COMFAST: A Comparative Framework for Analysis of Scheduling Techniques in Multi-core Systems

Sarah Shah
School of Electrical Engineering and Computer Science
National University of Sciences and Technology
Islamabad, Pakistan
13beesshah@seecs.nust.edu.pk

Abdul Qahir
School of Electrical Engineering and Computer Science
National University of Sciences and Technology
Islamabad, Pakistan
13beeaqahir@seecs.nust.edu.pk

Masooma Safeer
School of Electrical Engineering and Computer Science
National University of Sciences and Technology
Islamabad, Pakistan
13beemsafeer@seecs.nust.edu.pk

Sana Mazahir
School of Electrical Engineering and Computer Science
National University of Sciences and Technology
Islamabad, Pakistan
sana.mazahir@seecs.nust.edu.pk

Osman Hasan
School of Electrical Engineering and Computer Science
National University of Sciences and Technology
Islamabad, Pakistan
osman.hasan@seecs.nust.edu.pk

***Abstract*—Multi-core systems are increasingly being used for developing many real-time systems to meet the growing demands of high-speed computing and low-power dissipation. However, the delays due to the conflicts that arise when multiple cores access the shared system resources simultaneously encumber these systems to reach their peak efficiency. Several scheduling algorithms have been proposed to abate the issue of resource contention. However, comparing various scheduling techniques is a major challenge due to the different underlying assumptions and the lack of a common ground for analysis. In this paper, we propose a unified Comparative Framework for Analysis of Scheduling Techniques (COMFAST) in multi-core systems. The proposed framework supports the comparison based on a general data set and characterizes the performance in terms of the worst-case response times (WCRT) and acceptance ratios. For illustration, the paper presents the comparison of two of the recently proposed scheduling techniques, i.e., 1) Dynamic Task Aware Scheduling that uses complimentary tasks to mitigate contention, and 2) First Fit Allocation Based on Symmetric Analysis that approaches response time through the shared-resource-centric perspective and the core-centric-perspective. Our results show that Dynamic Task Aware Scheduling provides considerably reduced response times for a variety of resource access frequencies compared to First Fit Allocation. However, in terms of acceptance ratios, First Fit Allocation demonstrates a much better performance.**

***Index Terms*—Multi-core, Shared Resources, Resource Contention, Scheduling Algorithm, Worst Case Response Time (WCRT), Acceptance Ratio**

I. Introduction

With the swift escalation in the demand for faster and energy-efficient computing, multi-core processors [1] have emerged as a promising solution to provide a scalable performance enhancement by parallel execution of multiple tasks. These multi-core processors are increasingly being embedded into real-time systems, thus making their way into many technological fields, such as, Robotics, Supercomputers, Graphic Processing Units (GPU), Automated Industrial Processes, Telecommunication and intricate Space, Automotive and Avionic applications [2], [3].

The multiple cores have to share certain resources of the processor, such as the memory bank, system bus, I/O and low level cache etc. When multiple cores require the same resource at the same time, they face a conflict, termed as *Resource Contention* [4]. If the resource arbiter allows access to the core running the task with the highest priority among the contending tasks, the other cores will have to wait till the task, which has been granted access to the shared resource, completes its execution on the shared resource. This results in the addition of surplus time in the response period of the tasks, which upon accumulation leads to a delayed completion of processes and inefficient use of the processor's resources. These outcomes in turn prevent the processor from achieving peak efficiency. Therefore, minimizing resource contention in multi-core systems has gained significant attention recently. [5]–[7].

In real time systems, it is very important that all tasks meet their deadlines. A set of tasks is said to be schedulable if none of the tasks exceed their deadlines. A scheduling algorithm is used to allocate tasks to cores, ensuring that the time constraints of all tasks are met. The performance

of a scheduling algorithm is measured by a schedulability test [6], which gives the amount of task sets that were deemed schedulable using that scheduling algorithm. In order to perform a schedulability analysis, the Worst Case Response Time (WCRT) [8]–[10] must be known. The WCRT of a task is the sum of its execution time and the delays that are accumulated due to preemptions from higher priority tasks on the same or different cores.

