International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-9 Issue-2, December, 2019

Parameter Analysis of Interfering Applications in Multi-Core Environment for Throughput Enhancement

Surendra Kumar Shukla, P.K. Chande

Abstract- In Multi-core systems the applications co-execute in Multi-programmed mode, have interfere with each other during execution, which creates resource bottleneck affecting the performance. To reduce the interference in a given set of resources some conventional approaches don't give guarantee of performance in a conflicting application environment. In this paper, we make an in-depth analysis of benchmark applications interference for shared resources and find out application set which could be executed adopting a designated policy to mitigate the interference effects. In this work, we have performed profiling and analysis of applications on the state-of-the-art simulator gem5. Finally, we conclude the possibility of performance improvement through the designated policy. The simulation results show the scope to have a new scheduler for performance improvement in such systems.

Keywords: ***Interference, Multi-core, analysis, performance, policy, co-schedule.***

I. INTRODUCTION

Multi-core systems execute a diverse set of applications. The diversity is due to the distinct characteristics of the applications which exit in its virtue. The diversity causes conflicts among the applications, which further create the issue of interference effects in the Multi-programmed environment [1]. The interference, in turn, affects the system behavior, due to the activities ordered by the applications running on various cores. In other words, interference is an undesired phenomenon that alters the system performance.

To reduce the interference effects, it is essential to find the behavior of the applications, which would cause this [2]. To find the behavior of the application, some profiling based approaches are available, where the applications are run in a reference environment [3]. Knowing the essential characteristics of applications in terms of system parameters like latency could help in finding a suitable approach to execute the application in the said conflicting environment.

To find the behavior of different applications running in isolation, a lot of research has been carried out in literature [4] [5]. However, to find the application's behavior (in totality some intended combination), few attempts are made to some extent [6] [7]. The behavior of the applications becomes more unpredictable when executed in some intended "combinations," which is termed here as co-schedule [8]. Co-schedule is a group of independent applications/benchmarks which get executed in the cores simultaneously. The behavior of application also depends on the application would be mapped to a processor core. However, the decision of the core mapping solely depends on the scheduler of the host machine. The performance of the system also depends on the number of threads in a benchmark. In general, increasing the number of threads in a benchmark may increase the system throughput, but it also affects the performance adversely after a saturation stage. The concurrent execution of benchmarks is also a possibility to improve the performance.

The benchmarks of co-schedules might be single-threaded or multi-threaded. Single-threaded benchmarks, in general, do not require specific scheduling policy, and they are bound to a core as per the order specified in the simulation script, and they could not switch to other cores in run time. However, multi-threaded benchmarks threads could switch from one core to another core as per the scheduling policy.

It is known that, in general, increasing the number of cores, can increase the possibility of performance improvement in terms of throughput. However, it is not always valid as Multi-core systems shares some common elements like last level cache, bus, DRAM controller, etc. which create the resource conflicting issues at run time.

To increase the performance of Multi-core systems, it is essential to quantify the behavior of the applications and interference effects. The analysis is a useful tool to find out the causes, factors in terms of parameters responsible for performance degradation [9]. There are various performance factors which stand latent and show their effect when multiple co-scheduled applications interact with each other. It is vital to find such factors which require in-depth intended simulations and their analysis.

One such concealed factor is memory access latency. Memory access latency is a critical factor for the performance of Multi-core systems. Memory Access latency becomes vital when applications/benchmarks conflicts/interfere with the shared resources like bus and Last level cache. The conflicts mainly influence memory access latency in Last level cache and contention in the bandwidth [10]. Hence, it is required to have an in-depth understanding of the behavior of benchmarks/applications in combination; to diminish the effect of memory latency effects.

Revised Manuscript Received on December 30, 2019.
* Correspondence Author
Surendra Kumar Shukla*, Research Scholar, School of Computer Science & IT, Devi Ahilya University, Indore, India, Email: surendr.shukla@gmail.com
Dr. P.K Chande, Ex-Director, MNIT Bhopal, SGSITS Indore, India, Email-pkchandein@gmail.com

Analysis of system parameters like miss rate, bandwidth utilization, the total numbers of branches committed, etc
Are proved to be beneficial to find out the hidden causes of the performance degradation in Multi-core systems, [11]. However, the system parameters behavior is observed to be different from different application combinations, which calls for an in-depth approach precisely interpret the outcome of the execution of the application in terms of performance [12].
For example, on an application execution scenario, a higher value of bandwidth utilization indicates the performance improvement, mainly when an application executes in isolation. In another scenario, the same parameter indicates the performance degradation when co-scheduled applications cause conflicts for shared resources, and miss rate increases, which means we need to pay attention to the phenomenon, which causes performance degradation.
Further, in a situation, the performance improvement in terms of IPC; the higher value of bandwidth parameter could be beneficial for the performance in a scenario when bandwidth has a higher requirement, and it is provided to the needy applications. In another situation, it might be that the bandwidth is available to the application, but due to co-schedule characteristics/conflicts, it is remaining underutilized.
In conclusion, the Parameter behavior and its interpretation change as per the characteristics of the application and their coexistence with other applications. There seems to be a serious need to have a precise classification of applications combination and correct interpretation of the parameters in distinct scenarios. This proposition is important, which would address the said observation in a holistic manner for which a new scheduling policy would be required.
Therefore the accurate interpretation of the system parameters would help in mitigating the effect of application/co-schedule [13] interference, which affects the Multi-core system performance severely. The interference effects are low-level effects whose identification would require a detailed simulation of application co-schedules and analysis of the system parameters.
It means appropriate parameters like bandwidth consumption, data bus utilization, and the number of branches is required to extract from the co-schedules. And a detailed analysis would be required on the parameter behavior. Thus Parameters and their correlation would be useful to have a holistic view of the interference.
The simulation and analysis of the designated co-schedules would require a specialized policy through which the interference scenario could be created by executing the applications, first in isolation and then in combinations. Here we consider an execution environment in terms of the processor cores, L1 and L2 cache size, bandwidth size, type of core used for the simulation, and a process to execute the benchmarks in a varied situation, at length, to see effects on the parameters of the co-schedules.
In this research work, our contribution is mentioned below-

- We have simulated the Mi-bench applications in In-order and Out-of-order-core in isolation and measured system parameters.
- We simulated the applications in the form of co-schedules with varied sizes (2, 4 and 8) under In-order and Out-of-order cores. And measured the system performance in terms of performance metrics, Weighted IPC and Average normalized turnaround time (ANTT)
- We have done a critical analysis of application behavior when executed in isolation and an intended combination.
- We have explored the possibility of mapping the conflicting co-schedules with the appropriate policy, which might diminish the interference effects.
- Finally, we have prepared a roadmap to automate the interference removal process through a holistic policy scheduler.

The rest of the paper is organized as, in section-II, we have done the critical review of the work done in the profiling of benchmarks and related performance measurement process, section-II has covered the simulation setup and performance metrics used for the evaluation of results. In section-IV, we have presented the simulation results in terms of graphs. Section-V describes the critical analysis of results found in the profiling. Finally, we have concluded the research work in section-VI.
In the next section, we have surveyed profiling approaches.

II. REVIEW OF PREVIOUS WORK

To find the application's behavior their characterization is essential. Various research efforts have made in this direction. One of the most popular approaches for characterizing the benchmarks is through the simulation tools. The popular application characterization tool, which is used for many years in computer architecture research, which has an active user community, is the gem5 [14]. The gem5 simulation tool could characterize the applications in SE (System emulation) and Full mode. In SE mode, executable binary is profiled using the ISA of the host machine through system calls. Whereas, in Full mode, the applications are executed using the actual ISA and disk image of the target architecture. Gem5 supports two types of processor core known as-In-order and Out-of-order [15]. In-order core executes the applications in the program order, whereas the Out-of-order core does not consider the program flow and executes an instruction whose data is available.
Using the gem5 simulation tool, various researchers have done the characterization of applications in isolation as well as in the intended combination (multi-programmed mode). Mi-bench, SPLASH-2, and PARSEC [16][17][18] are well-known Open source benchmarks used for many years in Multi-core architecture research.

A. Characterization in Isolation

There are some conventional policies like increasing the L1 and L2 cache size mentioned in the literature to perform the simulation and analysis of interference effects in the system and analyzing the effect on the performance. The analysis revealed that increasing the cache size reduces the conflicts for memory bandwidth and shared L2 cache, which in turn mitigates the interference effect to some extent and increases the performance [19]. In the same line, to find the characteristics of applications in isolation, Mi-bench applications are simulated on X86 and ARM processors considering the in-order and Out-of-order processor cores [20]. The performance metrics like energy consumption, CPI, L2 miss rate, bandwidth utilization, etc, are measured after the simulation. These metrics have used to find the application behavior.

The simulation results detail important system parameters responsible for application behavior. However, the analysis is limited to the single benchmarks and does not consider the interference effect when multiple benchmarks execute simultaneously.

In another context, some studies revealed that the L1-cache parameter does not have a significant impact on the performance. However, the Last level cache partitioning is identified as a vital source for performance degradation, which is further proved by the experiments done on micro-benchmarks.

Finally, it is verified that partitioning the last level cache in applications interference environment could not always beneficial for performance improvement. [21]. To find the correlation of interference effect with memory access latency characterization of cache behavior is carried out through the reuse distance histogram model [22]. L2 cache miss rate is measured by considering L1 cache configuration as input and applying random and LRU cache replacement policies in the gem5 simulator. The profiling results are compared with the histogram model, and the accuracy is calculated through the absolute error.

