


Article

# Affinity-Based Task Scheduling on Heterogeneous Multicore Systems Using CBS and QBICTM

Sohaib Iftikhar Abbasi <sup>1</sup>, Shaharyar Kamal <sup>1</sup>, Munkhjargal Gochoo <sup>2</sup> , Ahmad Jalal <sup>1</sup> and Kibum Kim <sup>3,\*</sup> 

<sup>1</sup> Department of Computer Science, Air University, Islamabad 44200, Pakistan; 181490@students.au.edu.pk (S.I.A.); shaharyar.kamal@mail.au.edu.pk (S.K.); ahmadjalal@mail.au.edu.pk (A.J.)

<sup>2</sup> Department of Computer Science and Software Engineering, United Arab Emirates University, Al Ain 15551, United Arab Emirates; mgochoo@uaeu.ac.ae

<sup>3</sup> Department of Human-Computer Interaction, Hanyang University, Ansan 15588, Korea

\* Correspondence: kikum@hanyang.ac.kr

**Abstract:** This work presents the grouping of dependent tasks into a cluster using the Bayesian analysis model to solve the affinity scheduling problem in heterogeneous multicore systems. The non-affinity scheduling of tasks has a negative impact as the overall execution time for the tasks increases. Furthermore, non-affinity-based scheduling also limits the potential for data reuse in the caches so it becomes necessary to bring the same data into the caches multiple times. In heterogeneous multicore systems, it is essential to address the load balancing problem as all cores are operating at varying frequencies. We propose two techniques to solve the load balancing issue, one being designated “chunk-based scheduler” (CBS) which is applied to the heterogeneous systems while the other system is “quantum-based intra-core task migration” (QBICTM) where each task is given a fair and equal chance to run on the fastest core. Results show 30–55% improvement in the average execution time of the tasks by applying our CBS or QBICTM scheduler compare to other traditional schedulers when compared using the same operating system.

**Keywords:** affinity-based scheduling; Bayesian generative model; high-performance computing; load balancing; parallel computing



**Citation:** Abbasi, S.I.; Kamal, S.; Gochoo, M.; Jalal, A.; Kim, K. Affinity-Based Task Scheduling on Heterogeneous Multicore Systems Using CBS and QBICTM. *Appl. Sci.* **2021**, *11*, 5740. <https://doi.org/10.3390/app11125740>

Academic Editor: Carlos A. Iglesias

Received: 4 May 2021  
Accepted: 18 June 2021  
Published: 21 June 2021

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Multicore systems usually operate in a shared resource environment. They must cooperate in order to process sharable resources, especially the cache. Caches in multicore systems are private as well as shared. Task scheduling in heterogeneous multicore systems is mainly dependent on how the various tasks are distributed among all available cores [1]. The processing power of each core in a heterogeneous system is different, so distributing the workload according to each core's maximum capacity or efficiency is essential. Thus, the computing power of each core has to be considered. Shared caches play a vital role in increasing the throughput of the overall system, but dependent tasks need to be scheduled concurrently [2].

The prime objective of multicore systems is to execute many tasks in considerably less time. In such circumstances, the role of the cache memory is vital and, if tasks are scheduled on a non-affinity basis, the cache miss rate will increase significantly [3]. When a higher priority task requests use of the CPU, the system grants that task processing priority over the currently running lower priority task. In granting this priority, the higher priority task wipes the data of the previously running lower priority task from the cache resulting in wastage of time as the system will have to bring back the same data into the cache from off chip memory when processing of the lower priority task resumes [4]. The affinity-based scheduling of tasks maximizes the cache hit rate which reduces the overall execution time of the total volume of tasks [5,6]. The slower nature of some cores in a

heterogeneous multicore system negatively affects the overall performance of the system. The cores in heterogeneous systems operate at different computing speeds, thus, even after equal distribution of the workload, the slowest core might end up being overloaded while, at the same time, the fastest core remains idle, doing nothing [7].

Heuristics have been proposed for affinity-based scheduling of tasks using task migration in NUMA (non-uniform memory access) multicore systems [8]. The identical or dependent tasks in a multicore system are scheduled on the same core, thereby reducing the cache thrashing while maximizing the cache hit rate [9]. In multicore systems, shared caches operate between the CPU and the off-chip memory, mainly random access memory (RAM). Gollapudi et al. [10] proposed a completely fair scheduler (CFS) that uses classification to allocate the tasks on the cores in a way that tasks scheduled on the same core use the core's shared cache without thrashing the data of other tasks in the cache. Several heuristics are proposed for the concurrent scheduling of tasks that share the same cache content and cooperate to improve cache performance [11,12]. The overhead of shared caches in multicore systems has been considered in the context of WCET (worst case execution time) analysis [13]. Several shared cache partitioning mechanisms have been proposed to reduce cache interference and thrashing [14,15].

Classifying the dependent tasks together into a same cluster using different algorithmic approaches is proposed [16,17]. Algorithms have been proposed to predict the possible dependency ratios among the tasks that are to be processed [18–20]. Some tasks have read or write dependencies on other tasks for certain memory locations; in such cases, the task waits for a certain memory location to be written which is later read by that task and then carries on with its execution [21,22]. The total number of interactions between certain tasks indicates the level of dependency the tasks are having [23]. The affinity relations between multiple queries is checked and closely related queries are assigned to the same cache structure [24]. Affinity scheduling allows multiple tasks to execute more quickly due to locality of the data, cache utilization or interaction with other tasks [25]. Certain approaches are proposed where the conditions are defined on the basis of which the affinity-based tasks are grouped into a cluster, making sure the grouped data have some kind of dependency [26–29]. To ensure load balancing in heterogeneous systems, imbalance between scheduling domains NUMA and SMT (simultaneous multithreading) NUMA is identified and load balancing is performed by the balancing interval of scheduling domains [30]. Static and dynamic load balancing techniques are proposed. In static load balancing, the load is distributed according to the computation speed of processors. In dynamic load balancing, the load is distributed after the computation starts [31]. Tasks in the same cluster are allocated together to the processing cores taking their dependency advantages [32]. Dependent data clustered together is beneficial as all the data can be processed using the same resources at once [33,34].

