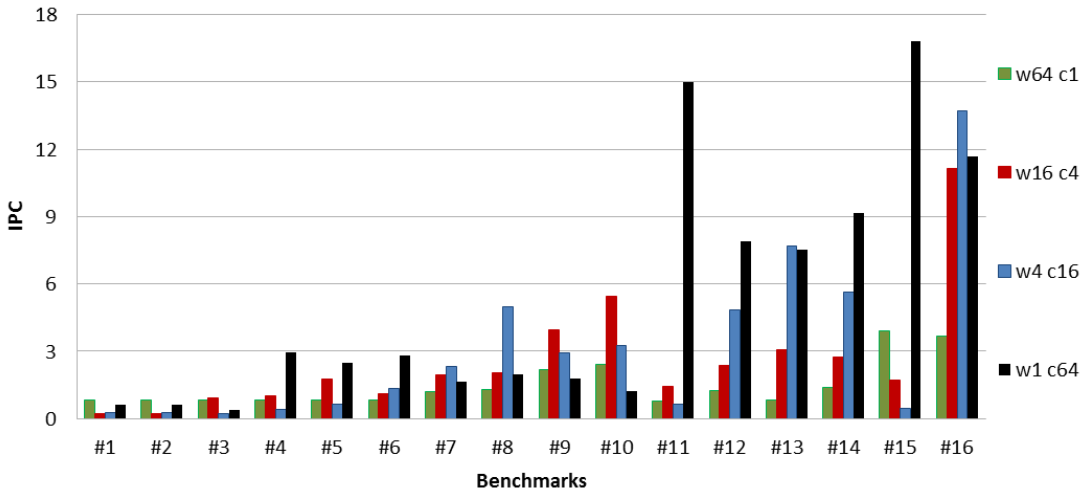
[1] L.Benini and G. de Micheli, "Networks on chips: A new SoC paradigm," IEEE Computer, vol. 35, no. 1, Jan. 2002, pp. 70-7.
[2] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi, and S. Reinhardt, "M5 simulator: Modeling networked systems," Micro, IEEE, vol. 26, no. 4, 2006, pp. 52-60.
[3] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," ACM Computing Surveys, vol. 38, no. 1, article No. 1, Jun. 2006, 51 pages.
[4] C. Chen, J. Meng, A.K. Coskun, A. Joshi, "Express Virtual Channels with Taps (EVC-T): A Flow Control Technique for Network-on-Chip (NoC) in Manycore Systems," in High Performance Interconnects (HOTI), Aug. 2011, pp.1-10.
[5] C. Chen, A, Joshi, "Runtime Management of Laser Power in Silicon-Photonic Multibus NoC Architecture," IEEE Journal of Selected Topics in Quantum Electronics, vol. 19, no. 2, Mar-Apr. 2013, 13 pages.
[6] W. Dally and B. Towles, Principles and Practices of Interconnection Networks. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.
[7] MultiBench 1.0 Multicore Benchmark Software, The Embedded Microprocessor Benchmark Consortium, [Online] Available: http://www.eembc.org/benchmark/multi_sl.php
[8] C. Grecu, A. Ivanov, P.P. Pande, A. Jantsch, E. Salminen, U. Ogras, R. Marculescu, "Towards Open Network-on-Chip Benchmarks," in International Symposium on Networks-on-Chip (NOCS), May 2007, pp. 205-205
[9] T.R. Halfhill, "EEMBC MultiBench arrives," Microprocessor report, July 2008, 8 pages.
[10] J.A. Poovey, T.M. Conte, M. Levy, S. Gal-On, "A Benchmark Characterization of the EEMBC Benchmark Suite," IEEE Micro, vol.29, no.5, Sep-Oct. 2009, pp.18-29.



(a) Instructions-per-cycle



(b) Bandwidth



(c) IPC with 64 threads.

Fig. 7: Average amd maximum IPC and bandwidth as function of thread count. Both values rise first, peak at 64 threads, and then drop.