To explore the behavior of media applications on heterogeneous systems, some experiments are carried out on the Gem5-gpu simulator [23]. The approach used for finding the application characteristic is conventional; some attributes of the system like cache and processor clock frequency are changed and the performance of the system is measured.

In other work, for addressing the interference issue, analysis through cache coloring approaches is also carried out [24]. Cache coloring is an effective approach to model the interference situation. Also, the said policy ensures that the data and instructions of different processor cores should be placed in the different cache sets. However, it requires special allocators for the distribution of instructions and data to make it independent. Also, accessing the data distributed through this policy in parallel is very difficult.

In the same line, to find the alternate of SRAM, comparative analysis of SRAM and STT-MRAM cache memories using the SPLASH-2 benchmarks is performed in the gem5 simulator [25]. The architecture considered for the simulation is X-86 and ARM. The purpose of the benchmark to quantify how SPLASH-2 benchmarks behavior changes considering the said memory types.

For finding the interference effects in real-time systems, the worst-case execution time is considered an effective approach [26]. The worst-case execution time helps on finding the 'severe interference situation' in real-time Multi-core systems, which determines the peak value of slow down an application would feel on the conflicting environments. This helps in estimating the minimum resource requirement of an application that executes in the mixed time-critical application execution environment.

In another context [27], for finding the behavior of gadgets like mobile phones it is found that we don't have sufficient and suitable programs to test the power and performance. Some research work prepared the benchmarks for mobile phones and integrated them with the gem5 simulator. The benchmarks include the typical user inputs like a scroll, typing, etc. The benchmarks are very effective when applications are used in isolation; however, how applications would behave in a concurrent way is the topic to be explored.

In the same line, the characterization of Smartphone applications in terms of CPI parameters is carried out using the simpoint and gem5 tools [28]. The simpoint is beneficial for verifying the performance of the applications which are in the testing phase and needs to be tested through a list of distinct design parameters. The simpoint and full simulation results are compared in terms of absolute error and it is found that the simpoint approach is having approximately the same accuracy as the full simulation.

B. Characterization in Multi-programmed environment

In the above section, it is found that most of the work of benchmark characterization considers the applications in isolation which covers the interference among the applications itself for the shared resources. However, in real-time systems, there are various applications run in a concurrent way.

To find the behavior of applications in terms of interference, running concurrently on Multi-core systems, some research efforts have made in the literature.

To find the behavior of the application on conflicting multi-programmed environment EEMBC MultiBench are executed concurrently in 64-many core architecture [29]. The simulation considered the single thread and multiple threads on each workload. The simulation explores the bandwidth saturation scenario and its effect on L1 cache miss penalties. In the mentioned experiment it is found that in many-core architecture increasing the number of concurrent workloads and threads are beneficial for the performance in terms of IPC.

In the same line, bandwidth utilization analysis is carried out to ensure the quality of service for the Multi-programmed applications, co-scheduled in the Multi-core environment using gem5 simulator [30]. The analysis done through the various bandwidth partitioning approaches like square root, proportional and the performance is measured through the multi-programmed performance metrics like harmonic weighted speedup, minimum fairness, weighted speedup, and sum-of-IPCs.

Various simulations carried out in gem5 simulator covers state-of-the-art applications like No-SQL, big data, etc. [31]. In these simulations, a comparative analysis of No-SQL benchmarks is done in terms of system parameters related to the memory hierarchy. The analysis explores the behavior of data-intensive applications using the gem5 simulator in full system mode. The main objective of the analysis is to know whether the characteristics of No-SQL applications are similar to the well known parallel benchmark applications like SPEC, PARSEC, and NAS. The said benchmarks similarity in terms of system parameters might be used to utilize the existing optimization approaches of well-known benchmarks to the contemporary database applications. The results show that despite having the distinct specification of No-SQL databases they all show uniform behavior for the cache hierarchy. Analysis of memory interferences in terms of memory access delay for the parallel memory operations performed in the Multi-core systems [32]. The analysis considers that each processor core could generate a parallel memory request for the DRAM controller. The parallel request for memory is possible due to approaches like non-blocking, speculative execution and Out-of-order instruction execution.

These approaches are advantageous in terms of throughput, however, generates very high pressure for the main memory. The simulation and analysis are carried out through the gem5 simulator using SPEC2006 benchmarks to find the worst-case performance in terms of delay.

Some researchers have emphasized on Interference analysis of shared L2 cache, by finding the nature of schedules in the real-time environment [33]. The analysis provides a maximum value of cache parameters for each application in a schedule, which indicates the interference effects on an application to explore the possible performance degradation. The upper bound interference parameter is further used for making scheduling decisions. For finding the upper bound interference value of an application, the integer linear programming is used which further helps in making the iterative algorithm.

In the next section, we have discussed various performance metrics that are used in the literature for the application's characterization and analysis.

C. Metrics for Performance analysis

To perform the analysis of application interference, various metrics are proposed in the literature. Metrics are based on two perspectives-the users and the system [34]. User perspective metrics consider the response of individual applications in terms of the slowdown. System-level progress is measured through the system throughput; measured in terms of weighted speedup [35].

Which performance metrics are suitable for a workload has been an issue of debate for a long time in a multi-core research community [36]? Performance metrics accuracy/clarity depends on the type of environment we are executing. For example; time-critical application deadline is the prime concern and thus deadline is an appropriate metric whereas for normal application system throughput is considered for performance measurement.

For single-threaded workload, IPC and CPI are the important metrics. However, for multi-threaded workload IPC and CPI are not suitable performance metrics as if some benchmark threads were busy on time-consuming program constructs a falsy interpretation may have resulted. For example, an application may get stuck in a loop.

In the same line, system-oriented metrics that were used earlier measures the total time the processor was busy and did not consider the fairness among co-executing applications, load balancing, etc. Another interesting metrics for the performance measurement is Fairness used to find the slowdown of applications in the Multi-programmed scenarios. The system could be termed as "Fair" if the rate of performance degradation (slow-down) of all the applications in Multi-programmed environment is similarly considered for their execution in isolation [37].

D. Profiling Approaches

In the review, it is found that profiling is proved to be an approach, which uses a hardware or software instrument to quantify and measure the performance of an application and obtain the behavior of the application in isolation as well as comprehensively. There are two popular approaches for profiling the benchmarks; full simulation and simpoint analysis [38]. The full simulation profiling approach performs the profiling process for the whole benchmark which requires weeks or months of time for some benchmarks, especially to SPEC benchmarks [39]. Another approach of profiling is to find the simulation points in a benchmark and profiling them in full mode. Other parts of the benchmark are not considered and they are just fast-forwarded. In this work, we simulated the Mi-bench benchmarks completely as they take a reasonable time. For the execution of each benchmark, it takes a maximum of five hrs. As compared to SPEC2006 benchmarks which take two to three days for the completion. Since the situation is always complex in real life, & would not fit into a logical scenario, we propose a proactive approach to know about the behavior of the application in a given environment in the next section.

III. METHODOLOGY

For finding the applications conflicting behavior, we have performed the profiling of applications in two ways- First in isolation and second in an intended combination which we termed as the co-schedule. For profiling, state-of-the-art gem5 simulator has been used. The input for the simulation we used is well known Mi-bench benchmarks that are suitable for embedded devices. The simulation process produced the stat.txt files which contain various system parameters like L2 cache miss rate, bandwidth utilization, energy consumption (and others which are not considered here). The schematic view of simulation steps is shown in Figure-1.

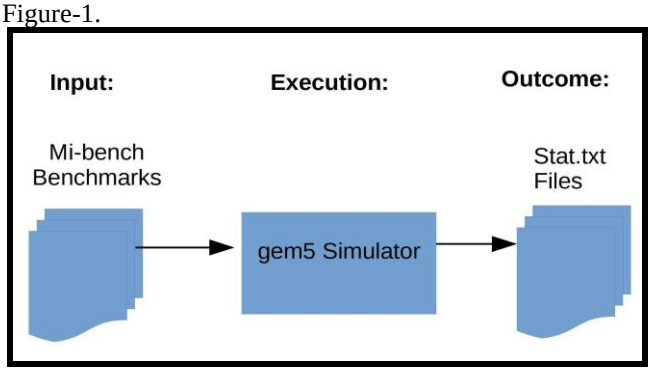

Fig.1. Schematic view of profiling Steps

In the next section, we have explored how to form a group/co-schedule which would further help in creating the interference scenario.

A. Group/Co-schedule Forming Process

In order to create and quantify interference situations in multi-programmed environment, five benchmarks of Mi-bench benchmark suite are selected. These benchmarks are Dijkstra, Basic math, qsort, Bitcount, and Patricia. Each benchmark has distinct characteristics in terms type of instructions they contain. Each benchmark has the intuitive function which they perform during the execution. For example, most of the instructions in Dijkstra benchmark are CPU bound and it performs the task of finding the shortest path among the given set of nodes. The above-mentioned benchmarks executed under X86 ISA (Instruction set architecture). Also, the simulation has been carried out in gem5 simulator. Experiments performed using in-order (Minor-CPU) and Out-of-order cores [40]. Gem5 does not support the MinorCPU processor core option in the default setting for X-86 ISA (in the list of CPU-Type supported for X86 ISA). To get that option for the simulation we have done the modification on gem5 source code particular to some files under the build folder.

Each benchmark binary is executed completely for the input specified in the benchmark suite. For the experiments, we prepared the python script and executed it in the command line in gem5 SE mode.