Several affinity techniques on a number of realistic applications are presented in [35]. Jia et al. [36] presented a task scheduling mechanism where dependent tasks are scheduled concurrently on the same core ensuring memory affinity, while reducing task switching overhead. Regueira et al. [37] presented a technique which enables efficient utilization of NUMA systems by assigning tasks to a set of cores taking into account the communication and synchronization times between tasks. The proposed scheme calculated the communication cost and the synchronization cost between the tasks. Johari et al. [38] performed load balancing and scheduling tasks on cores by maintaining a queue (sequential search, random search, random search with spatial locality, average method) and without maintaining the queue (sequential, random and random search with spatial locality). Muneeswari [39] proposed a technique in which critical and non-critical tasks are classified, where the critical tasks are scheduled, such that re-utilization of the cache is minimized while, for non-critical tasks, round robin scheduling is used. Holmbacka et al. and Gantel et al. [40,41] proposed a task replication mechanism in which tasks are migrated between different cores. When a task is created on one core, a replica of the same task is created on other available cores. This enables a task's replica to become active on a certain core when a task is migrated

to that core. However, existing work does not clearly address the load balancing issue in heterogeneous systems, or, if it intended to do so, the way in which tasks are scheduled onto the cores where the computing power of each core is different is not clearly defined. Although all the cores in a multicore system are given a fairly equal number of jobs to process, still the best potential performance of the system cannot be exploited because some cores perform slowly by comparison with the fastest cores.

In this paper, we propose two task scheduling techniques, named CBS (chunk-based scheduler) and QBICTM (quantum-based intra core task migration) based on even load balancing by considering the processing speed of all the cores. The main contributions of our work are as follows.

1. We use the Bayesian data analysis model to cluster dependent tasks. All similar tasks based on whether they are communicating or they have any kind of dependency are grouped into a single cluster. Using the Bayesian analysis model, we found the dependencies between tasks, given they are communicating or synchronized.
2. Clusters are then allocated to the cores in a heterogeneous multicore architecture ensuring fair load distribution. We propose a chunk-based scheduler (CBS), where the computing speed of each core is determined. All the tasks that are to be processed are divided into variable size chunks equal to the number of cores available. Each chunk contains a number of tasks depending on and commensurate with the processing frequencies of the respective cores. The largest chunk of tasks is allocated to the fastest core while the smallest chunk is given to the slowest core.
3. The second scheduler we propose in our work is the quantum-based intra-core task migration (QBICTM). All the tasks are divided into equal size chunks equal to the number of cores available. We introduced a time quantum at the fastest core giving each task a fair and equal chance to execute at the fastest core. Fastest core in a multicore architecture is the one which operates at a greater maximum frequency than the other available cores. As soon as the quantum expires, the migration of tasks between the cores takes place in a heterogeneous multicore architecture.

We used the Pycharm community IDE development tool to obtain the results. The Python multiprocessing module is used to implement our proposed schedulers. All the results that are explained in detail in the later part of this paper were obtained on the core i5 7th generation multicore system with 8 GB of DDR4 RAM. The multicore system contains four logical and two physical cores. Each physical core is operating at a maximum frequency of 2700 MHz which can be boosted up to 3100 MHz. The maximum operating frequency for each physical core is changed to a different value ensuring that all the cores are now heterogeneous in nature. All the cores in a multicore architecture are utilized to their full capacity, i.e., 100% utilization.

The rest of the paper is organized as follows. Section 2 introduces the methods used in this study. Sections 3 and 4 describe the experimental results and provide a discussion, respectively. Section 5 concludes our work.

## 2. Materials and Methods

Figure 1 shows the structure of our proposed affinity-based CBS and QBICTM schedulers. Affinity-based task scheduling can significantly reduce cache thrashing and for this we used a Bayesian data analysis model. The affinity-based scheduling of the tasks on the cores has already been briefly described in the latter part of Introduction.

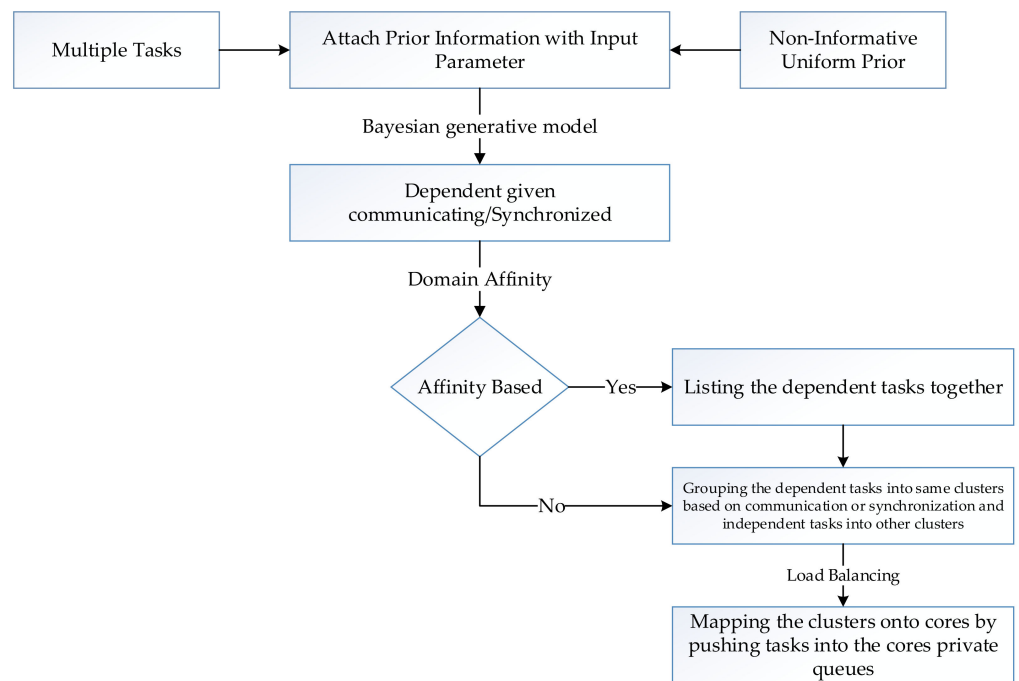


Figure 1. Affinity-based scheduling mechanism via CBS and QBICTM.