Most of the existing scheduling techniques have very different infrastructures. Moreover, contention is often evaluated for certain specific system resources, usually memory and low level cache [9], [10]. Different scheduling techniques optimize different metrics for overcoming contention. For instance, Dynamic Task Aware Scheduling [5] and First Fit Allocation using Symmetric Response Time Analysis [6], have been tested in different environments and with different variables to analyze the efficiency of the underlying algorithms. The former uses the cache miss rate to calculate the resource requirement of tasks on runtime and focuses on minimizing contention by curtailing the scenarios in which the tasks with similar resource demands may have a contention. The latter however, is driven by the schedulability analysis of the task sets and generates an estimate of the resource requirement of a task by varying utilizations. Thus, based on the existing literature, it is very difficult to compare the two scheduling algorithms and identify the best for a given scenario.

In order to overcome the above-mentioned limitations, we propose a COMparative Framework for Analysis of Scheduling Techniques (COMFAST) in multi-core systems. COMFAST generates a general data set with randomly generated resource access rates for all scheduling techniques under study. It takes into account the total number of resource accesses instead of the different types of shared resources that a task accesses in order to measure performance for the overall resource demand. For the sake of having a fair comparison, all the given algorithms are assessed under the same parameters. This way, the evaluation of the given scheduling schemes and algorithms is solely based on the efficiency of their task allocation schemes. This efficiency is primarily measured in terms of response times and schedulability of given task sets.

Additionally, we use COMFAST for the comparative analysis of two scheduling algorithms: *Dynamic Task Aware Scheduling Algorithm* [5] and *First Fit Allocation Based on Symmetrically Calculated Response Times* [6]. The two techniques were selected because they were proposed recently and adopt different methods to address delays specifically due to resource contention. COMFAST was successful in providing a thorough analysis of the two algorithms; exposing their pros and cons in different contention scenarios. Our investigation reveals that using complementary tasks to tackle contention gives considerably lower response times than using First Fit Allocation. On the other hand, the performance of complimentary tasks on acceptance ratios is found to be lower than that of First Fit Allocation.

II. Related Work

McGregor et al. [11] presented a scheduling algorithm that incorporates cache miss rate, bus transaction and stall cycle rate in the scheduling process of a task. Within a package, each processor chooses *complementary tasks* with different characteristics and resource requirements simultaneously. This results in less competition for shared resources in the package, thus, accomplishing better execution of a task set.

Failure to equalize loads is also detrimental to system's performance. Hence, a load balancing mechanism was proposed by Merkel et al. [7] that migrates appropriate tasks between processors to maintain balanced resource utilization. But task relocation causes an overhead in the execution time for each task and continuous migration over the cores can cause a non-negligible escalation in the overall response time.

Chiang et al. [5] modified the code of the Linux kernels 2.6.33 and 3.11.0 to implement *Dynamic Task Aware Scheduling (DTAS)* in Non-Uniform Memory Access (NUMA) multi-core systems. DTAS divides the cores and incoming tasks into memory-bound and compute-bound categories and the tasks are assigned to their respective cores based on their type. Moreover, the scheduler implements a load balancing mechanism by migrating tasks from overloaded cores to idle cores. Since the decision about a task's resource demand is made by using its number of cache misses, the algorithm considers memory as the only shared resource, disregarding other resource requirements of a task.

Huang et al. [6] proposed a task partitioning algorithm based on *First Fit Allocation* and a *symmetric approach* to evaluate the response time of tasks from core-centric and shared resource-centric perspectives. The response times are then used to check the schedulability of a task set. The suggested mechanism gives a considerable computational speedup and a high acceptance ratio but only for tasks with a small number of resource accesses.

Choi et al. [12] proposed a novel approach for evaluating contention based on *task dependency*. The tasks on each core undergo a *separation analysis* to determine which tasks overlap in time and can run independently in parallel, and which are dependent and require sequential execution. A target task is chosen and a set of tasks that can be candidates for causing resource contention for the target task is made on each core and then put into clusters. These clusters are used to evaluate the *PE-Level Shared Resource Demand Bound Functions* that are then used to find out the maximum resource demand generated by each core for the target task.

All the above-mentioned works use ad hoc methods for analysis purposes and thus fail to provide a unified framework for the comparative analysis of different scheduling techniques because of the incompatible approaches on which each technique is built upon.

