International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-7, Issue-2, Feb 2019 E-ISSN: 2347-2693

Investigating Policies for Performance of Multi-core Processors

Surendra Kumar Shukla[1*], P.K. Chande[2]

[1]School of Computer Science & IT, DAVV, Indore, India
[2]SVKM's ,NMIMS, Deemed to be UNIVERSITY, Indore, India

[*]*Corresponding Author:* surendrakshukla21@gmail.com *Tel.: +7987748990*

DOI: https://doi.org/10.26438/ijcse/v7i2.964980 | Available online at: www.ijcseonline.org

Accepted: 15/Feb/2019, Published: 28/Feb/2019

***Abstract*—** Performance is a critical concern of multi-core systems. There are some issues which affect the performance of multicore systems especially shared resource contention and application to core mapping. To address the performance issues various software and hardware-based policies are proposed in different works of literature. These policies address the particular performance issue through some specific approach in isolation. However, having many performance issues and the corresponding number of policies to solve the issues; it is not clear which policy would be beneficial for a particular situation for application execution. There is a need of investigation & classification of existing policies through various aspects like the approach used to address the performance issues, tools used for profiling the application and metrics used to find the source of performance degradation. The classification of policies could help make static and runtime decisions for addressing different performance issues which arise owing to resource allocation and contention. In this paper, we reviewed various policies employed for performance improvement of multicore systems. Policies like the application to core scheduling, memory allocation, bandwidth allocation, parameter tuning & self-awareness are investigated on various angles and resulted in an in-depth classification which is conferred from the tables. Further, classification could be used to design a holistic policy scheduler which could schedule a policy considering the application workload characteristics in totality. Also, the scheduler could help on performance improvement through scheduling/switching the appropriate policies at run time for application execution while considering the system status.

***Keywords*—**Investigation, Multi-core, Parameter, Policy, Performance.

I. INTRODUCTION

Multi-core architecture is a growing trend today as the single-core processors rapidly reach the physical limits of the performance [1][2]. Now Multi-core processors are being used in various areas such as Virtualization, High Performing Computing, Database, and Cloud and also in many gadgets nowadays[3]. There are a diverse set of applications which run on these systems. Each application runs on the multi-core system has different characteristics and different criteria for their execution, but all have a common goal; high performance. A time-critical application wants to meet the deadline; memory sensitive application demands the higher memory bandwidth & CPU oriented applications would have CPU the priority. At runtime there, execution criteria contradict each other & lead to various performance issues related to the thread & core mapping, resource allocation, etc. [4][5][6][7].

To address various performance issues, most of the research focuses on conventional software or hardware approaches[8][9]. Hardware-based approaches have the advantage of speed whereas software-based approaches are comparatively slow [10]. Hardware-based approaches use FPGA based reconfigurable architectures. Reconfigurable architectures use core fusion approach [11]to transform the homogeneous cores to superscalar[12] dynamically. This approach accelerates the execution pace of non-parallel part of the application. However, such reconfigurable policies have chip design issues like the form factor. It would also make energy consumption & power dissipation critical.

On the other hand, software-based approaches tie-up with the OS, and invariably use profilers & observing run time behavior of task to address performance issues[13][14]. OS uses schedulers, memory mapping schemes for application execution. OS developers focus to regular applications and cannot customize it for a diverse set of applications which would run on multi-core systems. The software-based approaches may be available to handle slow, but complexity is avoiding the use of extra hardware on the chip.

In the same line, resource availability is also a critical factor

for performance. Earlier researchers had attempted resource allocations issues through static allocation approaches. However, static methods are not sufficient as applications have different resource requirements at different phases in the execution[15]. Considering application phase change in concern with resource requirements at runtime require analyzing application thread behavior at run time. Dynamic approaches need a sampling of threads after some intervals, creating sampling overhead & switching core at the run-time lead to switching cost [16][17]. To address the sampling overhead, some mapping policies rely on on-line monitoring of CPI stack metrics termed as "bias". The application is mapped to the "bias" which fulfills resource requirements in run time[18]. One step ahead, CPU utilization based mapping approaches calculate the CPU utilization of a thread in each core and further compares with all the cores[19]. A thread which has highest CPU utilization is mapped to the fast core and vice-versa.

Conventionally, research addressed resource allocation strategies to address shared resource bottleneck for raised performance. Resources should be allocated to applications having in-depth knowledge of their characteristics & available resources competency. Most of the performance problems like contention occur due to improper thread-to-core-mapping, unbalanced shared resources' allocation, and utilization like cache & main memory [20]. Improper allocation creates the problem of synchronization & locking and would severely affect the performance.

In brief, the issues related to performance are correlated with each other. Hardware & software solution to one issue would force to trade-off the performance due to another parameter. For example, a performance issue like data locality to reduce the latency could affect the shared cache contention [21].In the following section, we present a contemporary study on various proposed research.

Rest of the paper is organized as follows; Section II contains a general overview of contemporary approaches employed for performance improvement. Section III contains the application to core mapping policies, Section IV contain memory allocation schemes and its effect on shared resources contention, Section V contains the cache optimization policies, section VI explores the bandwidth allocation issues and related policies, Section VII addresses the parameter tuning issues for performance enhancement, Section VIII contains the importance of self-awareness & related policies, Section IX summarize the policies, and Section X concludes research work with future directions.

II. CONTEMPORARY RESEARCH

The new ways of application execution have promised improvement in performance. However, the execution of an application is still conventional owing to its design. The application designers, do not keep the system architecture in perspective. So, the application could run on any system, and the performance would depend on the system itself, which means the application runs without having any system input so far as performance is concerned. It seems to be a gap which can help improve performance if appropriately addressed. We can imagine some feedback mechanism from the system to the application and vice-versa which can provide an advantage for performance tuning[22].

Looking forward, for application to core mapping, some researchers have used state-of-the-art Hardware performance monitoring counters[HPC] for measuring & monitoring essential events like LLC miss penalty, memory capacity utilization & stall cycles waiting for the memory [23]. The HPC is the latest trend for performance monitoring & malware detection in ARM architectures. However, studies reported that HPC has accuracy issues; if performance related events are not appropriately measured.

Some research on reservation based memory bandwidth allocation has been carried out for fair distribution of memory bandwidth to the applications. Hardware-based controller, named "memory guard", has been proposed which ensures minimum bandwidth to all applications, and in turn helps on application execution in isolation. [24]

Unpredictable runtime interferences of applications also affect the performance. In such cases, researchers have proposed a parameter tuning based policy named "code instrumentation & auto-tuning". The proposed policy works by putting tuning instructions in between the hot spot(part of the code in application taking much time for execution) & with the help of the feedback reduces the execution time [25]. However, the applications need to know the tuning parameters.

On the same lines, for improving the performance design-time parameter tuning policy is proposed [26]. The policy could suggest the appropriate parameter values to individual applications which might be run in the embedded system. Some authors have emphasized that the application communication is a critical factor for performance improvement as it creates shared resources contention issues. To reduce the application communication pattern policies based on reuse distance and pipeline are proposed[27][28]. In the reuse distance approach, an application memory access pattern is analyzed to find out the possibility of applications intercommunication in the future. In the pipeline approach, at first applications is portioned into coarse-grained regions and these regions are cascaded in the form of pipeline. Pipeline approach reduces the communication overhead compared to fine grain parallelism. Also, the Pipeline based tuning approach is

fruitful for state-of-the-art Java applications, however, are not beneficial to legacy applications (procedure oriented).
For reducing the interference between the applications which causes many performance issues, a concept of the logical cluster[29] for NUMA multi-core systems is proposed. Hence, the logical cluster is created among the cores separated through the interconnect & the application threads are scheduled among the logical cores, in place of physical cores. It mitigates the contention for the memory controller but increases the latency to access the data.

To handle the performance issues in some different manner, few conceptual models were proposed by the researchers-like "partner core". Partner core is a small, personal assistant, tightly coupled to the CPU, comes with the general core; responsible for handling general issues related to performance & free the central core to concentrate on the application.[30]

To solve the performance issues related to data & task assignment in multi-core systems, some researchers proposed 'on-chip' main memory named "Scratch-pad memory (SPM)"[31]. As data transfer between off-chip main memory & SPM done through the software; it would be beneficial to handle the performance issues related to cache at programmer end, contrary to the on-chip cache memory hardware controlled. The scratch-pad memory has benefits to mitigate the cache contention effect [32], meeting the deadline for real-time applications; also useful for embedded systems. However, it is not used in mainstream desktop processors.