TABLE IV: Summary of profiling the EEMBC MultiBench

Parameter	Value	Notes
Benchmark	EEMBC MultiBench	www.eembc.org
#workloads	16	4MByte dataset
#concurrent workloads	$c = 1, 4, 16, 64$	
#workers	$w = 1, 4, 16, 64$	
Simulator	M5	+BookSim optionally
CPUs	64 dual-issue cores	2 GHz
L1 Cache	16+16KB per core	500 MHz
L2 cache	16 * 1 MB	1 GHz, 64 B L1-L2 bus
Avg IPC	$0.7 - 1.5$	$w1c1 - w1w64$
	$1.1 - 2.6$	$w4c1 - w4w16$
	$2.3 - 3.1$	$w16c1 - w16w4$
	5.3	$w64c1$
Avg traffic on L1-L2 bus [GB/s]	$3.7 - 6.2$	$w1w1 - w1c64$
	$5.8 - 11.0$	$w4c1 - w4c16$
	$11.8 - 13.3$	$w16c1 - w16c4$
	20.3	$w64c1$

[11] A. Kejariwal, A.V. Veidenbaum, A. Nicolau, M. Girkar, Xinmin Tian, H. Saito "Challenges in Exploitation of Loop Parallelism in Embedded Applications," in CODES+ISSS, Oct. 2006, pp. 173-180.
[12] M.M. Islam, A. Busck, M. Engbom, S. Lee, M. Dubois, P. Stenström, Limits on Thread-Level Speculative Parallelism in Embedded Applications, in INTERACT-11 (in conjuntion with HPCA-13), Feb 2007, pp. 40-49.
[13] D. Lattard, E. Beigne, F. Clermidy, Y. Durand, R. Lemaire, P. Vivet, and F. Berens, "A reconfigurable baseband platform based on an asynchronous network-on-chip," IEEE J. Solid-State Circuits, vol. 43, no. 1, Jan. 2008, pp. 223-235.
[14] W. Liu, J. Xu, X. Wu, Y. Ye, X. Wang, W. Zhang, M. Nikdast, Z. Wang, "NoC Traffic Suite Based on Real Applications," IEEE Computer Society Annual Symposium on VLSI (ISVLSI), July 2011, 6 pages.
[15] J. Meng, C. Chen, A. Coskun and A. Joshi, "Run-Time Energy Management of Manycore Systems Through Reconfigurable Interconnects," in Great Lakes Symposium on VLSI (GLSVLSI), May 2011, pp. 43-48.
[16] U. Y. Ogras, J. Hu, and R. Marculescu, "Key research problems in NoC design: a holistic perspective," in CODES, Sep. 2005, pp. 69-75.
[17] J. Owens, W.J. Dally, R. Ho, D.N. (Jay) Jayasimha, S.W. Keckler, Li-Shiuan Peh, "Research challenges for on-chip interconnection networks," IEEE Micro, vol. 27, no. 5, Sep-Oct. 2007, pp. 96-108.
[18] E. Pekkarinen, L. Lehtonen, E. Salminen, T.D. Hämäläinen, "A Set of Traffic Models for Network-on-Chip Benchmarking," in International Symposium on System-on-Chip, Oct-Nov. 2011, pp. 78-81.
[19] E. Salminen, A. Kulmala, and T. Hämäläinen, "On network-on-chip comparison," in Euromicro DSD, Aug. 2007, pp. 503-510.
[20] E. Salminen, A. Kulmala, and T. Hämäläinen, "Survey of Network-on-chip Proposals," OCP-IP white paper, March 2008, 13 pages.
[21] Erno Salminen, Krishnan Srinivasan, Zhonghai Lu, "OCP-IP Network-on-chip benchmarking workgroup," OCP-IP, [online], Dec. 2010, 5 pages.
[22] V. Soteriou, N. Eisley, H.Wang, B. Li, and L.-S. Peh, "Polaris: A system-level roadmapping toolchain for on-chip interconnection networks," IEEE Trans. VLSI Syst., vol. 15, no. 8, Aug. 2007, pp. 855-868.
[23] Transaction Generator package, OCP-IP, April 2012, [Online] Available: http://www.ocpip.org/tg_package.php
[24] S. Vangal et al., "An 80-tile Sub-100-W TeraFLOPS processor in 65-nm CMOS," IEEE J. Solid-State Circuits, vol. 43, no. 1, Jan. 2008, pp. 29-41.

Chao Chen received the B.Eng. degree in electronics engineering and B.Econ. degree in international trade and business from Shanghai Jiao Tong University, Shanghai, China, in 2005 and 2006, respectively, and theM.S. degree in electrical engineering from the Pohang University of Science and Engineering, Pohang, South Korea, in 2007. He has been working toward the Ph.D. degree in electrical and computer engineering at Boston University, Boston, MA, since 2009. From 2007 to 2009, he was an SOC Engineer at Core Logic, Inc., Seoul, Korea, where he was mainly involved in designing multimedia processors formobile applications.His research interests include reconfigurable network architectures and silicon-photonic technology for on-chip and off-chip communications. His research interests also include embedded system design and field-programmable gate array implementations for scientific applications.