2.1. Bayesian Generative Model

The Bayesian data model is used in the research to group identical tasks. We used the Bayesian model because of the advantages it has over other algorithms; these include the use of the prior information that is available with the data. The Bayesian model analyzes the data and makes use of previously available information and forms priors for future analysis. It works on the basis of assumption and estimation. Priors are the information the model contains before seeing the data [42]. We used the non-informative uniform prior for the tasks in this research. The Bayesian generative model does not contain any information about the tasks that are to be processed before seeing the data. The prior dependencies between the tasks have a uniform value between “1% to 100%”.

Figure 2 shows the working mechanism of the Bayesian generative model.

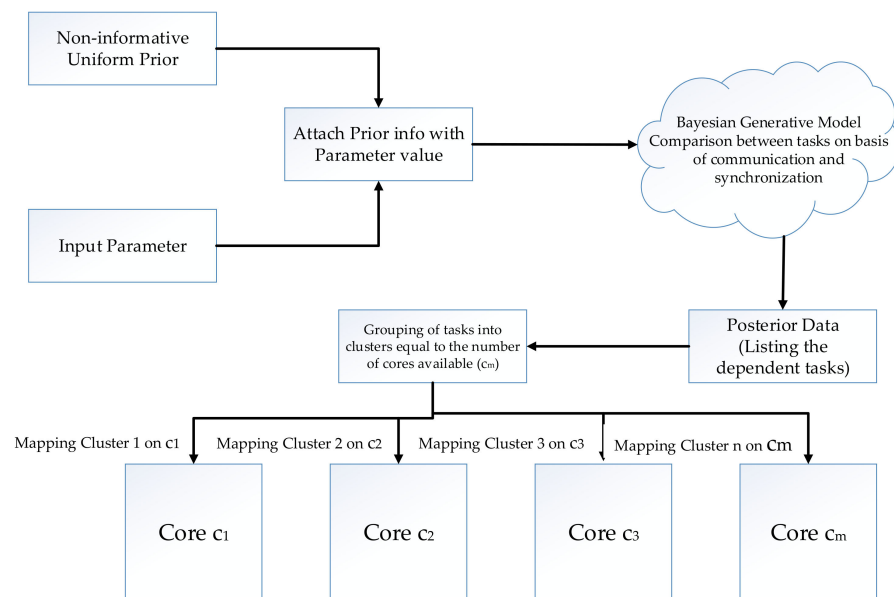


Figure 2. Prior and posterior tasks dependencies using Bayesian generative model.

All incoming tasks contain prior information which is then forwarded to a generative model as input; the information contained in each task is analyzed by the Bayesian model and the similarities between tasks are identified as either the tasks are communicating or synchronized. The generative model produces the posterior data as an output which is affinity-based. The tasks which tend to be similar, given the communication or synchronization between them are grouped together to form a single cluster. The total number of clusters made by the Bayesian data model equals the number of cores present in a heterogeneous multicore system.

$$p(D|C) = \frac{p(C|D)p(D)}{p(C)} \quad (1)$$

$D$  is the dependency and  $C$  is the communication between tasks, where  $p(D|C)$  is the posterior distribution for dependency given that tasks are communicating.

$$p(C) = \int p(C|D)p(D)dD \quad (2)$$

$p(C)$  is the marginal probability, specifying that the tasks are communicating. The Bayes theorem for probability distributions is stated as

$$\text{Posterior} \propto \text{Likelihood} \times \text{Prior} \quad (3)$$

$$p(D) = e^{-D+Q} \quad (4)$$

where  $p(D)$  is the prior distribution for parameters and  $Q$  is an arbitrary constant.

$$p(D) \propto e^{-D} \quad (5)$$

$$\int_0^{\infty} e^{-D+Q}dD = -\frac{1}{e^{D-Q}} + e^Q \quad (6)$$

We assumed different ratios of dependent and independent tasks and, using the prior information contained with the tasks, we input them into a generative model to find the existing dependencies between tasks on the basis of communication or synchronization. The detailed results are presented in Section 3.

## 2.2. Chunk-Based Scheduler (CBS)

We performed load balancing on a heterogeneous multicore system by proportionately dividing the total load into multiple chunks of variable sizes according to the various processing speeds of all the respective cores. The Bayesian generative model gives the list of dependent and independent tasks in the form of posterior data as an output. All tasks, whether dependent or independent, are listed together without making clusters. Similar or dependent tasks are listed in a contiguous manner. All tasks that are to be processed are present in a single list that functions like a queue, where each task is requesting allocation to the core in order to be processed. Tasks are then divided into chunks that may differ in size according to the respective processing speeds of the cores.

The computing power or capacity of each core in a heterogeneous system is pre-determined, with each core taking a job or task from its private queue. The largest chunk of tasks is allocated to the fastest core. All the tasks in a chunk are put into the core's private queue. In the same way, each core gets a chunk of tasks with respect to its computing capacity, the smallest chunk being given to the slowest core in the heterogeneous multicore system. This results in each core being proportionately loaded, the fastest core being more heavily loaded and the slowest core processing fewer jobs or tasks, each according to its capacity. Figure 3 shows how the CBS technique is applied in the system.