On the same lines of performance optimization of multi-core systems, an effort is made on improving the performance through cache optimization. Policies used for cache optimization are based on cache affinity & parameter tuning[33]. Affinity reduces the warm-up time of cache however have not shown much improvement on LLC[34]. On the other hand, tuning based policies adjust the value of system parameters which are responsible for application execution delay; few researchers attempted it through hardware tuners. However, cache tuning parameters exhibit the circular tuning dependencies [35]; which means tuning the cache for an application & respective core, affects the behavior of caches of other cores.

Moreover, to address the circular tuning dependency approaches based on heuristics, like application classification(using cache misses obtained at runtime) & profiling is proposed [36]. However, these approaches have not a much significant impact on the performance of state-of-the-art multi-core architectures.

To optimize the cache for the performance improvement of multi-core systems some research done on Shared cache partitioning[37]. Shared cache has two portions, one for private data of threads & another for shared data. The benefit of this scheme is that it separates the cache evictions of a different category of data in shared cache & reduces the miss rate.

For performance improvement & saving the energy, scheduling LLC and DRAM are essential factors; few traditional & contemporary research addressed this issue. One state-of-the-art approach named "orchestration" for scheduling the LLC and DRAM is also proposed [38]. In LLC miss scenario it is required to fetch the data from the DRAM, however improper scheduling of LLC miss to DRAM row buffer wastes lots of energy. Orchestrated scheduler & unified memory controller help on identifying the correct row buffer in the DRAM bank to charge & save energy. Having separate controllers for LLC cache & for the DRAM created the visibility issue of data traffic caused by the LLC miss, establishes the scope for the unified controller.
To save the energy in multi-core systems, cognitive computing based scheduling approach is proposed[39]. Hence the task is categorized into three categories- Big, medium & low energy tasks. Also, the processor cores are classified as the big, small, medium as per the frequency on which they operate. Mapping of task-to-core is done on one to one basis; the big task is mapped to the big core. The advantage of the proposed approach is saving energy. This approach is suitable for energy savings. However, the researchers have not discussed much on performance issues.
Researchers have proposed an approach to solving the performance problems autonomically by matching the problem with the fault scenario[40]present(maintained with the system). This autonomic feature of fault handling is saved by carefully tracking the execution path of the application & noting the events related to the performance problem. If the same scenario repeats then match the situations with saved solutions. However, this approach is not using application characteristics to handle the faults; it is just a static approach to match the unsolved problems with the model had encountered, the same kind of problem in the past. To address the performance issues, machine learning based policies are also employed for resource allocation & auto-tuning. To speed up the execution of scientific applications which uses stencil codes machine learning policies have shown improvement[41]. Also, the issue of dynamic requirement of the resources and availing the resources at runtime on-line reinforcement learning based approach is proposed which addresses resource mapping issues dynamically[42].

To enhance the performance & optimize resource management attempt has been made using architecture level modeling languages[43] to knowing resource requirements & their influence on the performance in advance, runtime

approaches like on-line prediction may be useful. However, to implement prediction based approach successfully some challenges like, how to predict at what stage of application will change its behavior for the resources, what will be the effect on the system performance after the changes & what may be the set of solutions to tackle the changes. For addressing the challenges mentioned above, model-based approaches may be beneficial[44][45]. The model-based approach requires to describe the resources needed, performance requirements using the modeling languages.

To improve the performance researchers are also moving towards autonomic computing based policies. Autonomic systems could have a list of self* properties like- self-adapting, self-diagnosis, self-governing, self-organized, self-recovering. However, in an existing scenario, neither the operating system nor the system elements like main memory have such autonomy. Some researchers have attempted for performance and security issues through self aware policies[46][47][48][49][50][51][52].

To improve multi-core system performance, Self-awareness, nature-inspired & state-of-the-art energy savey policies could be beneficial [52][53][54]. In conventional systems, an application executes through some predefined policies available with OS. Also, application & system does not have much interaction in the course of execution. There is a need of investigation, that If the application could also participate in the performance improvement process, start providing their input to the system related to the performance issues; whether we could improve the system performance.

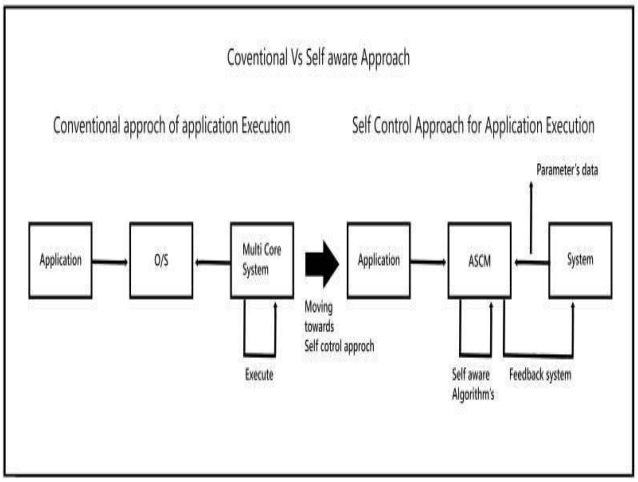

Fig.1. Conventional vs Self-aware approach for application execution

The figure-1 contain a conceptual model, here ASCM (Application self-control manager) is a hypothetical interface which could help on implementing the self-awareness policies. With the review cited above, we could have a classification of performance improvement policies depicted in figure-1.

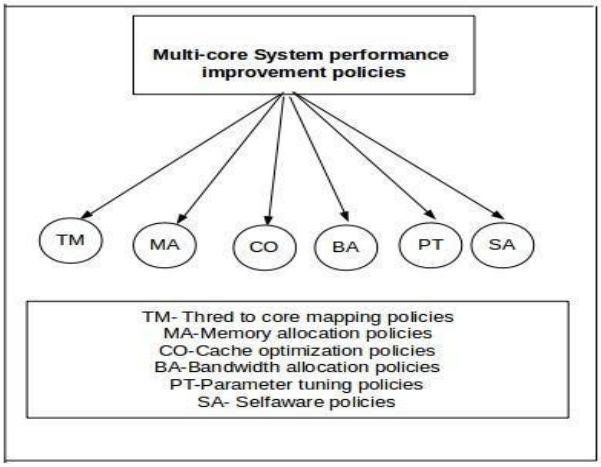

Fig. 2. Multi-core system performance improvement policies

III. APPLICATION AND CORE MAPPING POLICIES: SCOPE FOR PERFORMANCE IMPROVEMENT

In the literature review cited above, we have seen that there are various factors which affect the system performance and different types of policies are proposed by the authors to address them. It is found that application to thread mapping policies had shown vital importance on system performance. In this section, we will discuss policies related to the application-to-thread mapping and its impact on performance.

Application to core mapping issue does not affect much to the homogeneous multi-core architectures. The OS scheduler manages the scheduling activity efficiently as all the cores are symmetric. However, in the case of heterogeneous multi-core architecture, applications have different options for cores. OS scheduler must have to be more attentive and careful to map applications to appropriate core in terms CPU bound applications could be assigned to fast core and I/O bound applications to slow cores. Application to core mapping can be made through off-line profiling, online profiling and on-line monitoring of the system parameters like cache miss, miss penalty, etc.

Application to core mapping approaches falls in two broad categories which work in a static & dynamic way. Static policies depend on off-line profiling and do not consider the application phase change during the run time [15]. Further, static methods are free from thread migration at run time and reduce the overhead of sampling. On the other hand, dynamic approaches consider phase change characteristics of application & does the thread migration by calculating

IPC of applications on different cores, finds the appropriate cores at the run time. However, frequent thread migration could cause performance degradation. Further, for the thread to core scheduling heuristic based dynamic scheduling algorithms shown significant promises. Almost all scheduling algorithms whether it is working statically or dynamically, performing on-line or off-line profiling are relying on the "application characteristics" a primary source to influence the scheduler for mapping the thread to appropriate cores.