Khera et al. [13] presented a study on the comparison of different scheduling techniques that are used in real-time environments. The study does not provide a platform for the analysis of the discussed scheduling techniques, rather gives

TABLE I
PARAMETERS OF THE PROPOSED SYSTEM MODEL

Symbol	Definition
G	Task graph
τ_k	Target task
τ_i	Current task
D_i	Task deadline
τ^{BCET}	Best case execution time
τ^{WCET}	Worst case execution time
R_i	Response time
τ^{minS}	Minimum start time
τ^{maxS}	Maximum start time
τ^{minF}	Minimum finish time
τ^{maxF}	Maximum finish time
τ^{N}	Maximum number of resource accesses
τ^{M}	Maximum access duration
τ^{d}	Total resource demand
τ^{pri}	Task priority
τ^{p}	Processor
τ^{CON}	Amount of contention
G^S	Schedulability of a task graph
σ	Acceptance ratio

a general overview of different eminent scheduling algorithms with respect to their processor utilization, priority schemes and their classification. The paper concludes by stating that the dynamic priority scheduling algorithms increase utilization and produce an overhead, which is why they are not employed in commercial real-time systems. It also concludes that EDF is the most suitable algorithm when computational utilization is less than 1.

Peloquin et al. [14] published a thorough analysis of three scheduling algorithms: Global EDF [15], LLREF [16], and PF [17]. The algorithms were analyzed based on their schedulability and number of invocations and migrations in different randomly generated task sets. The examination was performed on an open-source framework called RTSIM. However, the authors had to make several changes to some of the modules to implement the algorithms. Moreover, they had to face several memory leakage issues when the simulations were running for longer durations.

III. COMPARISON METHODOLOGY

A. System Model

We assume a system comprising of *m* cores and a data set containing several task graphs. A task graph G_i is a set of *n* dependent and independent tasks $\{\tau_1, \tau_2, \tau_3, ..., \tau_n\}$. Each task τ is modeled by its deadline *D*, time period *T*, worst case response time *R*, priority τ^{pri}, total resource demand τ^d, the processor it is mapped to τ^p and the best and worst case execution times, τ^{BCET} and τ^{WCET}, respectively. τ_i^{WCET} of a task τ_i is the summation of τ_i^{BCET} and the time duration for which the task is blocked by other tasks on the same core.

A task τ_i is labelled as dependent or independent with respect to a target task τ_k based on maximum and minimum starting and finish times of the two tasks $\{\tau^{maxS}, \tau^{maxF}, \tau^{minS}, \tau^{minF}\}$. Each task graph undergoes a separation analysis [12], where these four variables provide information regarding an overlap between the running times of the two tasks. The tasks that do not intersect in time are called *dependent tasks* as they follow a fixed precedence in time. The tasks that overlap with each other in time are labelled as *independent tasks* and are possible contenders for a shared resource.

The total resource demand τ^d of a task depends on two parameters: maximum access duration τ^M and maximum number of resource accesses τ^N, as demonstrated by Eq. (1). For simple calculations, we assume that all resource accesses of a task are of length τ^M. Thus,

$$\tau^d = \tau^N * \tau^M \tag{1}$$

The response time *R* of a task τ is calculated in terms of the worst case execution time τ^{WCET} and the surplus time τ^{CON} added due to resource contention as given in Eq. (2). For a task τ_k, τ_k^{CON} is calculated in terms of the total demand τ^d of all the tasks that may run in parallel with τ_k i.e., all tasks that are independent of τ_k and have a higher priority than τ_k. If none of the independent tasks interfere with τ_k, τ_k^{CON} would be zero and R_k would be equal to τ_k^{WCET}. We assume fixed-priority preemptive scheduling for both algorithms, hence a task can be blocked by any of the higher priority tasks in that task graph.

$$R_k = \tau_k^{CON} + \tau_k^{WCET} \tag{2}$$

All task graphs are deadline monotonic i.e., for all task graphs, the task with the smallest relative deadline is assigned the highest priority and the task with the largest deadline is given the lowest priority. This implies that all tasks that have a shorter deadline relative to τ_i have a higher priority than τ_i and can preempt it at any time.

A task graph may have an implicit-deadline $(D_i = T_i)$ or a constrained-deadline $(Di \leq Ti)$. However, for simplicity, all task graphs in our data set are assumed to be of implicit-deadline.