Ajay Joshi received the B.Eng. degree in computer engineering from the University of Mumbai, Maharashtra, India, in 2001, and the M.S. and Ph.D. degrees in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, GA, in 2003 and 2006, respectively. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Boston University, Boston, MA. Prior to joining Boston University, he was a Postdoctoral Researcher with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology from 2006 to 2009. His research interests include various aspects of very large scale integration design including circuits and systems for communication and computation, and emerging device technologies including silicon photonics and memristors. Dr. Joshi received the U.S. National Science Foundation CAREER Award in 2012.

Erno Salminen received his MSc and PhD degrees in 2001 and 2010 from Tampere University of Technology, Finland. Currently he works there as a post-doctoral researcher and acts as a chairman of OCP-IP Network-on-Chip benchmarking workgroup. His research interests include design of NoC and MPSoC, as well as digital logic in general. He has (co-) authored over 70 scientific articles and received the OCP-IP Contributor Of The Year award in 2009.

VI. Appendix: Detailed results

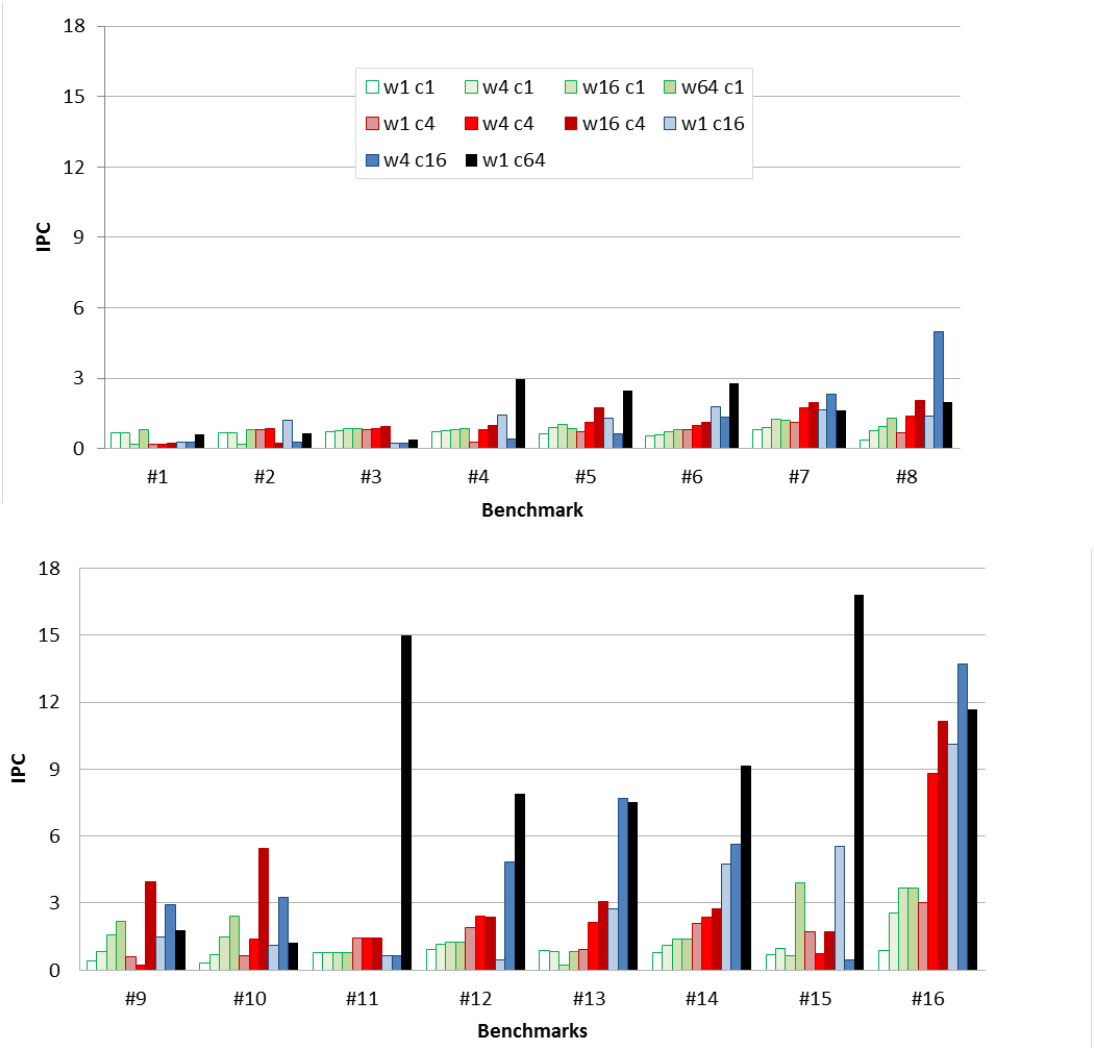


Fig. 8: IPC or each program.

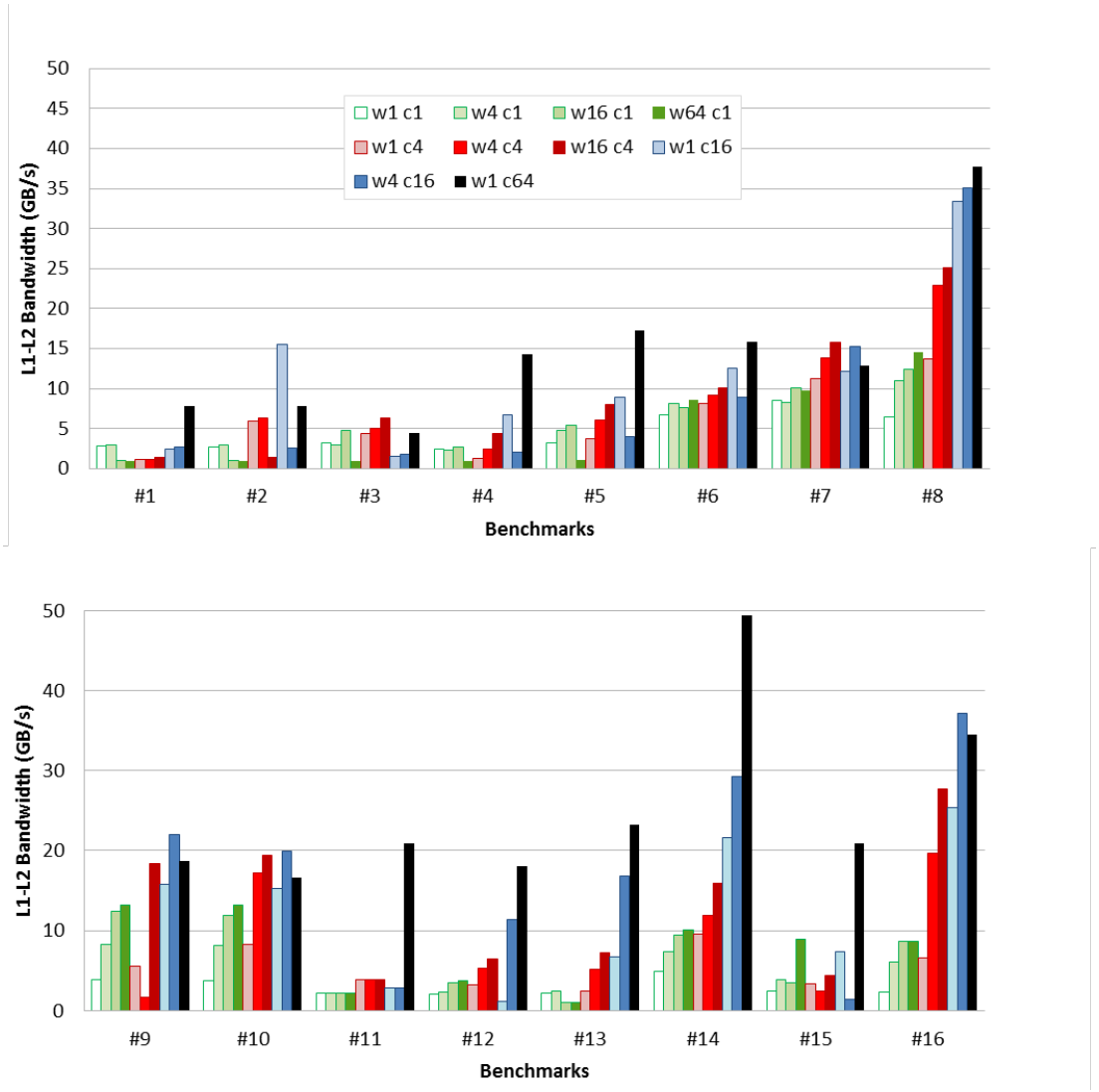


Fig. 9: L1-L2 bandwidth of each program.