III.I Thread to core mapping issues & approaches

To map the application-to-core, cache miss penalty based policies have shown improvement in performance[4]. It means an application assigned to a core whose cache miss penalty is relatively less. To measure the miss penalty parameter the hardware performance counters used. Hardware performance counters provide the miss penalty data to the scheduler in relatively less delay. However, selecting the appropriate performance counter among several ones available in modern CPUs requires expertise and detailed analysis.

Moreover, for application to core mapping, application architectural characteristics and their behavior could play an important role in mapping[19][16][17]. Further, an application is profiled to know the architectural characteristics through a profiler, and IPC is calculated. Then applications are mapped to the cores as per the high or low value of IPC it has to fast or slow cores respectively.

In the same line, some authors termed the run time matching of "application behavior to appropriate core" as bias[19]. Since Application "Bias" changes at run time, the scheduler can use this information to migrate the application to the appropriate core dynamically. The benefit of biased based mapping policy compare to sampling is that bias bases policy abolish the run time sampling overhead incur for monitoring CPI stack of the threads. The Biased based approach is a prominent and novel one, however, effective to heterogeneous workload only. It means, it is incompetent to the similar type of workload and requires accuracy in performance counters. In another work, thread mapping schemes use the CPU utilization of threads; primary criteria to schedule the threads to the core [18]. It means threads which spent most of its time on CPU is assumed to be eligible to be mapped to the fast CPU and threads which involve on I/O activity need to map on Slow cores. Proposed thread mapping schemes are helping to save energy. For applications performance improvement, memory latency is also a critical factor. Memory latency parameter depends on the distance between the application and its data. To reduce the distance, application and its data could be mapped close to each other, i.e., in the same cache. However, some applications interfere with shared resources like LLC and interconnect. Further, to address the interference issue among applications for a shared-resource; applications could be schedules to the septate clusters[29]. Also, applications which are network intensive could be mapped close to the memory controller. Besides, the performance, energy is also a prime concern for portable multi-core devices. Schedulers could map the threads to core concerning energy a criteria [18][39]. Another alternate of energy saving could be varying the frequency of the core as per application need for performance[5].

III.II Thread to core mapping evaluation methods

In this section, we have done the investigation of policies from some key points related to the evaluation method used by the policies. Table 2 detailed the policies by describing the tools; briefly, benchmarks used & the result obtained through the experiment. Most of the policies are using the IPC (Instruction per cycle) parameter for the scheduling decisions. It is also observed that speedup is the prime metrics used for measuring performance. The policies are summarized in TABLE-1 & 2.

Table-1 Investigating thread to core mapping policies as per criteria used. Reference

	Core to thread mapping Approach	Criteria for Scheduling	Benefits	Limitation
S. Hao [4]	Performance counters used to observe the behavior of LLC Miss penalty,	LLC miss penalty is used to know the program behavior for scheduling decision.	Fast to capture the Latency. Support load balancing.	More hardware support needed to prevent the performance loss
Koufaty D. [19]	First, find the application metric then map application to a core which satisfies the application resource requirements. An application could migrate across cores as per "bias" at run time	Application Bias Online monitoring of application metrics "bias" for scheduling it to a suitable core at the run time (Dynamic scheduling policy)	Dynamic approach Work through online application performance monitoring. It Does not require offline profiling or sampling of application.	on Suitable for heterogeneous bias and not much impact on the similar type of loads. Need special care during migration as it might cause latency issues.
R. Teodoresc u and J. Torrellas[5]	Core frequency variation as per application performance requirement. Frequency variation is done through the proposed algorithm	Core frequency power consumption	Core frequency variation control is given to the application. Saves energy	Useful for the power constraint applications only. Not tested for parallel applications.
D. Shelepov et al[15]	Profiling offline provides an application signature. Avoids dynamic profiling further avoids load imbalance	LLC Miss rate criteria is used to schedule the application to the proper core(Static scheduling policy)	Scalable Simple	Static Support for short live thread.Does not monitor the thread for their lifetime in execution
Bechhi et al. [16]]	Periodically sample the threads for core switching at run time calculate the IPC ratio.	IPC-driven (Dynamic scheduling policy)	Observing runtime behavior of the threads could help on thread migration if performance degrades.	Sampling overhead. Poor scalability If only a few cores are fast in CMP it could cause contention for fast cores. load balancing issues.
Kumar et al.[17]	Weighted speedup (IPC of thread sole/ IPC of all threads)	IPC-driven heuristic based(Dynamic scheduling policy)	Fair policy	Sampling overhead, Poor Scalability, For sampling: thread needs to migrate from one core to another which adversely affects the performance.
Anuradha P et al[39]	Used the concept of cognitive computing for the task to core mapping.	Energy requirement of threads	Cognitive-based approach energy savvy.	Supports only to ARM architecture
Das et al. [29]	For application to core mapping:- Make the distance among applications interfere with each other. For other applications: reduce the distance between application and memory controller.	Application characteristics and there effect on interconnect topology	Interconnection network traffic reduced. Memory latency reduced for network intensive applications.	Ranking of applications as per the sensitivity for the network may result in unfairness. Applications which are most sensitive for the network could suffer low down.
T. M. Birhanu [18]	Fastest thread fastest core approach	CPU utilization (High CPU utilization thread mapped to the fast core and Low to slow.	The proposed approach does not consider prime concern "frequency" for achieving heterogeneity	Starvation. All the application may demand fast CPU.

Table-2 Investigating thread to core mapping policies as per system configuration

Reference	Parameters used	Metric used	Tool & Benchmark Used	Result Obtained
S. Hao [4]	LLC miss penalty, miss rate	Speedup	Godson-3 RTL simulator and SPEC2000 Benchmark	32% & 18.4 % performance improvement compared with existing scheduling algorithms.
Koufaty D., [19]	Instruction window size, Request per kilo instructions	CPI Stack	Supermicro, X8DTN board with Intel Xeon processors.	Average 9% speedup compared with the stock scheduler.
R. Teodorescu and J. Torrellas[5]	IPC, Frequency	Average power, & frequency of the core, throughput	SESC Simulator & SPECint, SPECfp benchmarks	Increased average throughput 12-17%
D. Shelepov et al[15]	IPC, reuse distance, cache miss rate, memory access latency	Relative completion time of applications, execution time, stall time	Real machines, PIN, MICA, DVFS, SPEC CPU2000 suite.	13% speedup compared with Solaris static scheduling approach
Bechhi et al. [16]]	IPC, Switch duration, switch loss	Speedup is calculated with Global instruction count/ Execution time	Two configurations EV5, EV6 is built for M5 Simulator. SPEC2000 Benchmark used.	Mapping policy benefitted by 20% to 40% compared to its counterpart random one.
Kumar et al. [17]	system queue length, Job arrival rate	Weighted Speedup, Average response time of applications	SPEC2000 benchmark used. SMTSIM simulator used. A simpoint tool is used for fast forwarding.	Mapping Policy gets 31% improvement over random scheduler.
Anuradha P et al[39]	Process time	Total energy consumed, CPU time	SPEC 2010 Benchmark & ARM7 architecture.	The accuracy of scheduling algorithm. 98%.
Das et al. [29]	The degree of application interference teach other, L2 Cache MPKI	Stall time per miss (STPM), Memory access per kilo instructions (MPKI)	Cycle level CMP Simulator	Improves system throughput by 16.7%, Reduces system unfairness by 22.4 %.
T. M. Birhanu [18]	Cache Size, frequency, issue width	Power consumption, throughput	PARSEC benchmarks & ESESC Simulator used.	Compared with CFS Scheduler the results:- speedup 52.62%, Save Power 2.22 %

Iv. Memory Allocation & Contention Issues & Policies for Performance Improvement

In the previous section, we have seen that the allocation of application to appropriate core affects the performance of the multi-core systems. However, allocation of a suitable core for the application alone could not accelerate the performance of the system. In addition to this, thread to core mapping policies must have to consider the structure of memory and issues related to allocation, especially in NUMA multi-core architecture [21]. Overlooking the memory structure will adversely affect the system performance concerning latency and finally would affect the system performance.