Each task graph is represented by its schedulability G^S. A task graph is said to be schedulable if all its constituent tasks meet their deadlines $(G^S = 1)$ and is considered non-schedulable if any one of its tasks is unable to meet its deadline $(G^S = 0)$. Each data set is characterized by a quantity σ, called the *acceptance ratio*, which is the ratio of number of schedulable task graphs to the total number of task graphs in that data set. Table I summarizes all the variables used in our model.

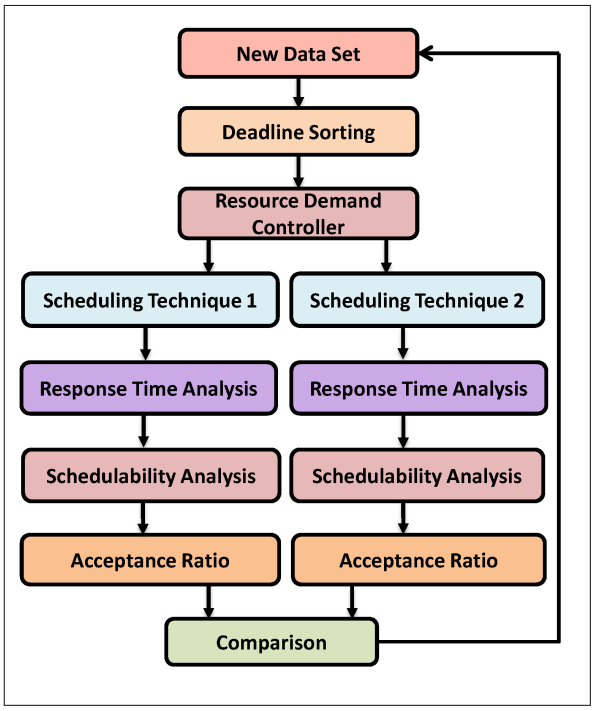

Fig. 1. COMFAST Methodology

B. COMFAST: Comparative Framework for Analyzing Scheduling Techniques

COMFAST provides a uniform platform for the analysis of different task allocation techniques. For this purpose, COMFAST establishes the following principles:

1) The tasks are sorted for each scheduling technique in a way to facilitate the priority scheme used by the scheduling technique under study. For instance, since First Fit Allocation uses a deadline monotonic scheme, COMFAST sorts the tasks in ascending order of their deadlines before dispatching them to the algorithm.
2) The number of resource accesses and amount of access durations are assumed to be known at the time of the analysis and generated randomly within a given range. The system assumes the type of shared resources to be anonymous. Since we need to compare efficiency of the scheduling techniques, given a contention scenario, hence we only require the number of resource accesses and their duration, rather than the type of resources.
3) The execution times are assumed to follow a Gaussian distribution to minimize excessively random execution times.
4) Task parameters are generated randomly, which results in a data with a large variety of tasks, thus enabling a more reliable analysis.

This data set is generic enough to exhibit the key features of a wide range of real-world applications. In simple words, the behavior of any particular application can be extracted from this data set. Similarly, the modeled tasks represent all the characteristics of a real-time task. Each task has a best and worst-case execution time, with explicitly assigned starting and finish times, fixed deadline and priority. A set of tasks, called a task graph, represents a Real Time Application (RTA), with some tasks following a sequential order and some running in parallel.

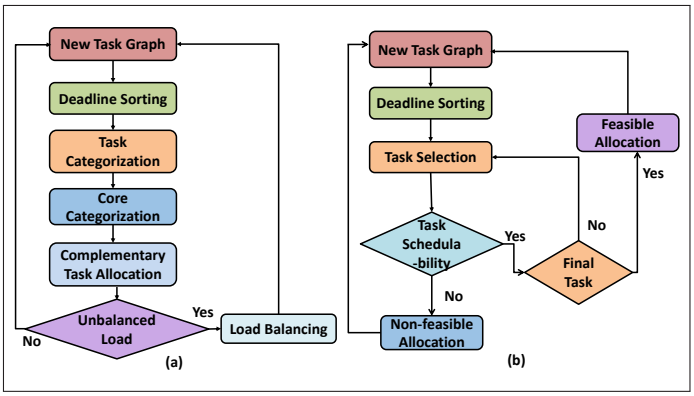

Fig. 2. a) Dynamic Task Aware Scheduling b) First Fit Scheduling

The data set produced using the above-mentioned model is then fed to two or more scheduling algorithms simultaneously and goes through the process described in Fig. 1. The algorithms are tested on the response time (R_k) of the lowest priority task, τ_k, of each task graph and the acceptance ratio of the data set. COMFAST generates simultaneous graphs for both algorithms against these metrics for high, medium and low frequency of shared resource accesses.