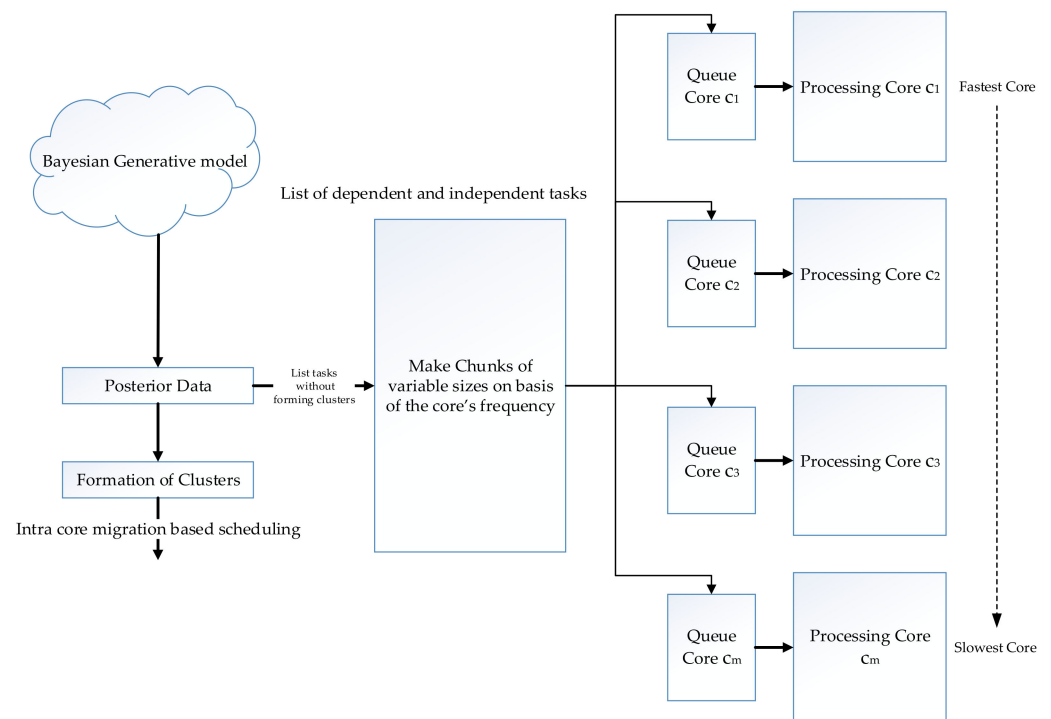


Figure 3. Allocation of tasks to cores by CBS.

Workloads consisting of multiple tasks are not distributed evenly and fairly across the cores because that would result in some cores being overloaded but because of the heterogeneous nature of the system under use, the fastest core which processes tasks at much higher speeds receive more of the load. The proportionate allocation of more tasks to the faster cores and fewer to the slower cores means all cores are more fully utilized and jobs or tasks are more likely to be completed in the same time. Ideally, no core is overloaded and no core is left idle. The fastest core might be idle when other cores are still operating and processing the tasks but we have made sure that the fastest core has already executed a larger portion of the tasks that were to be processed.

Table 1 provides the description of the terms used in the research.

Table 1. Description of the terms used.

Notation	Description
$C = \{c_1, c_2, \dots, c_m\}$	Set of cores in a heterogeneous multicore architecture
$c_e$	Total cores available
$T = \{t_1, t_2, \dots, t_t\}$	Set of tasks
$t_t$	Total tasks to be processed
$C_k = \{k_1, k_2, \dots, k_c\}$	Chunks of tasks equal to the number of cores available
$F = \{f_1, f_2, \dots, f_e\}$	Computing max frequency of all the cores
$\underline{f_n}$	Total/max processing speed of the system

The number of tasks in each chunk is computed using

$$f_n = \sum_{i=1}^e f_i \tag{7}$$

$$\begin{aligned} pc_1 &= c_1 f_1 \\ pc_2 &= c_2 f_2 \\ &\vdots \\ pc_p &= c_m f_e \end{aligned} \tag{8}$$

Each chunk is of variable size and contains different number of tasks.

$$\begin{aligned} k_1 &= \left\lceil \frac{p_{c_1}}{f_n} \times t_t \right\rceil \\ k_2 &= \left\lceil \frac{p_{c_2}}{f_n} \times t_t \right\rceil \\ &\vdots \\ k_c &= \left\lceil \frac{p_{c_p}}{f_n} \times t_t \right\rceil \end{aligned} \quad (9)$$

Algorithm 1 describes the working mechanism of CBS. All the tasks that are to be processed are divided into various-sized chunks. The next step is to calculate the processing speed of each of the cores in the multicore architecture. Once the processing speed of each of the cores is determined, the chunks are allocated to the cores by considering each core's speed, i.e., workload is allocated proportionate to each core's processing speed. The fastest core in a multicore architecture gets the largest chunk of tasks, the smallest chunk being allocated to the slowest core.

---

**Algorithm 1** Scheduling tasks on the cores using CBS algorithm

---

Inputs:	List of tasks, Mapping of chunks $k_c$ onto the set of Cores C
Outputs:	Computation of overall execution time of set of Tasks T
BEGIN	
1.	List tasks in the output generated by Bayesian generative model
2.	Compute processing speed of each core $c_1-c_m$ in set of Cores C
3.	Total_frequency = $\sum (\text{freq}(c_1, c_2, \dots, c_m))$
4.	Make Chunks $k_c$ of Tasks T = Number of available Cores C
5.	Chunk $k_c = \left\lceil \frac{c_m \text{ processing speed}}{\text{Total\_frequency}} \right\rceil * \text{Number of tasks in list}$
6.	Create Child of Parent process
7.	Largest Chunk $k_c$ is queued or piped at the fastest core $c_m$
8.	Smallest chunk being queued on the slowest core
9.	foreach core $c_m$ do
10.	Process each task $t_t$ from core's queue until no task left for processing in the queue
11.	end for
END	

---

### 2.3. Quantum Based Intra Core Task Migration Load Balancing Scheduler

Fair and even load balancing is performed on a heterogeneous system by using a quantum based intra core migration of tasks. The working mechanism of the quantum based intra core migration of tasks is presented in Figure 4.