IV.I Data locality factor for memory allocation

An important factor which affects the multi-core system performance is the time spent on availing the data to the application. The required data for the application during execution must be provided incurring without delay. In general, data could exist either in main memory or in the cache memory. An important parameter related to the availability of data is termed as data locality. Data locality refers to the memory of the memory hierarchy on which level the data is located when asked by the processor. Ideally, the required data should be available in the cache to avoid latency to access at a deeper level in the memory hierarchy. Data locality parameter affects the access time of the data in the different level of memories. By memory distribution, there are two types of architectures named UMA and NUMA. Memory latency parameter affects both the architectures. In UMA architecture main memory is shared by all the cores and accessing the data in memory incurs same time for all the cores. However, in the case of NUMA, memory is divided into parts; and distributed to cores. Time to access the data is determined through the distance of the data from a particular core. It is found that UMA architecture has scalability issues for the bandwidth. It means, UMA architectures supports quality bandwidth to some limited number of cores, further increase of core creates the bandwidth issues.

To address the scalability and quality issues, NUMA architecture uses memory controllers to support a large number of cores in the multi-core chip. With the help of the memory controller, each core is assigned some part of the main memory. In UMA architecture all the data present in the shared memory is local for all the cores and takes equal time to access it. NUMA architecture has a drawback that if the data of the thread is not available in the local memory, it

should be brought remotely and it will increase the latency time and finally will impact the performance. The remote memory can be accessed through the interconnect available on-chip. However, it would cost an additional delay.

IV.II Data locality trade-off with shared resource contentions

It is noted that data locality significantly affects the performance of NUMA architectures. To address the data locality issue some approaches like memory profiles are proposed [13]. Memory profilers help on understanding the thread memory interaction pattern for NUMA architectures. Profiler creates the temporal flow graph between the threads and memory objects like LLC, L1 cache or main memory. Profiler based approaches could be helpful to the programmers for writing memory latency aware code. However, profiler based policies are not much accurate. Profiler assumes each thread to memory interaction a "thread-memory allocation" relation; which is not true for every function calls. Although, data locality aspect reduces the latency to access the data, however, there are some side-effects found in works of literature. First, as per data locality principle, all the applications would try for putting data close to the CPU ideally in the L1 cache, if not at least on LLC. However, availing data of each application in the cache will create the pressure on the cache and further create the cache contention issue. Second, cache contention will force the applications to move towards the main memory to access the data, and it will create excessive pressure on the memory bandwidth also.

IV.III Contention Issues in virtual machines

The contention issues discussed in IV.II is related to the physical NUMA multi-core architectures. However, the contention for the shared resources become more complicated in virtualized multi-core systems. Multi-core systems could have employed the virtual cores in a physical core with the help of the operating system. Two virtual cores may compete for the resources of a physical core. To address the contention issues in the virtualized multi-core system, the researches have focused on characterizing the contention issue through contention sensitivity and contention intensity parameters[6]. Also, contention interference analysis suggests that working set size (total memory availed to an application for its proper working) and memory access pattern is also critical parameters which affect the contention issue severely. Memory allocation policies are summarized in TABLE-3

Table-3 Investigating memory allocation & contention resolution policies

Approach	Criteria for memory allocation	Parameters/Metric used	Workload/Experiment done	Benefits	Limitation
Z. Majoand T.Gross[21]	-Data locality is the primary criteria for memory allocation. Data locality should also consider the contention issues during memory allocation Put the application data close to the memory hierarchy level.	Parameters used: - Cache pressure due to applications, penalty on accessing remote data. Local references vs MPKI. Metric used:-Cache miss rate per thousand instructions executed (MPKI).	- Performance vs. data locality. Tool/workload used:- 2. NUMA arch. Contain Intel Lehman processors. & each node has 4 cores which share the LLC. SPEC Benchmark used	The policy is doing best to solve the Cache contention issue causes due to improper scheduling.	The scope is there to explore applications interference effect on interconnect.
Lachaize, R et al. [13]	Remote memory access delay. -Tracing the source code for getting pattern how thread accesses the memory -Memory profiler used for knowing thread to memory access pattern.	Parameter considered:- Total no. of remote memory access	. Experiment:- Two NUMA machine used with given config. (16, 24, 28) cores. AMD Opteron, 2.5 GHz,4 core in each(total 16 cores), 32 GB RAM. 4 Applications tested: - FaceRec etc.in Linux platform.	The approach uses profiler for detecting how threads change the pattern of accessing the memory during their execution helps on proper memory allocation.	To diagnosis the thread to memory interaction patterns, manual effort. Profiler is only useful for memory bound applications
Y. Cheng et al [6]	First, identify the factors which affect the contention. Use the identified factors as an input to the performance prediction model. Use machine learning to predict performance.	Parameters: - Working set size. contention sensitivity, contention intensity Metrics:- Performance degradation	Experiments: - Done on a real machine, Used 2 processors, and Intel Xeon series. Benchmark:- NPB, SPEC2006	Shared resource contention features and performance degradation tradeoffs are analyzed which is prominent for performance improvement.	NUMA multi-core system is not considered which could be a gap.

V. CACHE ALLOCATION AND TUNING POLICIES

Cache plays a vital role in performance improvement of multi-core processors. Also, All the cores have a different level of caches like L1, L2 & L3. To minimize the latency of accessing the data from main memory large size LLC cache is employed in multi-core processors. Among all the caches incorporated On-chip, LLC impacts the performance severely. There are various approaches to allocate the cache to applications like cache affinity [34]. Cache affinity allows an application to repeatedly get allocated to a static core. The advantage of cache affinity is that it saves the application context which helps on addressing the cache warm-up issue. However, cache affinity causes the load balancing, starvation and fairness issues. To harness the impact of cache in multi-core systems various parameters needed to considered in isolation as well as in totality to get some promising optimal parameter settings [33]. Moreover, an optimal cache parameter setting could work fine for one architecture; however, it may show different behaviors for another ISA [33]. Cache behavior is affected due to its own internal parameters like cache size, associatively and line size and also due to external parameters like core frequency, bandwidth and interconnect.

Table- 4 Investigating cache tuning policies as per simulation and criteria used

Reference	Approach /Innovation used for Performance Improvement	Parameters considered	Tools/other instrument used	Benefits	Limitations	Results
V. Kazem pour et.al. [34]	Hypothesis used- Cache affinity improves the performance. Saving the context of the cache during application execution avoids further reloading it.	size of L1 Cache, Clock frequency	The experiment is done on Sun Microsystems UltraSPARC T2000 "Niagara" With eight cores and a shared L2 cache. Intel Quad-core Xeon E5320	Overhead of Cache warm-up time of L1 & L2 cache is eliminated resulting in performance improvement.	Cache affinity Could cause starvation for a particular core. Could create a load balancing issue.	Upper bound performance improvement: L2 cache- 11% (Avg.) 27% (Max) L1-Cache-Nill
N. Ramas ubram anian et al [33]	Analysis of multi-core performance through cache parameters variation. Calculated & Compared the cache access time obtained in ALPHA & X86 ISA	cache access time, miss rate & miss penalty	M5Sim, SPLASH-2 Benchmark	Different ISA is simulated & almost all cache parameters tested through M5 simulator.	Simulations results are not promising. Scope for comparative analysis of simulated and actual results.	Processor core frequency variation does not affect much execution timeof benchmarks.

VI.I Bandwidth allocation policies

In previous sections, we have seen that policies related to application-to-thread mapping & memory allocation have shown the improvement in the performance. Moreover, thread and memory allocation strategies attempt to put applications data near to the CPU during execution. However, neither scheduling strategies nor memory allocation policies could make avail complete working-set data of all thread in the cache due to limited space on the chip. Also, for availing the data which is not in the cache would require to access the main memory through the bandwidth. Since the number of cores which compete for the bandwidth is more; makes the bandwidth allocation a critical issue for performance. To address the bandwidth allocation issues considering fair and starvation free aspects, researchers attempted through hardware-based policies; memory guard & effective bandwidth utilization respectively[23][24]. All the bandwidth allocation policies broadly fall into two categories- fair and guaranteed allocation. We will discuss these policies in subsequent sections.