Benchmarks are executed in isolation mode under the reference architecture (System configuration is detailed in the next section). The benchmark groups which we also refer to as co-schedule is a possible distinct application combination. We formed three types of groups each has 2, 4, 8 applications. If in a group we have n applications then that group is termed as co-schedule size-n. In this way, a group that has 2 applications termed as co-schedule size-2. The reason for having the size of the group in the power of two is that the actual processor core where these groups would execute has the same convention for the number of cores.

Moreover, having five applications, five simulations have carried out in isolation mode. In the same line, considering the group size-two, and have five benchmarks, there could be a total of ten distinct combinations, where each combination has a single instance of a benchmark. In the same way, considering the group of four, have the same number of total benchmarks mentioned in the previous case, there would be a total of five combinations with respect to an instance of each benchmark.

After the simulations for group-size 2 and 4, we explored the group size of 8 hoping more intense interference effects.

Since in a real system there could be any combination of applications (single/multiple instances) as a candidate in a co-schedule for the execution. We moved to a new application combination where we have considered the multiple copies of the benchmarks and measured the interference effects.

We prepared the workloads (co-schedule) considering single-core, dual-core, quad-core, and octa-core processors. The benchmarks we have taken for the experiments are single-threaded; so each application is mapped to a core and it could not switch to other core during the simulation.

For all the simulations we prepared a separate python script which is executed in the terminal of Ubuntu 16.04 (open-source operating system). Also, the default configuration script "se.py" which exists in the gem5/config/example/se.py directory is considered for specifying the parameters in the experiments.

Table-I: Benchmark groups forming process

	Executing Alone	Application Combination(2)	Application Combination(4)	Application Combination Multiple copies(8)	Application Combination(Multiple copies)
	A	AB	ABCD	AAAABBBB	AA
	B	AC	ABCE	AAAACCCC	AAAA
	C	AD	ABDE	AAAADDDD	AAAAAAAA
	D	AE	ACDE	AAAAEEEE	BB
	E	BC	BCDE	AAAAAABB	BBBB
		BD		AAAAAACC	BBBBBBBB
		BE		AAAAAADD	AABB
		CD		AAAAAAEE	AACC
		CE			
		DE			
Total	5	10	5	8	8

The process of group forming is shown in Table-I. The alphabets A to E designate to a benchmark for an easy understanding of the process. Having a total of five applications, all the possible combinations are formed for the group size-2, 4 and 8. In the next section, we have presented the simulation environment created for application profiling through the process mentioned above.

IV. EXPERIMENT SETUP

In this section, we detailed the reference machine in terms of system parameters, type of processor core used for the simulation. Also, the benchmarks, their basic characteristics & the purpose are also elaborated. The co-schedule size, simulation procedure used is summarized briefly.

A. Co-schedule size & Number of core

In the experiments, the co-schedules are mapped to the processor cores as per the availability of the number of cores; typically we used 8 cores. The co-schedule size (possible combination size) depends on the number of cores. This means, if the total number of cores in a Multi-core system is four then the maximum sizes of the co-schedule would be four. However, a co-schedule whose size is less than four could also be mapped to the four cores having a condition that the co-schedule size must be in the power of two. However, if in the command line, the co-schedule whose size is greater than the number of core (parameter), default configuration file "se.py" does not allow to perform the experiment and ask for the modification on the script. For performing such experiments would require modifying the se.py script and need to enable the SMT mode on it.

B. Simulation Procedure

First, we run the benchmarks in isolation and measured the performance in terms of system parameters. In isolation mode, all the shared resources like L2 cache are fully available which could be used by the benchmark. After that, we executed the benchmarks in possible combinations to find the interference effects on the performance due to their co-execution and conflicts for the resources.

C. Reference Architecture

For the simulation the reference machine is prepared whose configuration in terms of system parameters is presented in Table-II. For the simulation, we used the two processor core types- In-order and Out-of-order which is supported in the gem5 simulator. The separate L1 cache is taken for storing the data and instructions. To create the interference situation among the processor cores, the unified L2 cache is used. In the same way, other parameters like cache associativity, cache line size, and bandwidth size are selected.

Table-II: Reference architecture for the experiments

Host Machine	**CPU**		**Cache Configuration**				
	CPU Type	**Core**	**L1-I/D**	**L2**	**Assoc**	**Line Size**	**Band width size**

Quad-core CPU,/Ubuntu-16.04	In/Out order Core	1/2/4/8	32 KB	64 KB	L1/L2-2	64	1280 Mbytes

D. Benchmarks

For the experiments, the Mi-bench benchmark suite is used. The benchmarks are executed in isolation and in the form of co-schedules. The basic functions and the characteristics of the benchmarks are provided in the Table-III

Table-III: Mi-bench applications

Benchmarks	Benchmark description
Dijkstra	Dijkstra is an application to find the shortest path between nodes in a graph. This benchmark creates a large graph in the form of the adjacency matrix and calculates the shortest path between every pair of vertices
Basic Math	Basic Math benchmark performs simple maths calculation for small embedded devices which is not having dedicated hardware for these calculations. Calculations like finding square root etc. are performed in this benchmark. The input for this benchmark is a list of predefined constant.
qsort	qsort benchmark sorts an array of string in ascending order using the qsort algorithm. The data set for this benchmark is a list of words.
bitcount	Bitcount benchmark is used to test the processor, how efficiently it counts the bits in an array. The input for this benchmark is an array of numbers that contain the 1's and 0's. Bitcount benchmark uses 5 different bit counting algorithms for counting purposes.
Patricia	Patricia is a data structure which is used in the computer network to represent routing tables.

E. Workload

To find the interference effects on performance, a comprehensive workload is prepared through the Mi-bench benchmark suite. The workload is prepared considering the applications co-existence aspects like isolation, combination, single & multiple instances. The workload in terms of co-schedules is detailed in Table-4.

Table-IV: Mi-bench Benchmarks Workload

Mi-Bench Applications Co-schedules				
	Executing Alone	**Application Combination(2)**	**Application Combination(4)**	**Application Combination(8)**
A	Dijkstra	1. Dijkstra, Basic Math AB	Dijkstra ,Basic Math,qsort,Bitcount, ABCD	qsort(4) Dijkstra(4)
B	Basic Math	2. Dijkstra, qsort AC	Dijkstra, Basic Math , qsort, Patricia ABCE	qsort(4) BasicMath (4)
C	qsort	3. Dijkstra, Bitcount AD	Dijkstra,Basic Math,Bitcount,PatriciaABDE	qsort(4) Bit-count(4)
D	Bitcount	4. Dijkstra, Patricia AE	Dijkstra, qsort ,Bitcount, Patricia ACDE	qsort(4) Patricia(4)
E	Patricia	5. Basic Math,qsort BC	Basic Math,qsort, Bitcount, Patricia BCDE	qsort(6) Dijkstra(2)
		6. Basic Math, Bitcount BD		qsort(6) BasicMath (2)
		7.Basicmath, Patricia BE		qsort(6) Bitcount(2)
		8. qsort, Bitcount CD		qsort(6) Patricia(2)
		9. qsort, Patricia CE		
		10. Bitcount, Patricia DE		
Total	5	10	5	8

Initially, the single instance of the workload is executed to find the behavior of the applications using the in-order and Out-of-order core. For creating more intense interference effects the applications are profiled in the Out-of-order core using the multiple instances of the benchmarks.

F. Performance Metrics

The performance metrics used for the performance analysis are enlisted and described in Table-V. For simplicity, here we assume that there are only two applications executing in the multi-programmed environment. We have denoted these applications as A and B. Applications are first executed in isolation mode and then in Multi-programmed mode. With the help of the performance metrics, the performance of co-schedules in terms of progress and slowdown is calculated. The Sum-of-IPC metrics which is also termed as IPC throughput is calculated through the simple arithmetic Sum-of-IPCs. Weighted IPC is the ratio of IPC in Multi-programmed mode and IPC when applications executed in isolation mode. The weighted speedup is the arithmetic sum of Weighted IPC of all the applications executed in the system. The harmonic mean is the sum of reciprocal of weighted IPC. The Average normalized turnaround time ANTT used to find the applications slowdown which they feel in multi-programmed mode compared with their execution in isolation. Weighted Speedup is used to find the progress of the applications when they executed in Multi-programmed mode compared with the isolation mode.

Table-V: Performance Metrics for the performance analysis

S.No	Performance metrics	Description	Remark
1.	**IPC**	$IPC = \frac{Total\ No.\ of\ instructions\ committed}{Total\ Cycle}$	**Higher value is better**

2.	**IPC Throughput**	IPC-A + IPC-B	**Higher value is better**
3.	**H-Mean of IPC'S**	$\frac{2}{((\frac{1}{IPC-A})+(\frac{1}{IPC-B}))}$	Here two denotes the total number of applications
4.	**Weighted IPC**	$\frac{IPC\ of\ applications\ in\ multi-programmed\ mode}{IPC\ of\ applications\ in\ isolation}$	**Higher value is better**
5.	**Weighted Speedup**	Weighted IPC-A Weighted IPC-B	**Higher value is better**
6.	**H-mean of Speedup**	$\frac{2}{((\frac{1}{IPC-A})+(\frac{1}{IPC-B}))}$	**Lower value is better**
7.	**STP(System throughput)**	Weighted Speedup	System throughput is used to measure the performance in system level
8.	**ANTT**	$\frac{1}{H-Mean\ of\ Speedup}$	**Lower value is better**

V. RESULT & ANALYSIS

In order to find the applications behavior in-depth, (in isolation & in some combination) it is essential to have their general characteristics in terms of the type of instructions and branches they contain. The general characteristics of Mi-bench applications are shown in Figure-2. In the Figure, it could be noted that Bitcount application has the highest number of CPU bound instructions as compared to other benchmarks. On the other hand, among all the benchmarks, the qsort benchmark contains the higher number of memory-bound instructions. Dijkstra benchmark contains the highest number of conditional branching instructions. The higher value of CPU bound and memory-bound instructions show the CPU & memory involvement respectively during the execution of the benchmark.