The processing speed of each core in a multicore architecture is computed. The cores are sorted in increasing order according to their respective processing speeds. Each task which is to be processed on a multicore system is given a fair and equal chance to run on the fastest core. A time quantum is introduced at the fastest core and tasks are being processed on a first come first served basis from each core's own private queue. Each core in a heterogeneous multicore system owns a private queue from where the tasks are allocated to a core and which functions like a ready queue for that core. The maximum and minimum threshold is defined for each core's queue for the total number of tasks present and as soon as the number of tasks in any queue is less than the defined minimum threshold, an intra core migration of tasks occurs. Every running task on each core is migrated to the next corresponding core's ready queue as shown in the Figure 4. When a task utilizes its quantum while running on the fastest core, the CPU is preempted from the task and it is migrated to the ready queue of the next in line processing core, i.e., 2nd fastest core. When the CPU is preempted from a task running on the fastest core, the number of tasks in the queue of the fastest core falls below the minimum threshold. This is because the next task waiting in the queue will be allocated to the fastest core. In that case, the task

running on the slowest core is migrated to the fastest core’s private queue or ready queue and the current state of the task is saved.

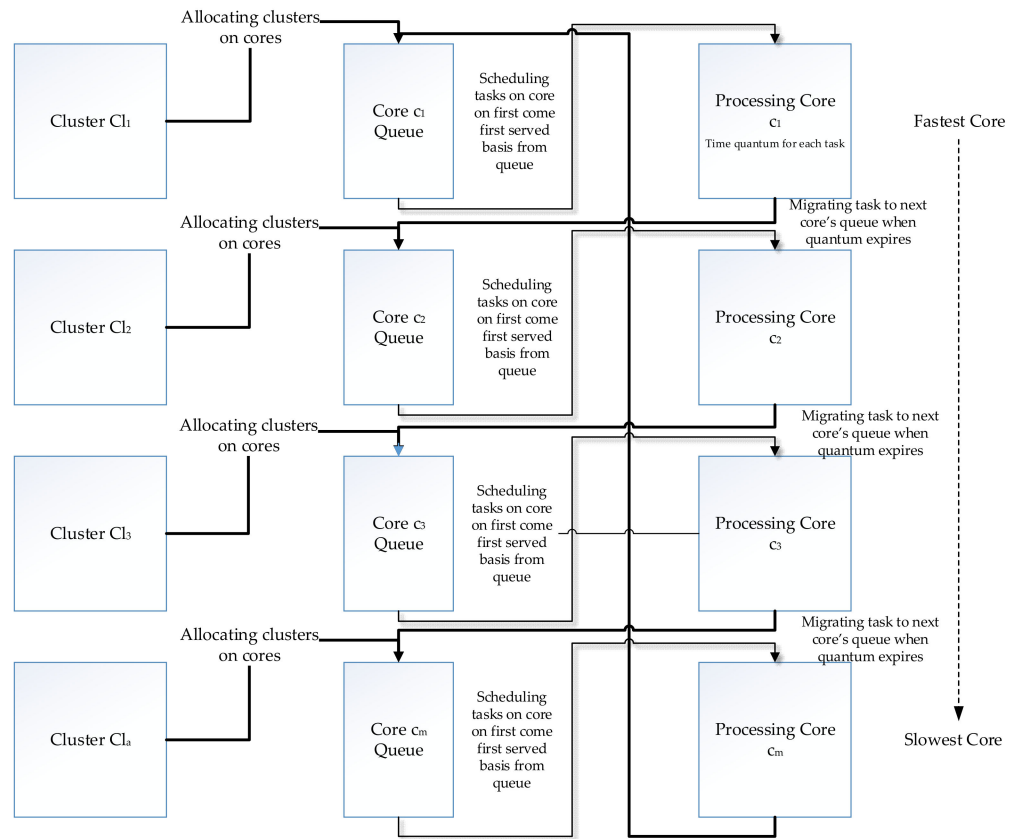


Figure 4. Working mechanism of QBICTM scheduler.

The total number of tasks in each queue of the multicore system is computed as

$$Cl_a = \left\lceil \frac{t_t}{c_c} \right\rceil \tag{10}$$

Whenever a quantum of a certain task running on the fastest core expires, the number of tasks in the queues of all the cores of a heterogeneous system fall below a defined minimum threshold and intra core migration of the tasks takes place. Using the described technique helps us eliminate problems related to load balancing on heterogeneous systems. The load balancing issue in heterogeneous systems, vis-à-vis that the fastest core finishes the tasks allocated much more quickly than the slowest core in the system, is addressed in our research and has been minimized. Our proposed technique ensures that the fastest core will always process a task as long as other cores are operating and the tasks are available for processing. Thus, the problem we dealt with to some extent is that one core may be under-loaded or idle while another core may be over-loaded even after fair and equal distribution through load balancing in heterogeneous systems.

Algorithm 2 describes the QBICTM scheduling of tasks. When a quantum of a certain task expires the intra core migration of tasks takes place and the next task waiting is allocated to the fastest core.