VI.II Effective bandwidth utilization: a source for knowing performance bottleneck

To measure the multi-core system performance, memory and

bandwidth related events should be recorded. Performance counters could help to track and save these activities. In general, it is assumed that high memory and bandwidth utilization is an indication of performance degradation. However, high memory utilization recorded by performance counters could be due to prefetching or out of order instruction execution: another way to improve performance [23]. Thus, high bandwidth utilization could be due to the out-of-order instruction executions. For the above-said scenario, it would be appropriate to calculate the actual usage of bandwidth. Actual usage could be estimated through running some benchmarks in a system which put sufficient load on bandwidth. Actual utilization of bandwidth is also termed bandwidth threshold value. Threshold value indicates the maximum effective utilization of bandwidth. It means bandwidth utilization is tested for the actual data transfer initiated through the CPU bound instructions. The actual bandwidth would be used as metrics to know whether system performance is increasing or decreasing. Measuring Effective bandwidth utilization through performance counters would indicate us actual utilization of the bandwidth and would be assumed a sign of good performance. However, in another scenario, it may be that a particular core is utilizing most of the bandwidth and some other core is trying to get the bandwidth and repeatedly fails resulted in starvation. Starvation would cause some time-critical applications delayed for the execution would affect the system in another way. To solve issues related to the starvation and fairness a novel approach is discussed in section VI.III.

VI.III Memory guard: guaranteed bandwidth allocation in isolation

To provide the minimum bandwidth to all the cores in the multi-core CPU a fair policy named memory guard is proposed [24]. A memory guard is a hardware-based approach which allocates guaranteed bandwidth to all the cores for an interval and predicts the requirement of bandwidth for the next interval. Further, cores have the opportunity to claim the bandwidth from contender cores which were not utilized the bandwidth in the previous interval. Further, as per the predictor estimate for the bandwidth, some core put updated demand for the bandwidth and some core release their bandwidth for the next interval. Also, predictor accuracy on estimating the bandwidth requirement for the subsequent interval is a critical factor for the success of the policy. However, if predictor wrongly evaluates the bandwidth budget, some core may falsy return the bandwidth and will suffer. The proposed approach works fine for soft real-time applications; however, it lacks supporting hard real-time applications. We have summarized, the discussed bandwidth allocation policies in TABLE-5.

Table-5 Investigating bandwidth allocation policies as per situation and criteria.

Reference	Approach Used for Performance Improvement	Metric Used	Tools/other instrument used	Experiment	Benefits	Limitations
D. Molka et al. [23]	1.With the help of the HPC knowing Onchip the utilization of the resources like memory & finding which memory is causing the performance degradation 2. Differentiating memory access is natural as CPU instructions are asking for it or due to some other reasons (dependency)	percentage of cycles waited for the memory hierarchy components, Memory bandwidth	Bull SAS bulkx R421E4 system.Hardware Performance Counters	Benchmark-x86-membench. PAPI API used to get the performance events in the processors	Useful to know the resources responsible for bandwidth underutilization.	Manual efforts to identify the events in performance counters related to a finding the memory issue.
Heechul et al. [24]	Guaranteed memory bandwidth for the applications. Divides bandwidth into two parts- Guaranteed, Best effort	Normalized IPC, Memory Bandwidth Usages	Per core regulator & reclaim manager	SPEC2006 Benchmark used. The approach applied in Linux kernel & tested in multicore systems	To utilize the guaranteed bandwidth reclaiming approach saves the wastage of bandwidth	Work best for soft-realtime systems. Does not support hard real-time systems

VII. Parameter tuning issues & policies

In previous sections, we have witnessed that researchers have emphasized the individual system components like memory, cache, bandwidth for the performance improvement in isolation. However, due to the diverse set of applications which run in a multi-core system considering the optimization through a particular policy type would not be beneficial. Also, the application changes its phase during execution makes it hard to predict behavior and resource requirement. Application during execution affects all the

components like main memory, bandwidth, cache.

VII.I Need for parameter tuning

Parameter setting for an application to one platform not works in another platform as the multi-core platform have wide diversity[25].

The behavior of an application in execution is dynamic, and we cannot accurately predict in prior; how the application will behave in a particular execution environment.[27]

Applications may have parameters that are dependent on each other.

Application parameters may have an extensive range of values that it may take, so it will be difficult to explore a large search space.[26]

Parameter values selected for an application will change if the same application executes with other applications. [25]

VII.II Parameter tuning policies

In general, each application contains a portion which consumes most of the application execution time is termed as "Hot-spot".To address the hot-spot issues software-based approach like auto-tuner could be beneficial [25]. Auto-tuner instruments some tuning code in between the hot spot and through variation in parameter values find the suitable number of threads which could speed up the execution through the feedback mechanism. Auto-tuner reduces application execution time.

However, auto-tuner requires tuning parameters as input for the tuning process. Tuning parameters should be exposed by the programmer in the application.

Finding hot-spot is a tedious task; it requires to profile the application and some time profiler fails to find hot-spot. Hence, if parameters and its range could be estimated at design time for each application which would execute in the system may address the issue of on-line tuning. However, for design time parameter tuning large design space need to be considered. Some heuristic based methods have attempted to explore some limited design space and provided the result close to the full design space search [26].

To address the overhead of architecture specific profiling & parameter tuning it could be beneficial if we find the application specific parameters like reuse distance and inter-thread communication once & reuse it for all the architectures where an application will run in future [27]. It is noticeable that the interaction among the threads is an essential factor that affects the multi-core system performance. To mitigate the effect of communication pipeline based policies could be beneficial[28]. In the pipeline approach, the application is partitioned in some parallel regions(coarse-grain) and executed in the cores. Further, the result of the regions is feed to the pipeline. The advantage of pipeline approach is that- it reduces the communication as we are considering the parallelism at coarse-grain level; also pipelining avoids using the memory resources to store the intermediate result would indirectly benefit on reducing shared resources contention.

In TABLE-6 we could observe the following aspects- all the strategies are focusing on application characteristics for the tuning process. Tuning is required to mitigate the interference effect of the applications for the shared resources.

Table-6 Investigating parameter tuning based policies as per situation and criteria used.

SNO	Reference	Approach Used	Innovation	Benefits	Limitations
1	Karcher,T. & Pankratius, V. [25]	Auto-tuner-Reduces applications interferences with auto-tuning the parameter values. The proposed algorithm uses Simplex heuristics for parameter tuning.	A specific portion of the application(Hot Spot) that executes repeatedly is tuned with a feedback approach.	No User involvement in the parameter tuning process. Capable of tune non- numerical applications. The tuner does the tuning at user level space at application.	Tuning parameters need to be exposed in an application for code instrumentation. Algorithm work for integer values only.
2	M. Kulkarni et al[27]	The reuse distance used to analyze the application communication behavior.	Tuning approach:- Application characteristics like data reuse and thread interaction are used.	The application need not have to profiled for each architecture for performance tuning.	Tuning process is explored with a limited number of parameters;
3	P. Kansaka r and A. Munir[2 6]	Provides Application specific design parameters for tuning.	For large design space; a proposed policy finds the design parameters in limited search space. Provides the tunable design parameter range to each application.	Need not to explore large search space for application specific design parameter tuning.	The approach considers the design parameters only; could not tune parameters online.
4	Y. Wang and K. B. Kent [28]	Data flow analysis to obtain pipeline parallelism in Java applications. Stream-graph used to detect the communication pattern.	Converting legacy application into their parallel representation.	Pipelining could reduce the communication overhead of threads and improve the performance.	Involvement of programmer to partition the program. Useful only for the object-oriented paradigm.

In TABLE-7 we have summarized the policies from the evaluation methodology used. Some policies use the simulation method for the evaluation, others using the actual machine. Also, PIN tool[46] is required in the tuning process for code instrumentation in the application to get the critical insights required during the tuning process. It is observed that if tuning would be performed considering the system parameters, it would be dependent on a particular platform and finally require to repeat the tuning process for each architecture. Instead, for tuning process parameters should be selected from the application characteristics. It is noticeable that all the policies cited and enlisted in the table considered application parameters like the number of threads, block size, file size, and reuse distance.

Table-7 Investigating parameter tuning policies through evaluation methodology used for experiments.