IV. Case Studies

The data set modelled in the above section is passed through two algorithms: a) Dynamic Task Aware Scheduling (DTAS) and b) First Fit Allocation (FF) by Symmetric WCRT. For each data set, the scheduling techniques under study are evaluated based on the acceptance ratio and average worst case response time. A target task τ_k is chosen as a representative from each task graph to derive an estimate of the worst case response time for that task graph.

A. Dynamic Task Aware Scheduling

Dynamic Task Aware Scheduling (DTAS) provides the solution to contention problem using two main approaches: conflict minimization and load balancing. Each task is categorized as memory-bound or compute-bound based on its resource demand. DTAS uses a variable called *memory-intensity* to calculate the number of cache misses per 1000 instructions. If the memory-intensity of a task is more than a set threshold, the task is labelled as memory-bound and if not, then it is labelled as compute-bound.

Similarly, the cores are also categorized as memory-bound and compute-bound. The memory-intensive tasks are allocated to memory-bound cores and the compute-intensive tasks are assigned to compute-bound cores. This allocation minimizes contention by executing tasks of different resource requirements in parallel. If the type of a task changes during runtime

due to variation in its memory intensity, then the task is migrated to another core of its respective type.

Initially, the cores are divided equally. However, if the ratio of compute-bound and memory-bound tasks exceeds a threshold, some of the cores are made to change their type to balance the load on all cores. Whenever one type of tasks exceeds the other type beyond a limit, the least busy core of the other type is changed and some of the tasks from the overloaded cores are shifted to the newly converted core. Despite adding an overhead due to task migration, load balancing helps in the efficient utilization of system resources and contributes towards reduction in resource contention.

We have simulated this mechanism in MATLAB by taking a task graph as input to the algorithm and running the task graph for a specific number of iterations, with the resource access frequencies of the constituent tasks being changed at every iteration. The memory intensity of each task and the ratio of the two types of tasks is calculated once every iteration and the tasks with the updated task types are relocated. Since the source of the algorithm does not mention the priority scheme used, we use a deadline monotonic system for DTAS to provide similar grounds for its comparison with First Fit Allocation by Symmetric Analysis. For a given task τ_i, the response time R_i is compared with its deadline, D_i. If $R_i > D_i$, for any task of the task graph, the entire task graph is deemed non-schedulable.

B. First Fit Allocation by Symmetric WCRT

This technique performs the WCRT analysis of tasks using MIRROR, which is a Worst-Case Response Time evaluation method that approaches response time symmetrically through the core-centric and shared resource-centric perspectives. The core-centric perspective states that a task executes on a core until it requires a shared resource, upon which it suspends its execution and queues in for the required resource. On the contrary, the shared resource-centric perspective is that a task continues to execute on a shared resource until it interrupts its execution on that shared resource to use the core. A task is said to be working on the shared resource if it is either queued for or is using the shared resource.

The symmetric analysis gives the times spent by a task on the core and on the shared resource. The sum of these values gives the response time R_i of the task, as a task can only be at one of these two locations during its runtime. Using this timing analysis, the algorithm uses First Fit Allocation to partition tasks on the cores. A task is dispatched to the first core that allows its scheduling without exceeding the deadline. If none of the cores allow that, the task graph is labelled non-schedulable.

For a task τ_k, higher priority tasks on the same core will interfere in the task's execution on the core, therefore, these tasks are used to calculate τ_k^{WCET}. Similarly, the higher priority tasks that are on other cores will contribute towards τ_k^{CON}, which when added to τ_k^{WCET} gives us R_k.

The priority scheme is deadline monotonic, i.e., for all task graphs, the task with the smallest relative deadline is assigned the highest priority and the task with the largest deadline is assigned the lowest priority. This implies that all tasks that have a deadline shorter than that of τ_k have a higher priority than τ_k and can preempt it at any time.