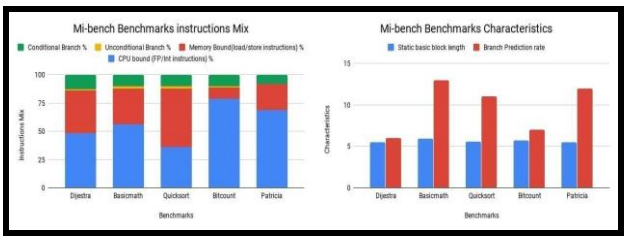

Fig.2. Mi-bench benchmark instructions mix [16]

This information would also be useful to analyze the behavior of these applications when executes in some intent combination. In the same line, the Basic Math and Patricia benchmarks have a higher branch prediction rate. The branch predictor rate plays a vital role in IPC improvement. If the branch predictor does not predict the branches accurately, the IPC of the system reduces due to the branch miss-prediction penalty. Branch miss-prediction causes flushing the miss-predicted instructions in the pipeline which was assumed to be executed in the near future. The static basic block length of Basic math and Bit count benchmarks are higher which indicates that these benchmarks do not have nested branches in between the instructions. The higher value of basic block length helps in fetching multiple instructions (exploiting instruction-level parallelism) in a cycle and further improves the system throughput in terms of IPC **[41]**. Patricia benchmark has a random data access pattern which makes it vulnerable to the higher miss rate for the L2 cache. The higher cache associativity might help in reducing the miss rate of Patricia benchmark.

A. Application profiling on In-order CPU

In this section, we have discussed the experimental results obtained when the applications are executed in In-order CPU. Applications are executed for generating their profiles in fixed reference architecture (Refer Table-2 for the configuration). The objective of the experiment is to find the general behavior of the application when it is executed in isolation and further in an intended combination.

Test Case-I Applications profiling in Isolation mode

Figure-3 shows the results (in terms of system parameters) obtained after profiling the applications in the reference architecture, in isolation mode. Here, the Dijkstra application enjoyed less miss rate and lead to a conclusion that it does not have many memory-bound instructions. Further, it does not have many conflicts among instructions for the shared L2 cache as the data is available in the cache and it does not have to access the main memory for the data, which means it has less bandwidth load(due to less required) compared to other benchmarks which we have run in subsequent sections. Then, the Basicmath benchmark has almost similar behavior as Dijkstra benchmark. qsort benchmark shows the higher miss rate as it has a higher number of memory-bound instructions [Refer Figure-2].

Another observation of having a high value of miss rate shown in the result is that the number of L2 cache replacements is higher due to conflicts for the same cache line. Further, the higher value of bandwidth engagement indicates a number of memory operations in the execution of this benchmark.

Patricia benchmark has shown higher bandwidth consumption as compared to other benchmarks, although it has a lower miss rate; the reason is that the instruction count of Patricia benchmark compared to other benchmarks is higher which creates high demand for bandwidth. Also, the higher branch prediction rate (indicates the scope of parallelism in a benchmark) reduces the value for L2-cache miss. The Bitcount benchmark shows strange behavior; it is showing a higher miss rate and it has higher CPU bound instructions compared to other benchmarks.

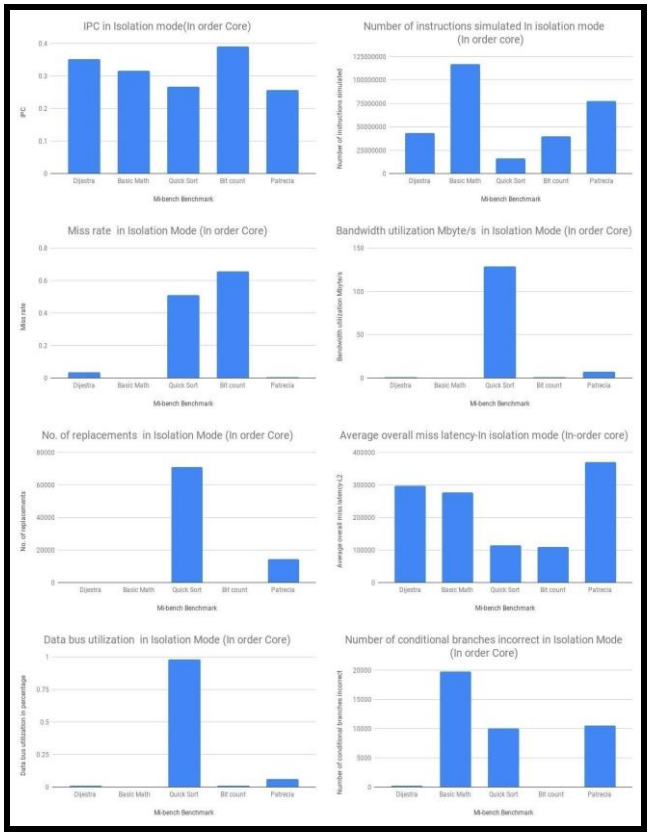

Fig.3. Mi-bench applications Execution in Isolation Mode (In-order Core)

The reason for such behavior is that the branch prediction rate of Bitcount benchmark is slightly low as compared to other benchmarks that restrict the parallel execution of bit counting algorithms (instructions). In the same line, other parameters like IPC, data bus utilization indicates that qsort benchmark suffers mostly in terms of performance. Higher Branch prediction rate and large basic block length **[16]** of Basic math benchmark have provided it parallel instruction execution opportunities which improve the overall IPC, compared to other benchmarks.

Test Case-II: -Application profiling on In-order core with Co-schedule Size-2

In this test case, we have discussed the results when the co-schedules of size-2 executed on the reference machine and have compared it with the results of the isolation mode. The variation in IPC, when the applications executed in isolation and in Multi-programmed mode, is shown in Figure-4. All the co-schedules are showing similar behavior in the presence of their partner benchmark. All the Co-schedules (numbered 1 to 10) have similar IPC when they are executed in isolation mode and after that in Multi-programmed mode. The performance of co-schedules in Multi-programmed environment is also presented through the Sum-of-IPC performance metrics in Figure-4. The Sum-of-IPC metric shows that the benchmark applications run in combination which has higher IPC at execution are performing better in terms of IPC. Specifically Co-schedules 3, 6, 8, 10 have shown such behavior where Bit-count benchmark exists. On the other hand, the weighted speedup and ANTT performance metrics indicate that all the applications have made the same progress, considering their execution in Multi-programmed mode as well as in isolation mode. The results, as expected indicates that in the in-order core, application execution in the form of co-schedules does not show significant interference effects for the shared resources. The reason for such behavior is that the In-order CPU core executes the application instructions in the same order as they appear in the application.

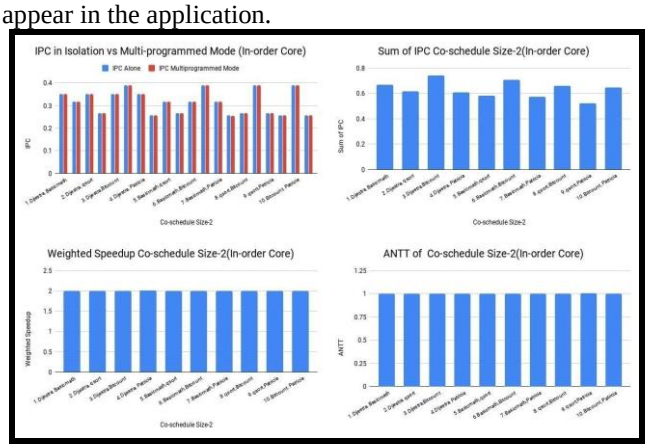

Fig.4. Mi-bench applications Execution Co-schedule size-2 (In-order Core)

Test Case-III Application profiling on In-order core and Co-schedule Size-4

In order to find the interference effects and their impact on performance for co-schedule size and number of cores taken as to four is shown in Figure-5. It may be noted that Co-schedules is not showing any noticeable difference in IPC in Multi-programmed mode with respect to isolation mode. To observe even the slightest behavior of the applications we have calculated the Sum-of-IPC. It is observed that Co-schedule ABCE has the lowest Sum-of-IPC value when applications are executed in the multi-programmed mode. The reason for such behavior is that, among all the co-schedules, this co-schedule has a higher number of memory-bound instructions (collectively) which resulted in a higher number of L2 Cache replacements (Figure-6). In other co-schedules, the bitcount benchmark, which contains a higher number of CPU intensive instructions, contributes to increasing the sum of the IPC performance metrics for all. However, this case has considered performance only for multi-programmed mode and thus, it is again proved that sum-of-IPC performance metrics favor those co-schedules which contain benchmarks of higher IPC.