SNO	Reference	Parameters considered	Metric Used	Tools/Workload used	Experiment	Results
1	Karcher, T. &Pankratius, V.[25]	No. of threads, block size	CPU time, Wall clock time(Total Time user wait in front of a desktop)	Intel Core 2 Quad Q6600 processor, Online tuner.	the tuner is integrated with the Linux kernel.Bzip2 application tested in proposed tuner.	bzip2 and parallel video processing application tuning. The result is promising.
2	M. Kulkarni et al[27]	Reuse distance, Cache size, inter-thread communication.	Miss rate	PIN tool, SIMICS,NAS, SpecOMP,SPEC benchmarks	Statical Sampling is used to detect the communication pattern of threads.	Sampling analysis for reuse distance is fast 177 times than full analysis.
3	P. Kansakar and A. Munir[26]	No of cores, Frequency of processor core, CacheSize(L1, L2,L3).	Execution time, Power consumption	ESESC Simulator. PARSEC and SPLASH-2 Benchmark used.	PARSEC & SPLASH-2 Benchmarks are tested on small and large search space.	Design quality improved 1.35-3.69 percent of the result obtained from a fully exhaustive search
4	Y. Wang and K. B. Kent [28]	numbers of records, size of files.	Total time is taken for application execution.	Ubuntu 12.04 OS, Quad-core processor. BDS Simulator.Annotation classes.	JVMTI tracing agent and Annotation classes are used to detect the parallel regions.	Performance improvement:- Time is taken to executee application improves-10-48% approx.

VIII. Self-Aware based policies for performance improvement

Parameter tuning policies are useful for the performance improvement of legacy and state-of-the-art applications. However continuous monitoring is required for performance improvement. Also, in future multi-core systems are going to get the highly dynamic load, multiple applications running in a concurrent way, applications changing their I/O Characteristics and not predictable system resources requirements. Managing such type of systems would be tedious. To utilize such systems in terms of high throughput, applications should show autonomic behavior [48].

VIII.I Need of Self-awareness for Performance Improvement

The policies enlisted above for the performance improvement are dependent on the sub-system performance like main memory, cache, bandwidth, etc. The applications running in the system follow the policies of the traditional OS for performance improvement. Applications are a prime entity in the system, besides to focus on system parameters and system internal attention should be given to the applications also[22].

Applications running in the system must have to show adaptive behavior. A form of behavior is called adaptive if it maintains the essential parameters within the psychological limit. Adaptive behavior of the application means applications change in such a way in changing the environment that the system performance should be in the limit. We can say that our system is adaptive when applications running in the system makes the system stable. Autonomic computing is the oldest method for making the system auto control.

Autonomic computing uses sensor channels to sense the changes caused by internal & external factors[48]. Internal factors such as excessive CPU utilization, external factors such as load in other cores affects the internal connection network which further affects the system performance.

VIII.II Proposed Self-aware policies

Self-aware policies used for the performance improvement

of multi-core systems are enlisted in the TABLE- 8. In another situation, it is noted that to provide security in the multi-core systems self-aware based approach could be beneficial. In this approach multi-core resource could be portioned; one partition set could be used as an observer for the other portion for the security concerns[51]. The approach would well be suited for the many-core architectures, however, would be not much fruitful for multi-cores as it already suffers from the resources shortage. Self-aware based policies are also used to find the best configuration to execute the stencil codes[41]. To accelerate the process of finding the appropriate parameter value in considerable search space machine learning based approaches shown the improvement. However, machine learning based approaches need to be modified to realize the actual self-aware behavior.

To incorporate the self-aware environment the system components like main memory could be adaptive. Further, whole memory could be portioned into independent, self-aware parts & each part has its own memory allocation/deallocation mechanism[52]. Since the self-aware memory is flexible, decentralized it could address the consistency issues like synchronization. However, for implementing effectively required to explore new policies & updations in the systems. Applications run in the system are not known how they behave during run time. Taking appropriate actions on performance degradation is necessary. Self-healing based policies could search for the appropriate action; needed to take for performance improvement[40].

Table-8 Investigating Self-aware policies as per situation and criteria used

Investigating Self Aware & Machine Learning based policies						
S.No	**Reference**	**Approach Used for Performance Improvement**	**Parameters Considered**	**Tools/other instrument used**	**Benefits**	**Limitation**
1	D. Dasgupta et al. [51]	A self-aware approach. Resource partitioning is done for addressing security issues in multicores systems. Cores are partitioned: one set is for general work & another for the security of the firstone.	Partition overhead	Theoretical Model	Providing security features in existing multi-core systems with little modification on hardware & software.	Not discussing the performance while addressing security issues. Could be useful for the many-core systems; using to the multicore system would be costly to dedicate for security.
2	A. Ganapath i [41]	Machine learning Used the Machine learning on multicore architectures for parameter auto-tuning	Thread count, cycles per thread	statistical machine learning	Helpful for executing scientific applications where a lot of computation involved. Novel approach:- used machine learning for parameter autotuning.	Useful for scientific applications computing, not much use for commodity processors
3	Hariri et al[48]	Autonomic features like self-configuration, self- healing, self-optimization are proposed.	Local & global sensors	Paradigm	Autonomic features like self- optimization could tune system performance autonomically.	Overhead of Converting traditional application into Autonomic One
4	O. Mattes and W. Karl [52]	Decentralized self- optimization, Flexible memory architecture	Migration cost	SystemC based simulation	Isolation of memory, flexible memory architecture, putting memory close to the processor.	Implementation of the proposed concept could create issues like- Interfacing, synchronization, application phase change and memory requirements.
5.	E. Lau, et al. [30]	Special core for running self-aware algorithms & energy saving. Two cores. Small core focus on energy and the main core focus of the application performance	Memory latency	Graphite simulator used to execute EM3D benchmark and its helper threads.	The programmer need not have to worry about the performance & power, it is managed by the special core.	Complexity on interfacing the extra core.

IX. Policy Summary

After the review cited above, we have summarized below:
It is noted that system performance severely affected due to the application characteristics interference for the shared resources like last level cache, main memory & bandwidth. There is a gap to mitigate the application interference effect to improve the performance by interference parameter tuning. It is also observed that researchers have proposed hardware or software-based approaches to tackle the performance issues, but less attention is drawn on the middle-ware & firmware based approaches that could be beneficial for the performance improvements.

Another important fact is that the researchers used architecture independent characteristics of the applications

to tackle the performance issues seem to be very useful for a diverse set of hardware platforms and further it could avoid the application parameter tuning overheads.

In another scenario, resource management through static & dynamic approaches; like profiling & sampling respectively, are less effective until we have some dedicated online monitoring module in the system which could track resources utilization for applications individually and collectively.

It is also noted that the application and execution environment works in isolation; which means the applications do not get the system's input in runtime & system does not bother about the application inherent goals like speed for the execution, rate at which application wants to transfer the data, leading to wastage of valuable resources like energy.

For monitoring & analysis of the system performance & resource utilization, many of the approaches rely on hardware performance counters(HPC). HPC is useful but, its use requires in-depth study & manual effort for reasoning on which HPC will be the best, for which event, and thus calls for the need of runtime autonomic mechanism for mapping of HPC to application threads at runtime to reduce the mapping overhead.

It is noted that tactics like code instrumentation(inserting instructions in the application hot-spot to observe the time taken for execution); creating logical clusters of cores; & memory guard are some of the substantial steps towards the performance improvements but are effective on limited performance issues.

Hardware-based approaches such as partner core; scratch pad memory; hardware tuners to address performance issues are lacking due to their on-chip integration & heat dissipation challenges.

Heuristic-based approaches like orchester schedulers, cognitive computing based scheduling, & on-line reinforcement learning to handle resources management & energy issues are prominent and shown a significant impact to solve the performance issues. However, it requires expertise to map the performance problems to the heuristic based solutions to harness their utility.

It is also noted that modeling based software approaches for resource management are useful for high-level infrastructure like data centers. However, due to the high calculation required for the prediction of resource usability in advance, are not successful for small multi-core devices.

It is also observed that proposed solutions for the performance of the systems are related to the application characteristics. The solution for data locality issue also had found the solution in the application itself; which means get the memory access pattern of the application in the source code & create the thread to the memory object graph & finally keep track about the creation and destruction of the memory to solve the issue.

It is noted that to address the conflicting goals of applications, tuning of performance parameters & allocation of resources to the applications in the dynamic execution environment; the existing centralized policies in the multi-core system are not sufficient.

In the multi-core system, global rules used for resource allocation & addressing performance issues, like contention are not effective & fair. There is a need for simple local rules which applications should follow in a cooperative manner; in the same way as adopted by the well-established computing paradigm used in implementing swarm intelligence for autonomic systems.