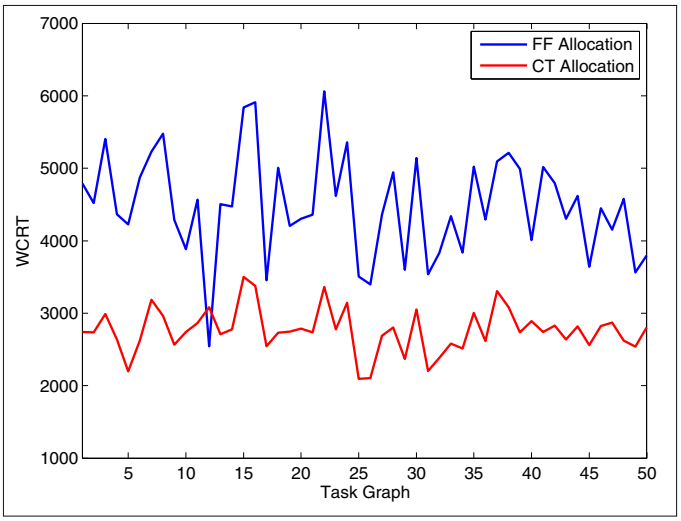

Fig. 3. Response Times for Low Resource Frequency

In our implementation of the algorithm, we only consider those higher priority tasks for calculating contention that are not completely separated i.e., the tasks that can be executed in parallel to τ_k.

V. Experimental Results

We modeled our system on MATLAB. A task graph is an array of objects of the class *TaskObj*, which is used to define various parameters of the tasks.

Our experimental setup comprises of 6 cores and general shared resource. Every core is an object of the class *Core* and is characterized by its type, ID and list of tasks mapped on it. Type is set to 0 if it is compute bound and set to 1 if it is memory bound. Every data set contains 50 task graphs, each containing 20 to 30 tasks. The number of resource accesses is generated randomly within a range of 10 to 40. The best case execution times follow a Gaussian distribution with a mean of 1250 and a standard deviation of 83.

Since the First Fit algorithm is based on a deadline monotonic system and we have chosen the same priority scheme for implementing DTAS as well in order to have an even ground for comparison, all task graphs are sorted in the ascending order of their deadlines. Thus, when a task τ_i is running, all tasks prior to it, $\tau_{j<i}$, may interfere with it on the shared resource (contributing to τ^{CON}) or on the core (contributing to τ^{WCET}).

Since our platform considers the resource accesses of a task to all types of shared resources, hence the memory intensity (used in DTAS) is quantified by the number of shared resource accesses and is thus relabeled as *resource intensity*. The threshold for resource intensity is set to 25; all tasks with a resource intensity below 25 are regarded as compute intensive