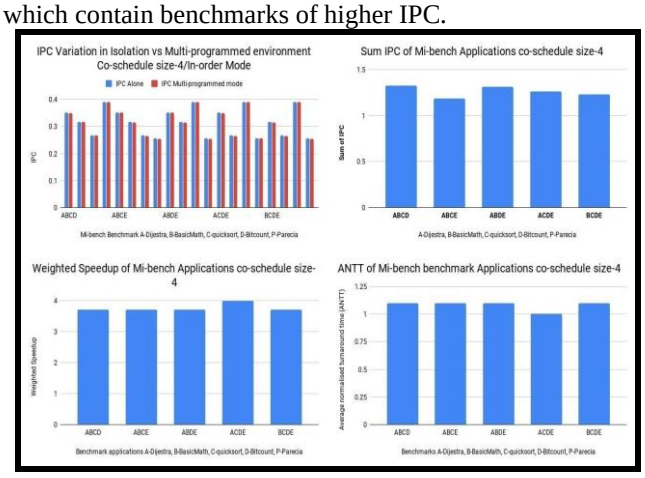

Fig.5. Mi-bench applications Execution Co-schedule size-4 (In-order Core)

Retrieval Number: B2922129219/2019©BEIESP
DOI: 10.35940/ijeat.B2922.129219
Journal Website: www.ijeat.org

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

To know the behavior of the applications in terms of relative progress by considering IPC in isolation also, we have another set of results presented in the next section.

It becomes obvious that the performance results obtained through Sum-of-IPC previously favors the higher IPC benchmarks. However, results when considering weighted speedup demonstrate that in the co-schedules ACDE; have shown higher progress as compared to other co-schedules. On the other hand, co-schedules (ABCD, ABCE, ABDE, and BCDE) have shown a slight interference effect as they might suffer for shared resources. ANTT performance metrics which is capable to observe the minor variation in IPC in multi-programmed mode compared with isolation mode is also indicating that the co-schedule ACDE has low slow-down.

This means, in all the co-schedules except ACDE, there are some applications that have not got a fair opportunity to get the shared resources or they have not utilized the available resources optimally and resulted in slow progress in Multi-programmed environment compared with their execution in isolation mode. We found the variation in performance for the co-schedules ABCE and ACDE for knowing the accurate reasons for such behavior we presented the system parameters for all the co-schedules in Figure-6.

It could be observed that in co-schedule ACDE has higher bandwidth & data bus engagement. Also, it has a huge number of L2 replacements. These parameters indicate that co-schedule ACDE has utilized the resources (bandwidth) efficiently and resulted in the higher weighted speedup. But in the co-schedule, there is an application(s) (most probably qsort) which contributes to a large number of L2 replacements. In the same way, other co-schedules which shows higher miss rate and relatively less progress is due to some applications in the schedule which contributes to the positive or negative effects.

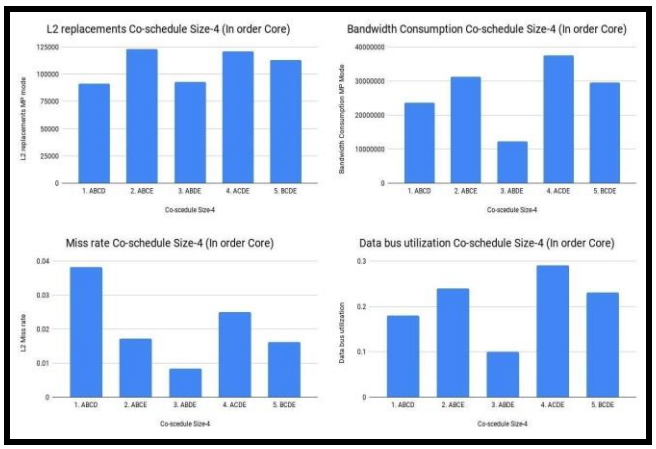

Fig.6. System parameters of Co-schedule size-4 (In-order Core)

The results discussed above are summarized as:

- The interference effect among the applications is slightly lower when the applications are executed in the In-order core.
- It is noticed that the co-schedule which contains some benchmarks having higher IPC as their virtue show significant improvement in overall IPC in Multi-programmed mode.
- The higher value for weighted speedup is the indication of better resource utilization and less interference.
- In most cases, qsort and Bit count benchmarks influence the co-schedules.
- qsort benchmark increases the miss rate and the Bit count benchmark increases the Sum-of-IPCs of the workloads in the multi-programmed environment.
- Bitcount benchmark influences the co-schedules in terms of sum-of-IPC and weighted speedup as it contains a higher number of CPU intensive instructions.
- Profiling results of co-schedule size-2 and 4 have concluded that we need not have to further profile the co-schedules having size 8, 16, etc. as the co-schedules are not showing much interference effects.
- The results shown above conclude that for executing these co-schedules the general policies like increasing the core frequency would be appropriate for increasing the Weighted speedup. However, how the general policies would behave in terms of energy consumption is the scope for further analysis.

To observe the interference situation in-depth, a more conflicting multiprogram environment would be required. For creating such an environment, we have done the profiling of co-schedules on Out-of-order CPU. The Out-of-order CPU core executes the applications in non-program order. It means the application instructions whose operands are available could be executed without following the program order. The Out-of-order CPU is expected to create a more conflicting environment compared to the In-order core for the shared resources. The results of the benchmark profiling for Out-of-order are presented in the next section.

B. Applications profiling on Out-of-order CPU

In this section, we have discussed the profiling results obtained when benchmarks/applications are executed in the Out-of-order core.

The Out-of-order CPU which executes the instructions on non-sequential order allows instruction execution on a cache miss also. The co-schedules are executed in the same reference machine which we used for the In-order core for application profiling.

Test Case-I Applications profiling on Out-of-order CPU in Isolation mode

In order to have an understanding of the system parameters when applications are executed in the Out-of-order core, the simulations have been carried out. and the results are presented in Figure-7.

To perform the analysis of application characteristics in terms of system parameters in both the CPU cores (In-order and Out-of-order), Figure-7 depicts the parameters for both the cases. It could be noted In Figure-7, that the miss rate for the qsort & Bitcount benchmark is high and for other benchmarks, its value is comparatively low.

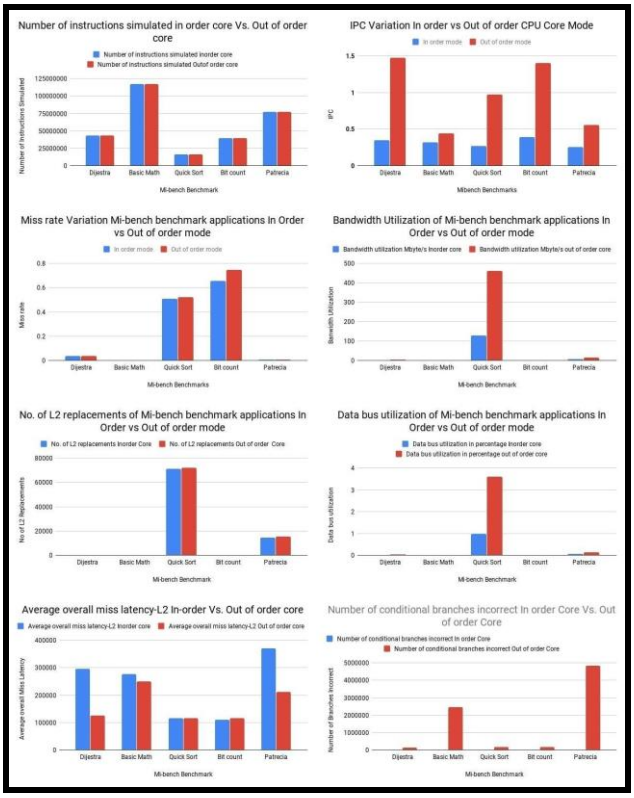

Fig.7. Mi-bench applications Execution in Isolation Mode (In-order vs. Out-of-order core)

The IPC of the applications in the Out-of-order core is higher compared with the In-order core for all the benchmarks. Among all the benchmarks it is observed that the qsort application has low IPC value in the In-order core which is drastically improved in the Out-of-order core.

The reason is that in the Out-of-order core, the qsort benchmark instructions have better utilized the available bandwidth, whereas in In-order core, due to the restrictions of instructions order, it was underutilized. Basic math application has negligible miss rate whereas Bit count and qsort benchmarks instructions have shown higher conflicts for resources. On the other hand, the Bandwidth utilization in Out-of-order CORE is higher as compared with the In-order-core for all the benchmarks. The reason for higher consumption of bandwidth is that in the Out-of-order core during the cache miss events the processor core, instead of waiting for the data, starts executing other instructions whose data is available.

The total number of conflicts for the L2 cache is higher in In Out-of-order core for the qsort and Patricia benchmark. The results, L2 conflict, and the Bandwidth utilization confirm that the qsort benchmark has higher memory-bound instructions & they conflict for the L2 cache and the bandwidth. The average overall miss latency for most of the benchmarks is reduced when they executed in the Out-of-order core.

The branch prediction accuracy is reduced as in the Out-of-order core as the instructions which are ready are executed without considering the branch predictor results.

It is observed that for the qsort benchmark, system parameters value is very high, especially for bandwidth and data bus utilization which has made the other benchmarks parameter values not clearly shown in Figure-7. The exact value of system parameters is listed in Table-7 & 8 for In-order and Out-of-order core respectively.