In other work, it is shown that it could be advantageous to shift from conventional multi-core architectures like reconfigurable, Bahurupi, etc. to new Autonomic multi-core architectures.

It may be noted that the solution for the performance enhancement of the multi-core system proposed does stand in isolation; like cache tuning, scratch-pad memory which is a gap as solutions lack on the totality.

It is also noted that Shared resources and thread to core mapping strategies are closely related to each other, so both have to perform by considering their trade-off.

Looking to the efforts cited above, till less attention is drawn to the well-established computing paradigms like self-aware & autonomic computing for improving multi-core system performance smartly & efficiently. Which means It will be beneficial if applications participate in solving the performance issues of an existing multi-core system; through cooperation & adapt itself in run time for resources shortage.

In the context to enhance the performance and utilization of multi-core (CPU) systems concerning throughput & reduced application's execution time, applications running in the system should have to show a self-aware & adaptive behavior.

It is also found that hardware performance counters have emerged as a vital instrument to capture the system parameters. Most of the policies are getting the system insights like cache latency, bandwidth utilization and cache miss rate through performance counters.

It is investigated that, Last level cache (LLC) has prime importance on system performance. Since applications execution delays due to high pressure on LLC to access the shared data.

CONCLUSION

It is worth mentioning that enormous work has been done for decades and diverse set of policies were proposed for performance improvement. However, selecting an appropriate policy for a distinct workload requires in-depth analysis of application characteristics and the policies effectiveness. We had done a detailed study of approaches employed for the multi-core system performance improvement. For contemporary applications and state-of-the-art multi-core architecture, attempts for performance

improvement through traditional approaches which focus on improvements of memory policies seems ineffective. Besides, complexity and power dissipation issues of the traditional single processors, it is not possible to overlook it completely. Another fact observed is that cache contention & data locality are two important factors which degrade the performance by 20% due to the parameter tradeoff. Data locality causes severe performance problems for the shared cache. It is found that the policies are interrelated to each other. One policy which commands on handling an issue may help in another way to tackle the other performance issues indirectly; for example, mapping a thread to a processor core which is far away from its local memory could be a policy of thread to core mapping, may solve the issue of contention of the memory controller. Scheduling & memory allocation activities should be done by looking at the aspects of overlapping and trade-offs of each other. In the classification, we had found that application to core mapping issues degrades the system performance by approximately 30% and the appropriate scheduling algorithm increases the speedup up to 20 to 40%.

It is also observed that all the policies discussed in the survey had shown improvement in system performance in varied situations. Also, these policies are beneficial for a particular application type and system configuration. However, to address the system performance issues, in totality which policy is best suited, what should be the order of policies if one has more than one options, how a policy could dominate the other policy during parameter tuning, how a policy is effective compared to other policies in a particular workload type is not considered in totality, could be considered for future work. In the future, we may perform a comparative analysis of how the above-mentioned policies if dynamically assigned in the system are fruitful.

Finally, in the future, we have to deal with very complex situations to improve multi-core system performance. We have a diverse set of applications, a list of performance issues due to heterogeneity on system resources and have the number of policies to address the performance issues; and all have a common goal- Performance. This situation could be handled through self-aware, autonomic, intelligent & dynamic software or hardware based interface exist between application and OS, which could consider complex scenarios and address them dynamically. In future two contributions could be helpful to improve the multi-core system performance. First, a software-based "multi-criteria aware holistic policy scheduler" which could pick an application and assign the best policy from a set of policies by considering the system performance dynamically. Second, a hardware or software based interface between applications and OS which could incorporate self-aware and autonomic properties in legacy as well as state-of-the-art applications also in system components.

REFERENCES

[1] D. Geer, "Chip makers turn to multi-core processors", Computer, vol. **38**, no. **5**, pp. **11-13**,**2005**.

[2] A. Roy, J. Xu, and M. Chowdhury, "Multi-core processors: A new wayforward and challenges", International Conference on Microelectronics, Sharjaha,**UAE,** pp.**454-457**, **2008.**

[3] G. Blake, R. Dreslinski and T. Mudge, "A survey of multi-core processors", IEEE Signal Processing Magazine, **vol.** 26, **no. 6**, pp. **26-37**,**2009**.

[4] S. Hao, Q. Liu, L. Zhang, and J. Wang, "Processes Scheduling on Heterogeneous Multi-core Architecture with Hardware Support", in International Conference on Networking, Architecture, and Storage, **China**, , pp. **236-241, 2011.**

[5] R. Teodorescu and J. Torrellas, "Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors",66 in International Symposium on Computer Architecture, Beijing, **China**, pp.**363-374, 2008.**

[6] Y. Cheng, W. Chen, Z. Wang, and Y. Xiang, "Precise contention-aware performance prediction on virtualized multicore system," Journal of system architecture, vol. **72**, pp. **42-50**, **2017**.

[7] A. Asaduzzaman "Performance modeling of multicore and manycore networked systems," in International Journal of Computer Networks and communications (IJCNC), vol.**4**, No.**2**, pp. **53-67, 2012.**

[8] S. Prasad, "Program Execution on Reconfigurable Multicore Architectures", Electronic Proceedings in Theoretical Computer Science, vol. **211**, pp. **83-91, 2016**.

[9] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo and L. Sha, "Memory Bandwidth Management for Efficient Performance Isolation in Multi-Core Platforms", IEEE Transactions on Computers, vol. **65**, no. **2**, pp. **562-576,2016**.

[10] S. Ren, L. Tan, C. Li, Z. Xiao and W. Song, "Leveraging Hardware-Assisted Virtualization for Deterministic Replay on Commodity Multi-Core Processors", IEEE Transactions on Computers, vol. **67**, no. **1**, pp. **45-58,2018.**

[11] M. Pricopi and T. Mitra, "Bahurupi", ACM Transactions on Architecture and Code Optimization, vol. **8**, no. **4**, pp. **1-21, 2012.**

[12] James E. Bennett, Michael J. Flynn, Performance Factors for Superscalar Processors, Stanford University, Stanford, CA,**1995**

[13] Lachaize, R., Lepers, B. and Quéma, V., "MemProf: a memory profiler for NUMA multicore systems", In: USENIX ATC'12 Proceedings of the 2012 USENIX conference on Annual Technical Conference. **Boston**: ACM, pp.**5-5, 2012.**

[14] R. Knauerhase, P. Brett, B. Hohlt, T. Li, and S. Hahn, "Using OS Observations to Improve Performance in Multicore Systems", IEEE Micro, vol. **28**, no. **3**, pp. **54-66, 2008**.

[15] D. Shelepov and et al., "HASS: A Scheduler for Heterogeneous Multicore Systems", in ACM SIGOPS Operating Systems Review, **New York**, pp. **66-75, 2009**

[16] Becchi M, Crowley P, "Dynamic thread assignment on heterogeneous multiprocessor architectures", In Proceedings of the 3rd conference on computing frontiers, New York, **2006**, pp. **29-40, 2006**

[17] Kumar R et al. "Single-ISA heterogeneous multi-core architectures for multithreaded workload performance", In Proceedings of the 31st annual international symposium on computer architecture, **Washington**, pp., **64-75, 2004**

[18] T.M.Birhanu, Z. Li, H. Sekiya, N. Komuro, Y.-J. Choi, "Efficient thread mapping for heterogeneous multicore iot systems", *Mobile Information Systems*, vol. **1565**, pp. **8**, **2017**

[19] D. Koufaty, D. Reddy, and S. Hahn, "Bias scheduling in heterogeneous multi-core architectures," in Proc. of the 5th European Conference on Computer Systems, **France** pp. **125-138, 2010**

[20] S. Zhuravlev, S. Blagodarov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," in ACM SIGARCH Computer Architecture News, vol. **38**, pp. **129–142**, **2010.**

[21] Z. Majo and T. Gross, "Memory management in NUMA multicore systems: trapped between cache contention and interconnect overhead", in ACM SIGPLAN Notices - ISMM '11, pp.**11-20 2011**.