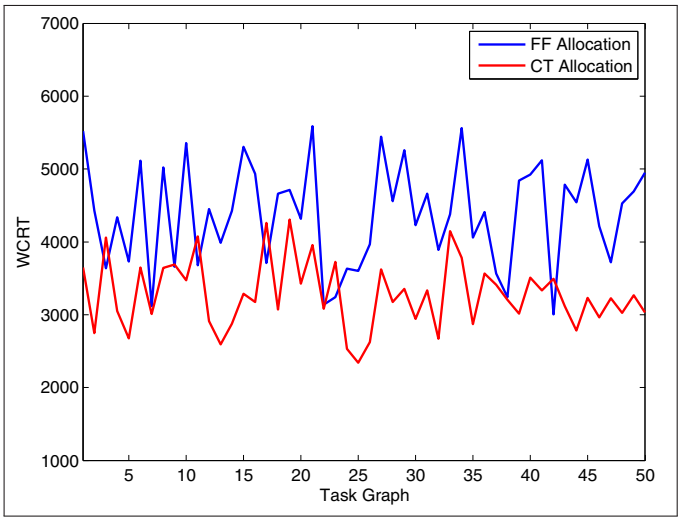

Fig. 4. Response Times for Medium Resource Frequency

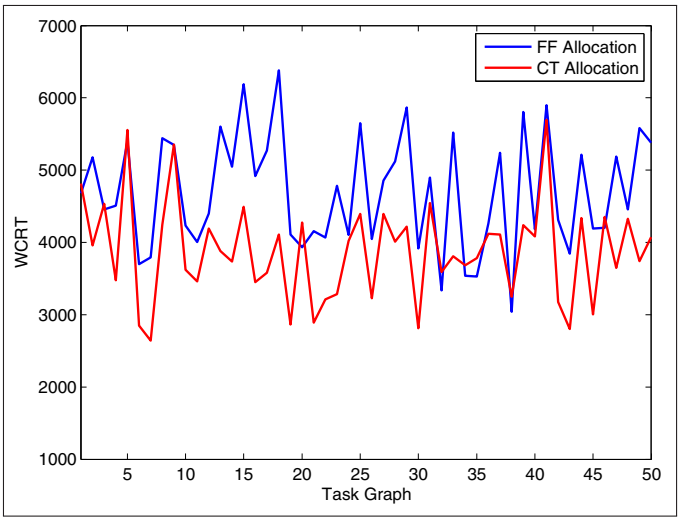

Fig. 5. Response Times for High Resource Frequency

and all tasks with a resource intensity above 25 are classified as resource intensive.

While testing DTAS, the cores are initially divided equally into the resource-bound and compute-bound categories. However, if the quantity of one type of tasks exceeds a limit, the least busy core of the other kind is made to change its type. The last task from each of the overloaded cores is migrated to the newly converted core. For each migration, an overhead of 60 is added to the response time of the task.

The data set is varied over three resource access frequencies: low, medium and high. Tasks with low resource access frequency have $10 < \tau^N < 20$, tasks with medium resource demands have $20 < \tau^N < 30$ and exceedingly resource intensive tasks have $30 < \tau^N < 40$.

Figs. 3, 4 and 5 show the response times resulting from a data set with low, medium and high resource accesses when it is subjected to the two scheduling techniques. The response times are on the y-axis and the x-axis shows the task graphs in a dataset. The task graphs are labelled from 1 to 50 and it is important to note that a task graph's label does not imply anything about the contents of the i-th task graph. The contents of a task graph are only dependent on the resource access frequency range that we choose before running a simulation; the rest of the variables are chosen from a fixed range for each task. For a deadline monotonic data set, Dynamic Task Aware Scheduling gives 35.38%, 26.56% and 14.68% better results on average for rare, medium and frequent accesses respectively, than First Fit Allocation. For a data set with a wider range of accesses, $10 < \tau^N < 40$, complimentary tasks achieve a performance increase of 26.11% on average.

However, evaluating the two algorithms for their acceptance ratios reveals that First Fit Allocation yields superior results than complementary tasks. These results become more pronounced in higher access frequencies, as depicted in Table II, where First Fit Allocation gives up to 15.3% better schedulability of task graphs. This can be justified by the fact that First Fit Allocation based on Symmetric Response-Time Analysis is a schedulability oriented technique, i.e., it ensures the schedulability of each task while assigning the cores, whereas, Dynamic Task Aware Scheduling is designed for contention reduction, therefore giving reduced response times. For the same reason, the increase in response times of tasks is more evident in Dynamic Task Aware Scheduling than First Fit allocation. The overall performance comparison of the two algorithms is given in Table II.

TABLE II
PERFORMANCE COMPARISON

Scheduling Techniques	Average Response Time			Acceptance Ratio		
	L	M	H	L	M	H
DTAS	2872	3339	4055	1	0.98	0.75
FF	4390	4502	4787	1	1	0.865

VI. CONCLUSION

This paper proposes an integrated platform, COMFAST, for the comparative analysis of different scheduling techniques by observing them for similar data and by developing an experimental setup in MATLAB with generic shared resources in order to incorporate accesses to all types of shared resources. We used COMFAST to conduct a comparative examination of two recently published techniques: Dynamic Task Aware Scheduling and First Fit Allocation using Symmetric Analysis. Dynamic Task Aware Scheduling uses complementary tasks for parallel execution to ensure minimum conflict between tasks at a particular instant and ensures a balanced distribution over all cores. First Fit Allocation, however, uses a symmetric approach to calculate response times in terms of the time spent on the core and the shared resource and then uses it to check the feasibility of the task graphs. Our experiments show that for a deadline monotonic system, Dynamic Task Aware Scheduling gives considerably lower response times for low, medium and high frequencies of resource accesses, with

the performance difference becoming more pronounced as the number of resource accesses decrease. The experiments also reveal the exceptional acceptance ratios generated by First Fit Allocation for a wide range of resource accesses.