Table-VI: Mi-bench Benchmarks System Parameters in Isolation Mode (In-order Core)

Mi-bench	IPC	Data bus utilization in percentage	Miss rate	No. of replacements	Bandwidth utilization Mbyte/s	Number of conditional branches incorrect	Average overall miss latency-L2	Number of instructions simulated
Dijkstra	0.351636	0.01	0.036057	89	1.096509	238	296536.2595	43182447
Basic Math	0.316479	0	0.001162	43	0.451634	19742	277211.1111	117007252
qsort	0.267617	0.98	0.510532	71086	128.56106	10024	115079.2031	16102506
Bit count	0.390914	0.01	0.656999	5	0.67017	35	109769.594	39534109
Patricia	0.256628	0.06	0.004873	14436	7.181053	10492	370826.4149	77237151

Table-VII: Mi-bench Benchmarks System Parameters in Isolation Mode (Out-of-order Core)

Benchmark	IPC	Data bus utilization in percentage	Miss rate	No. of replacements	Bandwidth utilization Mbyte/s	Number of conditional branches incorrect	Average overall miss latency-L2	Number of instructions simulated
Dijkstra	1.474416	0.04	0.035881	81	4.645751	136026	125374.4701	43182446
Basic Math	1.0114	0.01	0.001958	78	1.503625	2396127	250752.4825	117007216
qsort	0.974652	3.6	0.522453	72282	461.740467	177150	116690.569	16102505
Bit count	1.404736	0.02	0.747126	1	2.362736	152762	116581.7308	39534392
Patricia	0.558768	0.13	0.006437	15260	16.029211	4839041	212653.3339	77237150

The results are summarized:

- Benchmarks are showing different behavior in Out-of-order core as compared to the In-order core in terms of system parameters like miss rate, bandwidth utilization and the number of L2 cache conflicts.
- Benchmarks are showing remarkable progress in the Out-of-order core for IPC performance metric in isolation as compared with the In-order core. The reason for this is due to proper CPU cycle utilization when the data of any instruction is not available..

- In results presented above, it is found that the qsort benchmark has higher value of bandwidth utilization and L2 cache conflicts; concludes that there is a higher possibility the qsort benchmark would dominate other benchmarks in multi-programmed environment.

The summary presented above concludes that benchmarks execution in Out-of-order mode has significant potential creates a high conflicting environment. This motivates us to further explore the execution of the benchmark in the form of co-schedules. In the next section we have presented the results when applications are executed in co-schedules of size-2, 4 and 8 in Out-of-order mode.

Test Case-II Application profiling on Out-of-order core and Co-schedule Size-2

The simulation results in this case, are showing significant interference for the shared resources as compared with the in-order core having the same co-schedule size. The results are presented in Figure-8. Co-schedules are showing notable variations in IPC in Multi-programmed mode with respect to isolation mode. The variation shows that applications IPC is decreased when they have run in Multi-programmed mode. In all the cases, co-schedule-8[qsort, Bit count] has shown significant performance loss. In Co-schedule-8 the bitcount benchmark conflicts to qsort benchmark for the resources in multi-programmed core. The conflict is due to the randomness in the instruction selection and execution by the Out-of-order CPU core.

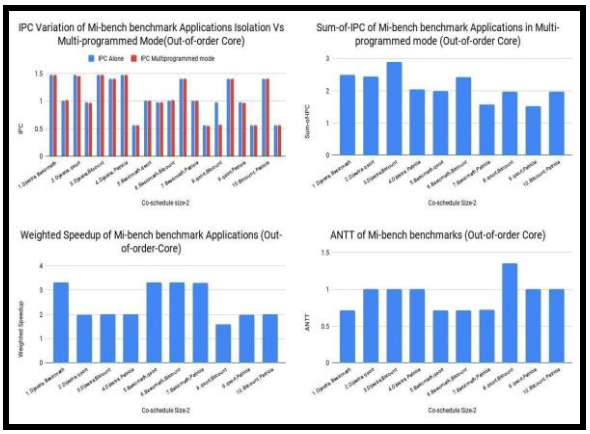

Fig.8. Mi-bench applications Execution Co-schedule size-2 (Out-of-order Core)

Sum-of-IPC performance metrics is favoring those co-schedules which have higher IPC benchmarks. It is not clearly indicating that co-schedule-8 have done less progress. The weighted speedup performance metrics clearly show that the co-schedules 1, 5, 6 and 7 have made higher relative progress and other co-schedules (2, 3, 4, 9, and 10) have shown moderate performance except co-schedule 8. Co-schedule-8 contain the qsort benchmark, and it is observed in previous cases, the schedules which have this application have made less progress as being the memory bound characteristics in it. ANTT performance metrics also indicate that the co-schedule-8 has suffered in terms of slow-down. The system parameters for Co-schedule 8 are shown in Table-9 & 10.

Table-VIII: Mi-bench Benchmarks System Parameters in Isolation core (Out-of-order Core)
IM (Isolation core), **MP** (Multi-programmed Mode)

Benchmark	L2 Cache miss(data)	L2 Cache miss(data)	L2 Cache miss(Instruc	L2 Cache miss(Instruc
	IM	MP	tion) IM	tion) MP
qsort	0.520342	0.521966	0.886493	0.900817
Bit count	0.964029	0.992933	0.692998	0.738826

Table-IX: Mi-bench Benchmarks System Parameters in Isolation Mode (Out-of-order Core)

Co-schedule-8	Data bus utilization IM %	Data bus utilization MP %	Bandwidth Utilization Mbytes/s IM	Bandwidth Utilization Mbytes/s MP
8.qsortt, Bit count	3.62	2.14	464.103203	273.973578

The system parameters in Table-9 and 10 clearly show that the applications have not got the required amount of resources in multi-programmed mode and due to that the performance is hampered.

Test Case-III Application profiling on Out-of-order core and Co-schedule Size-4

In this test case, we have attempted to analyze the interference effect on co-schedules for more number of applications compared to the previous one. The interference effects in terms of performance metrics are shown in Figure-9. In the Figure, a noticeable difference

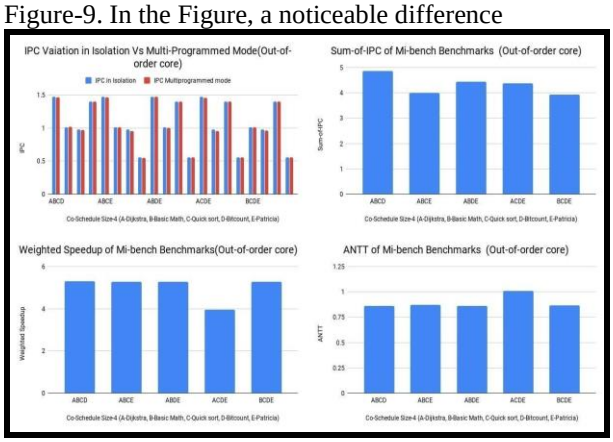

Fig.9 Mi-bench applications Execution Co-schedule size-4 (Out-of-order Core)

in terms of IPC among all the co-schedules could be observed except for the co-schedule-3, when applications are executed in the multi-programmed mode as compared to the Isolation mode. The reason for such behavior which could be observed in the Figure is that all the co-schedules which contain qsort application have faced the interference effect and in the co-schedule-3 qsort benchmark is absent and resulted in no interference among the applications. In the same line, the Sum-of-IPC metrics indicate that co-schedule BCDE has done low progress, the reason is that in this co-schedule qsort benchmark affecting the Bitcount and Dijkstra benchmark and a notable interference effect is observed. The weighted speedup indicates that the co-schedule ACDE is affected much in terms of progress and slows down respectively. Among all the Co-schedules the interference effect on Basic Math and Patricia benchmark is not recorded which is obvious as both have relatively less miss rate in Out-of-order core. The Co-schedule ACDE is affected as it contains most of the benchmarks which are affected due to the qsort benchmark.

C. Applications profiling Considering Multiple copies

In previous results, we have observed that a single copy of a benchmark in a co-schedule has not shown the notable interference effects, as a single copy of an application does not have a sufficient number of conflicting instructions to show the interference effects. To find the possibility of interference effects in the multiple copies we have done the simulations and the results are shown in Figure-10.

Test Case-I Applications profiling Multiple copies in In-order Mode

In Figure-10, it could be noted that when all the benchmarks are executed having its own 2,4 and 8 copies some benchmarks have shown the variation in IPC whereas some benchmarks remain neutral.

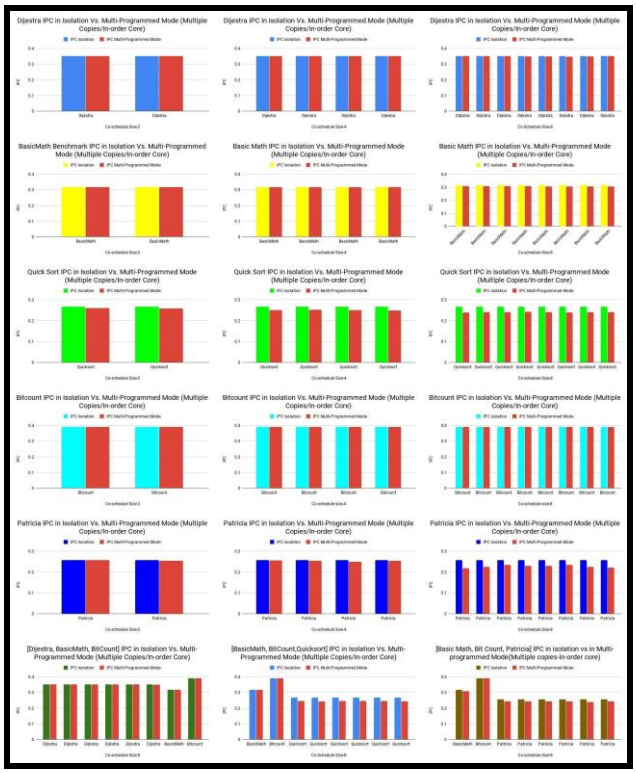

Fig.10. Mi-bench multiple copies of benchmarks Execution (In-order Core)

Dijkstra, Basic Math, and Bitcount benchmarks have not shown significant variation in IPC (interference among multiple copies). However, increasing the number of copies of Basic Math, qsort and Patricia benchmarks from two to eight shows the variation in IPC. Whereas the Patricia and qsort benchmark instructions have shown the sensitivity for the interference for the shared resources. In other words, the multiple copies of benchmarks (only a few) are affecting due to interference for the shared resources. To know the interference effect when multiple copies of two distinct benchmarks are simultaneously executed a special simulation is carried which is shown in Figure-10. For this simulation, the co-schedules are prepared by taking multiple copies of qsort, Patricia and Dijkstra benchmarks (6-copies in each case) and single copies of Basic math and Bitcount benchmarks. The purpose of

The simulation is to know the behavior of Bitcount and Basicmath benchmarks when executed with heavy conflicting partner benchmarks. However, no interference effect is observed.