**Algorithm 2** Load Balancing using QBICTM scheduler algorithm

---

```

Inputs:      Allocating each cluster  $cl_a$  to a Core  $c_m$ 
Outputs:    Computation of overall execution time of set of Tasks T
BEGIN
1.          Make clusters of tasks = Number of cores available
2.          Tasks in each Cluster  $Cl_a = \left\lceil \frac{\text{Total Number of tasks}}{\text{Total Cores available}} \right\rceil$ 
3.          Compute processing speed of each core  $c_m$  in set of Cores C
4.          Allocate one cluster  $cl_a$  to exactly one Core  $c_m$  ensuring load balancing with each core getting fair and equal
           number of tasks to process
5.          Tasks T in each cluster  $cl_a$  are queued onto the core  $c_m$ 
6.          Introduce a time quantum at the fastest core, each task getting fairly equal time for processing at fastest core in
           one cycle
7.          When time quantum for a task expires, preempt core  $c_m$  from the task  $t_i$  and allocate next task waiting in
           queue onto the core
8.          for each core  $c_m$  do
9.              Define a minimum threshold for each core's queue
10.             if total_number_of_tasks in core  $c_m$  queue < threshold then
11.                 Migrate task  $t_i$  being processed on every core C to the queue of next corresponding core  $c_m$ 
12.                 Continue processing tasks T until there are no tasks left to process
13.             end if
14.          end for
END

```

---

### 3. Results

In this section, we evaluate the performance of our proposed methodology. The chunk-based scheduling technique and quantum based intra core task migration scheduler are applied to two programs, namely, a factorial program and a real-life working example (ingredients ratio for baking a cake). For the factorial program we calculated execution time for an individual number, as well as for a process (a range of numbers). Real-life working example is a linear program for baking a cake with four ingredients, which are flour, butter, eggs and sugar. The scenario for the execution of a program is as follows

- All the ingredients weigh exactly 700 g in total.
- Amount of butter was half that of sugar.
- Combined weight of flour and eggs was at most 450 g.
- Weight of eggs and butter combined was at most 300 g.
- The combined weight of eggs and butter was at most that of flour plus sugar.

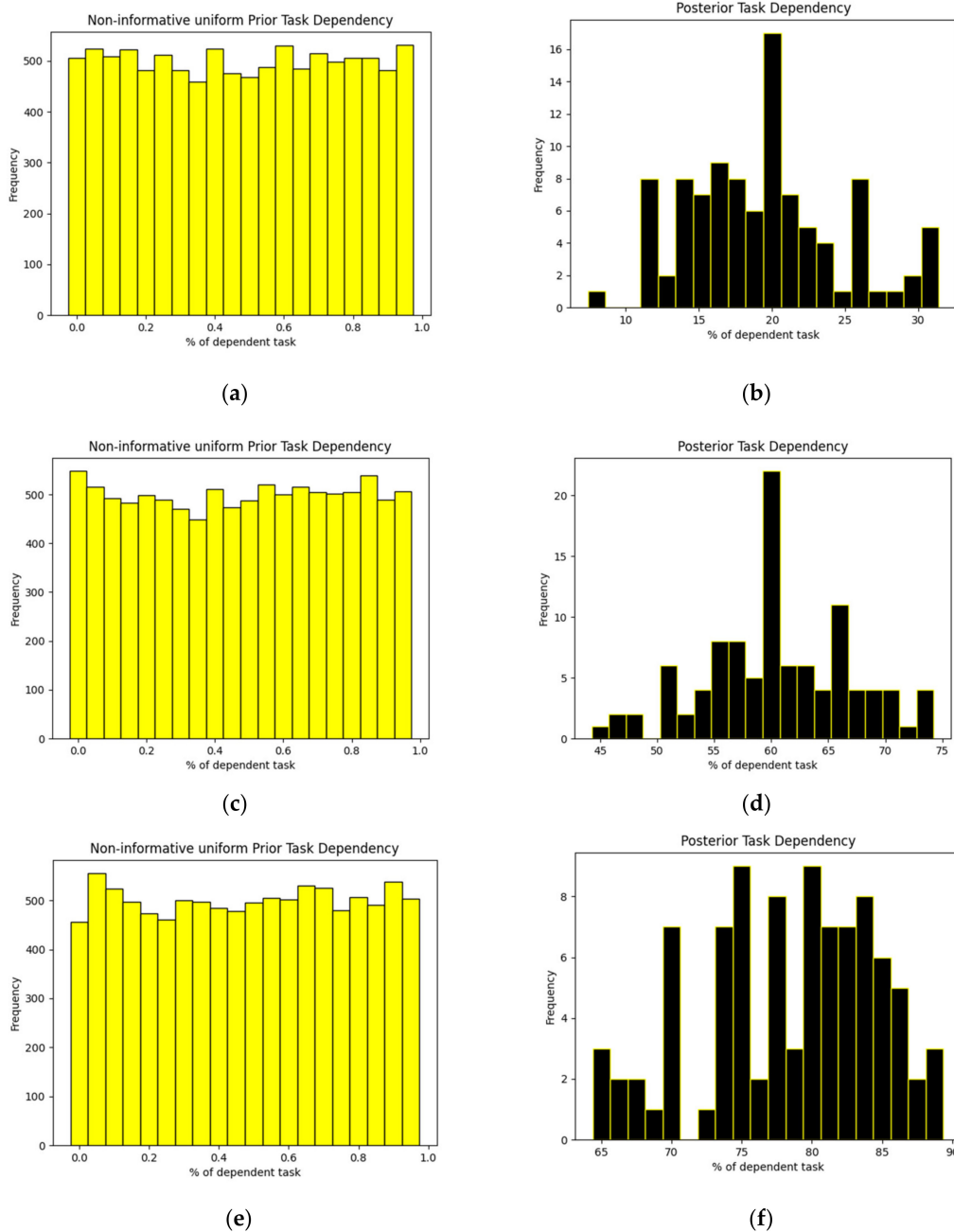
The program calculates the weight of each individual ingredient after processing multiple tasks, which have read-write or write-read dependencies. The characteristics of the system on which we have performed experiments are given in Table 2.