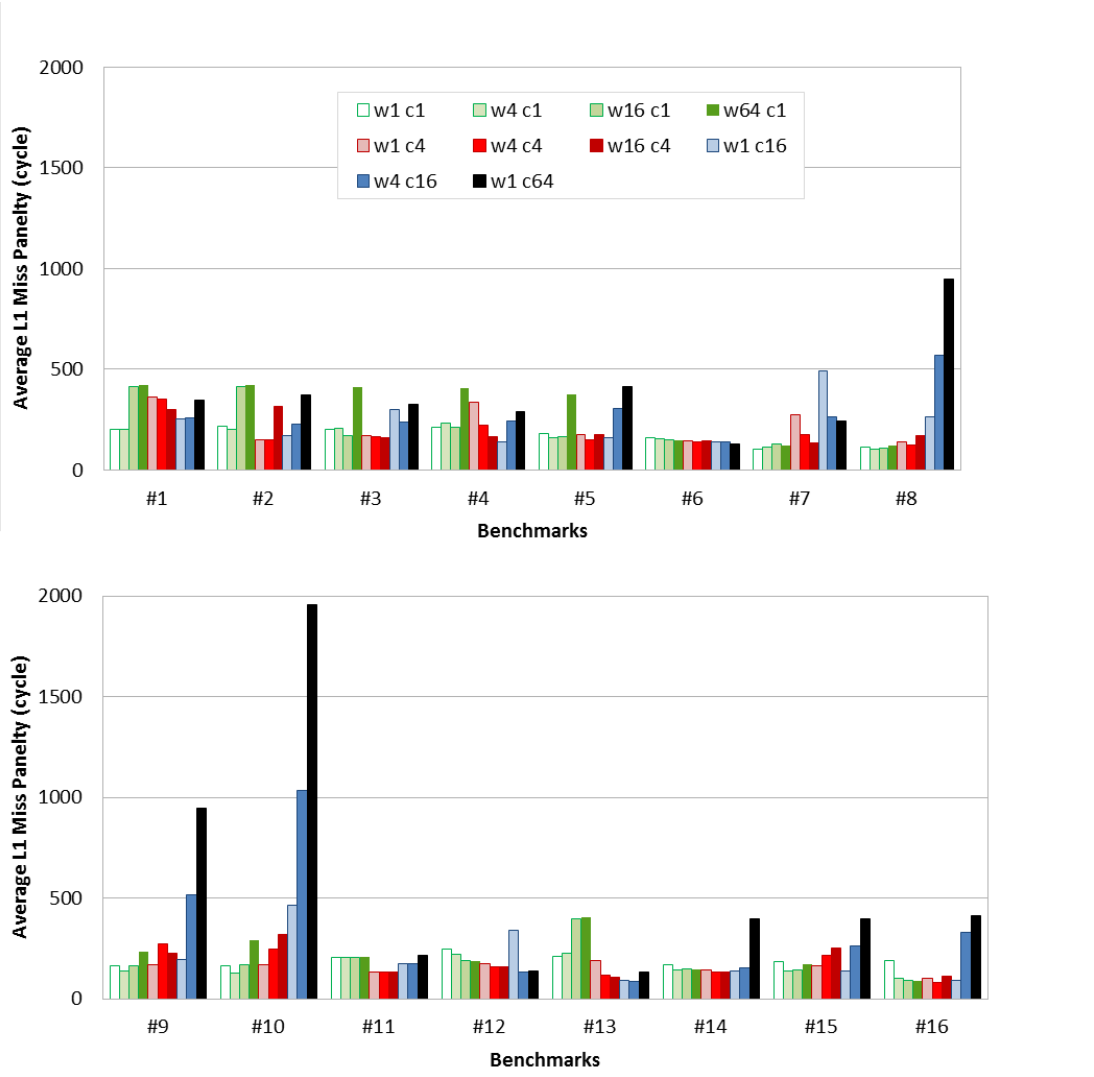


Fig. 10: Cache miss latency of each program.