The results discussed above are summarized as:-

- Multiple copies of a benchmark in a co-schedule have shown significant interference effects in In-order core, except Dijkstra, Basic Math, and Bitcount benchmarks
- Multiple copies of distinct benchmarks in a co-schedule are also not showing the interference effect.
- The results clearly indicate that some benchmarks (qsort and Patricia) which were neutral when executed considering a single copy in In-order core (in previous experiments) have also shown inherent interference characteristics.
- Interference effect could be more severe for benchmarks like qsort and Patricia in Out-of-order cores.

Test Case-II: Applications profiling multiple copies in Out-of-order core

In previous experiments, it is noted that multiple copies of benchmarks in a co-schedule have shown moderate interference effects in the In-order processor core. Previous results assure that the benchmarks which have shown the interference effects in the In-order core would repeat their behavior for Out-of-order core also. However, some co-schedules which contain benchmarks like Dijkstra, Bitcount and BasicMath have not shown the conflicting behavior in In-order core although they were executed with multiple copies of highly interference sensitive benchmarks. Thus, it is not known whether these benchmarks would behave in a similar fashion for the Out-of-order core also. In this section, we attempted to find the interference effects for the above mentioned applications which were neutral in In-order core.

In previous experiments, we witness co-schedules which contain more than two benchmarks have shown the interference effects for Out-of-order cores. Thus, in this experiment, we have done simulations and presented the interference results for the co-schedules of size-4 and 8.

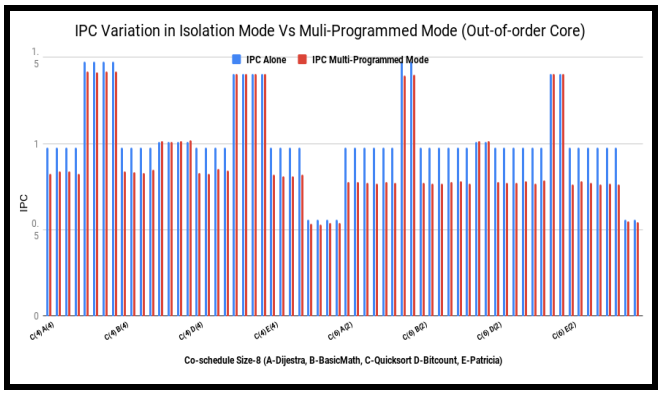

Fig.11. IPC Variation on Mi-bench benchmarks multiple copies Coschedule size-8 (Out-of-order Core)

In all the previous cases, the qsort benchmark is identified as a higher interference sensitive application for in-order and Out-of-order cores. Hence, qsort benchmark could be a suitable choice for further experiments. In this experiment, we considered qsort as a common benchmark for all the co-schedules. The co-schedules which contain multiple copies of distinct benchmarks are simulated in the Out-of-order core, variation in IPC and in-depth analysis through performance metrics is shown in Figure-11 & 12 respectively.

It could be noted In Figure-11 that four & six copies of qsort benchmark are common for all the co-schedules. Figure-11 depicts that, all the co-schedules have shown the interference effects due to the qsort benchmark. In all the co-schedules IPC is decreased when applications are executed in the Multi-programmed environment compared to the Isolation mode. The interference effect is higher for the qsort, Dijkstra and Patricia benchmarks whereas the BasicMath and Bit count benchmarks are less affected. The Bitcount & BasicMath benchmark contains higher number of CPU bound instructions, and they have higher branch prediction rate which makes them unaffected on highly conflicting environments also. Whereas the Dijkstra, qsort and Patricia benchmarks do not have higher branch prediction rate. Moreover, lower value of basic-block length of affected benchmarks has restricted to harness the parallelism potential of Out-of-order cores.

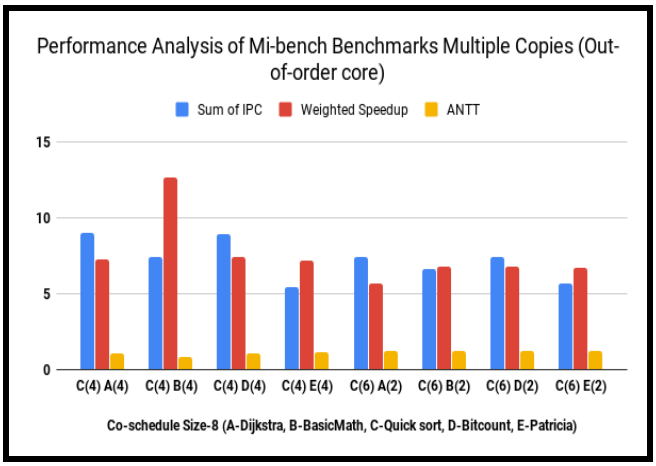

Fig.12. Performance Analysis of Mi-bench benchmarks multiple copies Co-schedule size-8 (Out-of-order Core)

In another context, the simulation results are also visualized through the Sum-of-IPC, weighted speedup and ANTT performance metrics, which is shown in Figure-12. Among all the co-schedules the sum-of-IPC performance metrics of co-schedule [(qsort (4), Dijkstra (4)] & [qsort (4) Bitcount (4)] is higher. However, these two are the co-schedules that are highly affected due to the interference. It is again proved that the Sum-of-IPC is not the fair performance metrics to measure the system throughput.

In the same line, the weighted speedup and ANTT performance metrics (as we have seen in all previous cases) are providing the different interpretations for the same set of results. The Weighted speedup and ANTT performance metrics, when the number of copies of all the co-schedules is considered as four shows that the Co-schedule (qsort (4), Basicmath (4)) has made higher relative progress and have Lower slowdown respectively.

Whereas when number of copies of the qsort benchmark is increased from four to six, the weighted speedup of [qsort(6), BasicMath(2)], [(qsort(6), BitCount(2)] and [(qsort(6),Patricia(2)]is higher, whereas the Co-schedule[(qsort(6), (Dijkstra(2))] is suffered more. In both cases one thing is common that the qsort and Dijkstra benchmarks are conflicted for the resources whereas the Bitcount, Basic Math benchmarks were safe. It means qsort and Dijkstra benchmarks are identified as very sensitive applications for the shared resources in the Out-of-order core. It could be noted in Figure-12, increasing the number of copies of qsort benchmark from four to six, the interference effect is increased. The Bitcount benchmark was neutral for the interference effects in all previous cases has repeated their behavior considering multiple copies in the Out-of-order core also. The benchmarks which were shown the interference behavior in In-order core were found more intense in the Out-of-order core.

The results discussed above are summarized as-

- Significant interference effects are observed when co-schedules are executed in the Out-of-order core considering multiple copies of the benchmark.
- Multiple copies of memory sensitive benchmarks affect all the other benchmarks in a co-schedule.

VII. CONCLUSION

The interference among applications is one of the most critical concerns for the need of performance. Thus, it is necessary to find out in what situations/scenarios the interference would impact on a higher degree. System Parameters are important internal sources to find out the real causes of interference effects among the applications which execute at run time. In this research, we have performed an in-depth analysis of interference effects among various co-schedules, which occur due to the varying characteristics of the applications. We have found the following important facts related to the shared resource contention/interference in multi-programmed environment-

- Interference effect is generally lesser on In-order CPU cores whereas Out-of-order CPU cores have shown the notable conflicts for the shared resources.
- The interference effect depends on the size of co-schedule for a given set of CPU cores. Large co-schedule size could be more prunes for interference situation.
- Sum-of-IPC performance metrics has higher value for those co-schedules which contain benchmarks of higher IPC. Weighted speedup and ANTT performance metrics were proved to be a better choice for interference measurement and analysis.
- If multiple copies of memory-bound benchmarks are co-scheduled with some CPU bound benchmarks in Out-of-order CPU core the higher interference effects are observed (Refer figure-11).
- L2 Cache parameter must be selected appropriately for the simulation setup; large L2 cache size would not reveal observable interference effect on shared resources contention.
- Multiple Copies of CPU intensive workloads do not create much contention effect.
- Multiple Copies of those benchmarks that have mixed CPU and memory-bound instructions show the contention effects for L2 Cache.
- Benchmarks have not shown similar Interference behavior for In-order and Out-of-order CPU cores. Moreover, Benchmarks like Dijkstra whose behavior was neutral in In-order CPU core has been shown higher interference behavior for Out-of-order cores.
- The Basic block length, Branch prediction rate, type of benchmark (CPU bound, Memory bound) are identified as the key system parameters.