In the future, we will focus on utilizing COMFAST for optimizing these algorithms by using the contention reducing features of the two algorithms to boost each other's performance. We plan to explore the effects of using distributions other than Gaussian distribution for setting the execution times of tasks, as our results may be dependent on the specific distribution we have chosen. We also intend to include dynamic prioritization while examining the efficiency of different scheduling techniques for reducing contention in the subsequent projects. We would also like to use this platform to perform comparative investigations on more scheduling techniques to better identify the features that contribute most towards diminishing contention and reducing overall response times.

REFERENCES

[1] J. Fruehe, "Multicore processor technology," *Reprinted from Dell Power Solutions www. dell. com/powersolutions*, pp. 67–72, 2005.

[2] J. Jalle, M. Fernandez, J. Abella, J. Andersson, M. Patte, L. Fossati, M. Zulianello, and F. J. Cazorla, "Bounding resource contention interference in the next-generation microprocessor (NGMP)," in *European Congress on Embedded Real Time Software and Systems*, 2016.

[3] A. Monot, N. Navet, B. Bavoux, and F. Simonot-Lion, "Multicore scheduling in automotive ECUs," in *Embedded Real Time Software and Systems*, 2010.

[4] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto, "Survey of scheduling techniques for addressing shared resources in multicore processors," *ACM Computing Surveys*, vol. 45, no. 1, p. 4, 2012.

[5] M.-L. Chiang, C.-J. Yang, and S.-W. Tu, "Kernel mechanisms with dynamic task-aware scheduling to reduce resource contention in NUMA multi-core systems," *Journal of Systems and Software*, vol. 121, pp. 72–87, 2016.

[6] W.-H. Huang, J.-J. Chen, and J. Reineke, "MIRROR: symmetric timing analysis for real-time tasks on multicore platforms with shared resources," in *Design Automation Conference*, 2016, p. 158.

[7] A. Merkel, J. Stoess, and F. Bellosa, "Resource-conscious scheduling for energy efficiency on multicore processors," in *European conference on Computing Systems*, 2010, pp. 153–166.

[8] A. Schranzhofer, R. Pellizzoni, J.-J. Chen, L. Thiele, and M. Caccamo, "Worst-case response time analysis of resource access models in multicore systems," in *Design Automation Conference*, 2010, pp. 332–337.

[9] Y. Li, V. Suhendra, Y. Liang, T. Mitra, and A. Roychoudhury, "Timing analysis of concurrent programs running on shared cache multi-cores," in *IEEE Real-Time Systems Symposium*, 2009, pp. 57–67.

[10] R. Pellizzoni, A. Schranzhofer, J.-J. Chen, M. Caccamo, and L. Thiele, "Worst case delay analysis for memory interference in multicore systems," in *Design, Automation and Test in Europe*, 2010, pp. 741–746.

[11] R. L. McGregor, C. D. Antonopoulos, and D. S. Nikolopoulos, "Scheduling algorithms for effective thread pairing on hybrid multiprocessors," in *IEEE International Parallel and Distributed Processing Symposium*, 2005, pp. 10–pp.

[12] J. Choi, D. Kang, and S. Ha, "Conservative modeling of shared resource contention for dependent tasks in partitioned multi-core systems," in *Design, Automation & Test in Europe Conference & Exhibition*, 2016, pp. 181–186.

[13] I. Khera and A. Kakkar, "Comparative study of scheduling algorithms for real time environment," *International journal of computer applications*, vol. 44, no. 2, pp. 5–8, 2012.

[14] D. McNulty, L. Olson, and M. Peloquin, "A comparison of scheduling algorithms for multi processors," in *International Conf. on Realtime and Network Systems*, 2010.

[15] B. B. Brandenburg, J. M. Calandrino, and J. H. Anderson, "On the scalability of real-time scheduling algorithms on multicore platforms: A case study," in *Real-Time Systems Symposium, 2008*. IEEE, 2008, pp. 157–169.

[16] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *Real-Time Systems Symposium, 2006. RTSS'06. 27th IEEE International*. IEEE, 2006, pp. 101–110.

[17] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996.