[22] Agarwal A, Miller J, Eastep J, Wentziaff D, Kasture H Self-aware computing. Technical report, MIT, **2009.**

[23] D. Molka, R. Schöne, D. Hackenberg and W. Nagel, "Detecting memory-boundedness with hardware performance counters", Proceedings of the 8th ACM/SPEC International Conference on Performance Engineering, **Italy**, pp. **27-38, 2017.**

[24] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, L. Sha, "MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms", *Proc. Real-Time Embedded Technol. Appl. Symp.*, **USA**, pp. **55-64, 2013.**

[25] Karcher, T., Pankratius, V.: Auto-Tuning Multicore Applications at Run-Time with a Cooperative Tuner. Technical Report, 2011-4, Karlsruhe Institute of Technology, Germany (2011)

[26] P. Kansakar and A. Munir, "A Design space exploration methodology for parameter optimization in multicore processors," IEEE Trans. Parallel Distrib. Syst., vol. **29**, no. **1**, pp. **2–15,2018**.

[27] M. Kulkarni, V. Pai, and D. Schuff, "Towards architecture independent metrics for multicore performance analysis," ACM SIGMETRICS Perform. Eval. Rev., vol. **38,** pp. **10-14**, **2011.**

[28] Y. Wang and K. B. Kent, "A Region-Based Approach to Pipeline Parallelism in Java Programs on Multicores," Proc. - 2017 25th Euromicro Int. Conf. Parallel, Distrib. Network-Based Process. PDP 2017, **Russia**, pp. **124–131, 2017.**

[29] R. Das et al., "Application-to-core Mapping Policies to Reduce Memory System Interference in Multi-core Systems", *HPCA 2013*.

[30] Eric Lau , Jason E. Miller , Inseok Choi , Donald Yeung , Saman Amarasinghe,Anant Agarwal, "Multicore performance optimization using partner cores", Proceedings of the 3rd USENIX conference on Hot topic in parallelism, **Berkeley,** pp.**11-11, 2011**,

[31] Da-WeiChang,Ing-ChaoLin,Yu-ShiangChien,Chn-LunLin,A.Su,andChung PingYoung,"CASA:Contention-Aware Scratchpad Memory Allocation for Online Hybrid On-Chip Memory Management", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. **33**, no. **12**, pp. **1806-1817, 2014.**

[32] S. Gu, Q. Zhuge, J. Yi, J. Hu, E. H.-M. Sha, "Optimizing task and data assignment on multi-core systems with multi-port SPMs", *IEEE Trans. Parallel Distrib. Syst.*, vol. **26**, no. **9**, pp. **2549-2560, 2015.**

[33] N. Ramasubramanian, V. V. Srnivas, and N. Ammasai Gounden, "Performance of Cache Memory Subsystems for Multicore Architectures," Int. J. Comput. Sci. Eng. Appl., vol. **1**, no. **5,** pp. **59–71,2011**.

[34] V. Kazempour, A. Fedorova, and P. Alagheband, "Performance implications of cache affinity on multicore processors," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. **5168**, pp. **151–161, 2008.**

[35] M. Rawlins, A. Gordon-Ross, "A cache tuning heuristic for multi-core architecture", *IEEE transaction on computers*, vol. **62**, no. **8**, pp. **1570-1583, 2013.**

[36] M. Rawlins and A. Gordon-Ross, "An application classification guided cache tuning heuristic for multi-core architectures", 17th Asia and South Pacific Design Automation Conference,2012.

[37] K. Huang, K. Wang, D. Zheng, X. Zhang and X. Yan, "Access Adaptive and Thread-Aware Cache Partitioning in Multicore Systems", Electronics, vol. **7**, no. **9**, pp. **172, 2018.**

[38] Y. Song, O. Alavoine, and B. Lin, "Row-buffer hit harvesting in orchestrated last-level cache and DRAM scheduling for heterogeneous multicore systems," in Proceedings of the 2018 Design, Automation and Test in Europe Conference and Exhibition, **Janua**, pp. **779–784, 2018**

[39] P. Anuradha, H. Rallapalli, and G. Narsimha, "Energy efficient scheduling algorithm for the multicore heterogeneous embedded architectures," Des. Autom. Embed. Syst., vol. **22**, no. **1–2**, **2018**

[40] Fuad, M., Deb, D. and Baek, J., "Self-Healing by Means of Runtime Execution Profiling", Proceedings of 14th International Conference on Computer and Information Technology, **Dhaka,** pp., **202-207, 2011.**

[41] A. Ganapathi, K. Datta, A. Fox, and D. A. Patterson, "A case for machine learning to optimize multicore performance," in Proceedings of the First USENIX conference on Hot topics in parallelism, 2009.

[42] Jain, R., Panda, P. and Subramoney, S. "Cooperative Multi-Agent Reinforcement Learning-Based Co-optimization of Cores, Caches, and On- chip Network", ACM Transactions on Architecture and Code Optimization, vol. **14**, issue-**4**, pp.**1-25, 2017.**

[43] N. Huber, F. Brosig, S. Spinner, S. Kounev and M. Bahr, "Model-Based Self- Aware Performance and Resource Management Using the Descartes Modeling Language", IEEE Transactions on Software Engineering, vol. **43**, no. **5**, pp. **432-452, 2017.**

[44] K.Hasan,J.Antonio, and S.Radhakrishnan," A model-driven approach for predicting and analysing the execution efficiency of multi-core processing", International Journal of Computational Science and Engineering, vol. **14**, no. **2**, pp. **105-125, 2017**.

[45] Khondker S. Hasan, John K. Antonio, and Sridhar Radhakrishnan, "A New Multi-core CPU Resource Availability Prediction Model for Concurrent Processes," Lecture Notes in Engineering and Computer Science: Proceedings of The International MultiConference of Engineers and Computer Scientists, **Hong Kong**, pp. **130-135, 2017.**

[46] K. Moazzemi, A. Kanduri, D. Juh´asz, A. Miele, A. M. Rahmani, P. Liljeberg, A. Jantsch, N. Dutt, "Trends in on-chip dynamic resource management", in21st Euromicro Conference on Digital System Design (DSD), **Prague**, pp. **62–69, 2018.**

[47] Yan-fei Zhu and Xiong-min Tang, "Overview of swarm intelligence",International Conference on Computer Application and System Modeling, **China,** pp. **400-403, 2010**.

[48] Hariri, B. Khargharia, H. Chen, J. Yang, Y. Zhang, M. Parashar, and H. Liu, "The Autonomic Computing Paradigm", Cluster Computing, vol. **9**, no. **1**, pp. **5-17, 2006.**

[49] Lewis, P.R., Self-aware computing systems: from psychology to engineering. In: Design, Automation and Test in Europe Conference and Exhibition, **Switzerland**,pp.,**1044–1049, 2017.**

[50] N. Dutt, A. Jantsch, and S. Sarma, "Toward Smart Embedded Systems", ACM Transactions on Embedded Computing Systems, vol. **15**, no. **2**, pp. **1-27, 2016**.

[51] D. Dasgupta, H. Bedi, and D. Garrett, "A conceptual model of self-monitoring multi-core systems," Proc. Sixth Annu. Work. Cyber Secur. Inf. Intell. Res. - CSIIRW '10, pp. 83 1,2010.

[52] O. Mattes and W. Karl, "Evaluating the Self-Optimization Process of the Adaptive Memory Management Architecture Self-aware Memory," in Proceedings of the 1st Workshop on Resource Awareness and Adaptivity in Multi-Core, **Germany** pp.**16–21, 2014.**

[53] G.P. Sunitha, B.P. Vijay Kumar, S.M. Dilip Kumar, "A Nature Inspired Optimal Path Finding Algorithm to Mitigate Congestion in WSNs", International Journal of Scientific Research in Network Security and Communication, Vol.**6**, Issue.**3**, pp.**50-57, 2018**

[54] Chingrace Guite, Kamaljeet Kaur Mangat, "*A Study on Energy Efficient VM Allocation in Green Cloud Computing*", International Journal of Scientific Research in Computer Science and Engineering, Vol.**6**, Issue.**4**, pp.**37-40, 2018**

Authors Profile

Mr. Surendra K Shukla Pursuing Ph.D. in Computer Science & Engineering from School of Computer Science & IT, DAVV University, Indore, M.P., India. He has received his B.E degree in Computer Engineering from, SGSITS College Indore, M.P., India in 2004 and M.E in Computer Engineering from IET, DAVV University, Indore, M.P. India in 2011. His research area is multicore architectures, Parallel Computing.

Dr. P.K. Chande is working as a Director, at SVKMs, NMIMS, Deemed to University, 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 have 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.