These parameters could help in deciding the suitable policy to run the benchmarks for enhancing the system throughput.

The interference analysis carried out in this research has shown a strong need for some mechanism in the form of scheduler in an existing Multi-core system. The scheduler, considering the above discussed co-schedule execution scenarios in terms of appropriate system parameter values could decide an appropriate policy for performance enhancement. Considering five benchmark applications and their possible co-schedules in the presence of different processor core has created an application execution scenario that looks like applications are executing in the real physical Multi-core machine. In this work, we considered only five applications for interference analysis. in the future, interference effects among applications having more number of benchmarks, different co-schedules options, and for more number of cores could be explored.

REFERENCES

1. Vincent N´elis, Patrick Meumeu Yomsi, and Lu´ıs Miguel Pinho. The Variability of Application Execution Times on a Multi-Core Platform. In Proc. of WCET, 2016.
2. Alexandre Kandalintsev. 2016. Application Interference in Multi-Core Architectures : Analysis and Effects. Ph.D. Dissertation. University of Trento, Italy.
3. Lakshminarasimhan, S. 2015. An Efficient Architecture for Dynamic Profiling of Multicore Systems. Master's thesis. Dept. Elect. and Comput. Eng., Univ of Arizona, USA.
4. Eklöv, D. 2012. Profiling Methods for Memory Centric Software Performance Analysis.Ph.D. Dissertation.Uppsala University, Sweden.
5. Andreas Sembrant, David Eklov, and Erik Hagersten. 2011. Efficient software-based online phase classification. In Proceedings - 2011 IEEE International Symposium on Workload Characterization, IISWC - 2011, 104–115.
6. Reetuparna Das, Rachata Ausavarungnirun, Onur Mutlu, Akhilesh Kumar, and Mani Azimi. 2013. Application-to-core mapping policies to reduce memory system interference in multi-core systems. In Proceedings - International Symposium on High-Performance Computer Architecture.
7. Sergey Zhuravlev, Sergey Blagodurov, and Alexandra Fedorova. 2010. Addressing shared resource contention in multicore processors via scheduling. In Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems - ASPLOS '10, 129.
8. Lavanya Subramanian, Donghyuk Lee, Vivek Seshadri, Harsha Rastogi, and Onur Mutlu. 2016. BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling. IEEE Trans. Parallel Distrib. Syst. 27, 10 (2016), 3071–3087.
9. Zhihua Gan, Mingquan Zhang, Zhimin Gu, Hai Tan, and Jizan Zhang. 2017. Delay analysis and optimization for inter-core interference in real-time embedded multicore systems. J. Parallel Distrib. Comput. 103, (2017), 77–86
10. Myonghoon Oh, Jongmoo Choi, Seong-je Cho, Jeesoo Kim, Changhwan Youn, and Woosuk Chung. 2018. Analyzing and modeling the impact of memory latency and bandwidth on application performance. 1095–1101
11. Nitin Chaturvedi and Gurunarayanan S. 2013. Study of Various Factors Affecting Performance of Multi-Core Processors. Int. J. Distrib. Parallel Syst. 4, 4 (2013), 37–45
12. Gerd Zellweger, Denny Lin, and Timothy Roscoe. 2016. So many performance events, so little time. 1–9. DOI:https://doi.org/10.1145/2967360.2967375
13. Xiaoya Xiang, Bin Bao, Chen Ding, and Kai Shen. 2012. Cache conscious task regrouping on multicore processors. In Proceedings - 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2012, 603–611.
14. N. Binkert et al., "The GEM5 Simulator," SIGARCH Computer Architecture News, vol. 39, no. 2, May 2011.
15. Anastasiia Butko, Rafael Garibotti, Luciano Ost, and Gilles Sassatelli. 2012. Accuracy evaluation of GEM5 simulator system. In ReCoSoC 2012 - 7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip,
16. M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. In Proceedings of the 4th Work. on Workload Characterization, pages 83–94, 2001.
17. Steven Cameron Woo, Evan Torriet, Jaswinder Pal Singh, and Anoop Guptat. 1995. The SPLASH-2 Programs : Characterization and Methodological Considerations and Approach Axes of Characterization Approach to Characterization. In ISCA '95 Proceedings of the 22nd annual international symposium on Computer architecture, 24–36.
18. Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In Proceedings of the 17th international conference on Parallel architectures and compilation techniques - PACT '08, 72
19. Vikas B."On the Cache Behavior of SPLASH-2 Benchmarks on ARM and ALPHA processors in Gem5 Full System Simulator" in 2014 3rd International Conference on Ecofriendly Computing and Communication Systems
20. A. Abudaqa, et al., "Simulation of ARM and x86 microprocessors using in-order and Out-of-order CPU models with Gem5 simulator", 2018 5th International Conference on Electrical and Electronic Engineering (ICEEE), 2018
21. H. Yun, P. Valsan, "Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Platforms", Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT), 2015.
22. J. Ge and M. Ling, "Fast Modeling of the L2 Cache Reuse Distance Histograms from Software Traces," 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Madison, WI, USA, 2019, pp. 145-146.
23. S. Sawal and N. Guinde, "Performance evaluation using GEM 5-GPU simulator", 2017 International Conference on Computing Methodologies and Communication (ICCMC), 2017.
24. Mandalapu, Srini. "White Paper on Issues Associated with Interference Applied to Multicore Processors." (2016).
25. A. Arya, "A Comparative Study of Cache Memories Based on MRAM and SRAM Technologies", 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS), 2018
26. F. Reghenzani, G. Massari and W. Fornaciari, "Mixed Time-Criticality Process Interferences Characterization on a Multicore Linux System", 2017 Euromicro Conference on Digital System Design (DSD), 2017.
27. S. Fan, B.C. Lee, "Evaluating asymmetric multiprocessing for mobile applications", Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software, pp. 235-244, April 2016.
28. D. Sunwoo et al., "A structured approach to the simulation, analysis and characterization of smartphone applications", IEEE International Symposium on Workload Characterization (IISWC), 2013.
29. Chen, C. et al., "Profiling EEMBC MultiBench Programs in 64-core Machine. EEMBC MultiBench Profiling" White Paper. 2013.
30. R. Wang, L. Chen and T. Pinkston, "An Analytical Performance Model for Partitioning Off-Chip Memory Bandwidth," 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, 2013
31. A. Colaso et al., "Memory Hierarchy Characterization of NoSQL Applications through Full-System Simulation", IEEE Transactions on Parallel and Distributed Systems, vol. 29, no. 5, pp. 1161-1173, 2018.
32. H. Yun, R. Pellizzon and P. Valsan, "Parallelism-Aware Memory Interference Delay Analysis for COTS Multicore Systems", 2015 27th Euromicro Conference on Real-Time Systems, 2015.
33. Xiao, S. Altmeyer and A. Pimentel, "Schedulability Analysis of Non-preemptive Real-Time Scheduling for Multicore Processors with Shared Caches", 2017 IEEE Real-Time Systems Symposium (RTSS), 2017
34. S. Eyerman and L. Eeckhout, "System-Level Performance Metrics for Multiprogram Workloads", IEEE Micro, vol. 28, no. 3, pp. 42-53, 2008
35. Stijn Eyerman and Lieven Eeckhout. 2014. Restating the Case for Weighted-IPC Metrics to Evaluate Multiprogram Workload Performance. IEEE Comput. Archit. Lett. 13, 2 (2014), 93–96.
36. Pierre Michaud. 2013. Demystifying multicore throughput metrics. IEEE Comput. Archit. Lett.
37. Hans Vandierendonck and André Seznec. 2011. Fairness metrics for multi-threaded processors. IEEE Comput. Archit. Lett. 10, 1 (2011), 4–7
38. Lieven Eeckhout. 2010. Computer Architecture Performance Evaluation Methods. In Synthesis Lectures on Computer Architecture, Morgan Claypool

39. A. Limaye and T. Adegbija. 2018. A Workload Characterization of the SPEC CPU2017 Benchmark Suite. In 2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 149–158
40. S. Padmanabha, A. Lukefahr, R. Das, and S. Mahlke. 2017. Mirage cores: The illusion of many Out-of-order cores using in-order hardware. In Proceedings of the Annual International Symposium on Microarchitecture, MICRO, 745–758
41. Basic block: 2019. https://en.wikipedia.org/wiki/Basic_block.
42. Main Page: http://gem5.org/Main_Page.

AUTHORS PROFILE

Surendra Kumar Shukla is currently associated as a Research Scholar with the School of Computer Science & IT, Devi Ahilya University, Indore, India. He is B.E in Computer Engineering from SGSITS College Indore, India. M.E in Computer Engineering from IET, Devi Ahilya University, Indore, India. He has worked with different engineering universities in a span of 15 years in the Department of Computer Engineering. His research area is Multi-core architectures, Parallel Computing.

Dr. P.K. Chande is working as a Chairman C'S MIND-a startup to think beyond AI Indore, M.P. India. He was Ex. Director MANIT Bhopal, SGSITS Indore, Visiting prof. Japan & Director MB&T MPSEDC. He has 2 co-authored books and has published more than 80 research papers in international/national journals. He has Pursued Research in areas like Fault-Tolerant Systems, Real-Time Knowledge Systems, Intelligent Automation, Smart Vehicular Systems etc. He has guided 9 Ph. D. scholars.