**Table 2.** System specifications.

Technology	Specification
Processor	Intel Core i5-7200 U @ 2.70 GHZ
Operating System	LINUX, Windows 10 Pro
Sockets	1
Cache Levels	L1: 128 KB
	L2: 512 KB
	L3: 3.0 MB
RAM	8 GB DDR4
Core Count	4 logical, 2 Physical

We have applied Bayesian data analysis on a factorial program. The input values for the factorial program contain different ranges. We have taken different parameter values based on assumptions for, e.g., if there are 100 tasks to be processed, we assumed 30 of

those are dependent while 70 tasks have no dependencies. We find the probability based on the task dependencies given they are communicating or synchronized or both out of those 30 dependent tasks based on the assumptions. The posterior data generated as an output by the Bayesian model show different ranges, as shown in Figure 5. We set the non-informative prior for task dependency as uniform where the dependency ratio of the tasks can be any value between 0 and 1. There might be 100% task dependency or it can also be 0%. The different parameters are given as an input to the Bayesian generative model. The total of 100 iterations were run because we took each value in the range of 1 to 100 as an input parameter.



**Figure 5.** Prior and posterior task dependencies based on different parameter values. (a) Non-informative uniform prior when parameter value is 20; (b) posterior tasks dependency for parameter value 20; (c) non-informative uniform prior when parameter value is 60; (d) posterior tasks dependency for parameter value 60; (e) non-informative uniform prior when parameter value is 80; (f) posterior tasks dependency for parameter value 80.

Table 3 shows the results for six different parameter values. Different parameter values are given as an input to the Bayesian generative model along with non-informative uniform priors.

**Table 3.** Task dependency ratios for a factorial program using the Bayesian model.

Parameter (%)	Dependency Ratio
20	12–31
40	26–54
50	36–63
60	47–74
80	66–88
90	79–96

The Bayesian generative model takes different parameter values as inputs and calculates the probability of the task's dependency based on the parameter value. The dependent tasks based on the communication or synchronization are listed together by the generative model, which are later grouped together in single or multiple clusters.

Once the dependent tasks are listed separately, we make the clusters of similar and independent tasks. The number of clusters equals the number of cores in a heterogeneous multicore architecture. Each output generated by the Bayesian analysis model for different parameters is listed and clusters are made one by one. Now, the mapping of the clusters onto the cores of the multicore architecture is initiated. Two different techniques are applied to ensure maximum load balancing on a heterogeneous system. The first technique used for load balancing puts the tasks from all clusters into a queue where similar or dependent tasks are listed in continuous order. We applied the technique on a factorial program by allocating a larger bunch of tasks to the fastest core based on its processing power. Results show an improvement in the execution time for each individual task and the process as a whole. The throughput of the overall system also increased whereas the waiting time for each task decreased significantly. Table 4 shows the average execution time of the tasks for our chunk-based technique which is used to find the factorial for a given range of numbers by taking different input parameters from the Bayesian model.

**Table 4.** Average execution time for the factorial program based on different input parameters.

Parameter	Range	Execution Time (Seconds)	
		CBS	OS Scheduler
20	1–3000	2.17	3.18
	5000–7000	8.56	21.60
	12,000–15,000	66.86	172.80
	16,000–18,000	88.37	173.75
40	1–3000	2.07	3.05
	5000–7000	8.40	21.54
	12,000–15,000	66.57	147.76
	16,000–18,000	86.68	173.40
60	1–3000	1.98	2.77
	5000–7000	8.31	20.89
	12,000–15,000	65.04	144.76
	16,000–18,000	70.05	155.38
80	1–3000	2.02	2.93
	5000–7000	8.31	21.20
	12,000–15,000	65.13	146.13
	16,000–18,000	70.21	171.30

Table 4. Cont.

Parameter	Range	Execution Time (Seconds)	
		CBS	OS Scheduler
90	1–3000	2.00	2.95
	5000–7000	8.37	21.22
	12,000–15,000	66.47	147.38
	16,000–18,000	75.92	171.66

We took each single output from the Bayesian model based on the input parameters and made clusters from identical tasks. We computed the execution time for the factorial of each individual number in the given range listed in the Table 4 and computed the overall execution time for the whole program. Figure 6 shows a comparison of the average execution time for the factorial program for the different range of numbers based on the input parameter value.

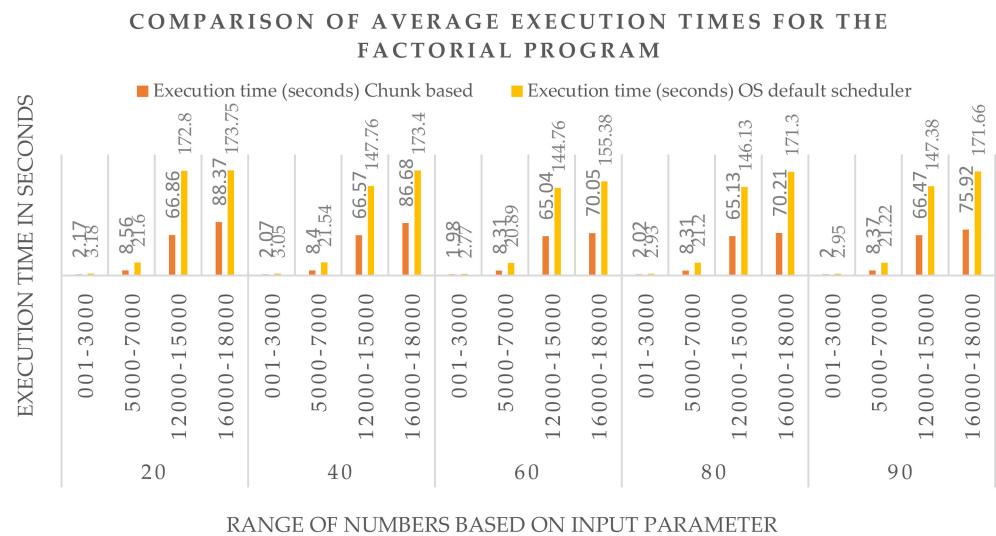


Figure 6. Average execution time comparisons between CBS and OS scheduler for the factorial program.

Results show an improvement in the overall average execution time by applying our chunk-based technique compared to the default operating system scheduler. The average execution time for the whole factorial program was reduced by 32–50% compared to the default OS scheduler. The CBS load balancing technique decreases the total execution time of the factorial program compared to the schedulers used by the operating system.

The results for the average execution time for the factorial program by scheduling the clusters on cores and applying the QBICTM scheduling mechanism by introducing a quantum at the fastest core are presented in Table 5.

We took different values for the time quantum and found the factorial for a different range of numbers. The quantum based intra-core migration technique is applied on each task generated as output by the Bayesian generative model based on the input parameter values. A time quantum is introduced at the fastest core and each single task gets a fair and equal chance to be processed at the fastest core. When the quantum expires, the intra core migration of tasks takes place as discussed earlier in Section 2.3. Our load balancing technique outperformed the default OS scheduler in terms of the execution time for the factorial program as it processed the whole program in significantly less time. The reduction in overall execution time for different ranges of numbers is between 30–55%.

**Table 5.** Average execution time for the factorial program computed using QBICTM scheduling.

Parameter	Range	Execution Time in Seconds for Different Time Quantum's				OS Scheduler
		Quantum = 0.5 s	Quantum = 1 s	Quantum = 2 s	Quantum = 4 s	
20	1–3000	2.98	2.52	2.58	1.43	3.18
	5000–7000	7.84	9.45	9.96	7.87	21.60
	12,000–15,000	72.43	70.61	69.42	66.57	172.80
	16,000–18,000	103.15	93.12	95.27	87.59	173.75
40	1–3000	2.31	2.27	2.07	1.79	3.05
	5000–7000	8.89	8.04	8.94	8.56	21.54
	12,000–15,000	71.12	68.47	64.80	65.11	147.76
	16,000–18,000	76.51	74.73	70.17	67.56	173.40
60	1–3000	2.24	1.95	1.99	1.91	2.77
	5000–7000	8.53	8.84	7.96	8.09	20.89
	12,000–15,000	68.65	65.35	65.14	65.62	144.76
	16,000–18,000	73.12	73.05	71.42	66.64	155.38
80	1–3000	2.84	2.48	2.11	1.90	2.93
	5000–7000	9.21	8.72	8.25	7.31	21.20
	12,000–15,000	66.85	66.79	66.06	66.52	146.13
	16,000–18,000	87.17	86.34	84.27	81.65	171.30
90	1–3000	2.34	2.12	1.81	1.58	2.95
	5000–7000	8.74	8.59	8.27	7.60	21.22
	12,000–15,000	67.88	65.68	64.40	63.12	147.38
	16,000–18,000	75.46	75.95	74.46	69.05	171.66

Smaller values for the quantum benefit tasks that require to be processed for shorter amounts of time but they increase the average waiting time for tasks that require larger amounts of time on the CPU. As we increase the quantum, the larger tasks benefit and waiting time is reduced. The quantum-based technique increased the context switching of the tasks and it thereby under-performed for some tasks when the quantum value was smaller compared to the chunk-based technique, but execution time slightly improved as compared to the chunk-based scheduler when the quantum value was increased. As shown in Table 6, we have executed different numbers of tasks for a real-life working example, namely, the ingredients ratio for baking a cake where there are multiple dependent and independent tasks.

**Table 6.** Average Execution time for the ingredients ratio for baking a cake, calculated using the CBS technique.

No. of Tasks to Process	Execution Time for Tasks in Seconds	
	CBS	OS Scheduler
7000	12.39	33.91
10,000	21.63	37.24
16,000	29.96	58.65
25,000	45.92	92.35
50,000	85.88	174.65

We first list the dependent tasks into a group by using the Bayesian model so that, while making chunks, all the dependent tasks are listed serially in contiguous manner. Results show that by using a chunk-based scheduling technique, we are able to reduce the overall execution time of the tasks by an average of 45–51% compared to the scheduler used by the operating system. Figure 7 shows the average execution time comparison between the CBS and the OS schedulers.

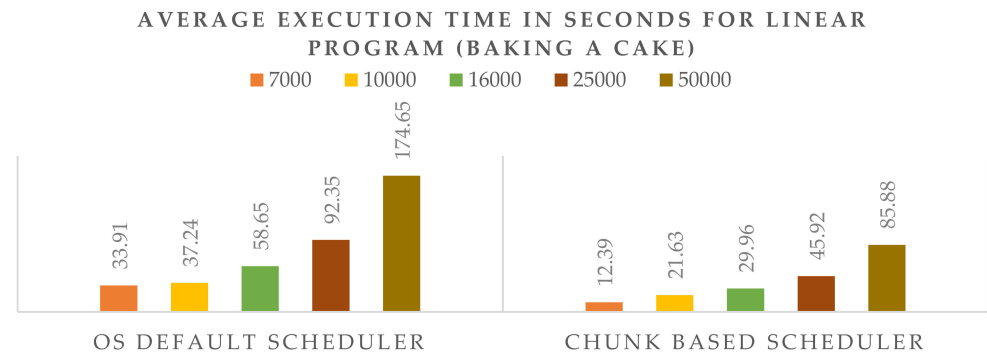


Figure 7. Comparison of average execution times between the CBS and the OS schedulers.

Quantum based scheduling is applied on a real-life working example and the results are shown in Table 7.

Table 7. Average Execution times for ingredients ratios for baking a cake calculated using the QBICTM scheduling technique.

No. of Tasks to Process	Execution Time in Seconds		
	QBICTM Scheduler		OS Scheduler
	Quantum = 2 s	Quantum = 4 s	
7000	13.17	11.06	33.91
10,000	21.09	21.13	37.24
16,000	29.91	28.19	58.65
25,000	47.03	44.37	92.35
50,000	83.01	81.10	174.65

We computed the execution time for different numbers of tasks in a process. Results show an improvement of 44–51% in the overall execution time compared to the traditional operating system scheduler. Figure 8 shows the average improvement rate in the execution time. We applied the chunk-based scheduling and quantum-based scheduling by comparison with the traditional scheduler used by the operating system. The overall execution time of the tasks improved quite significantly.

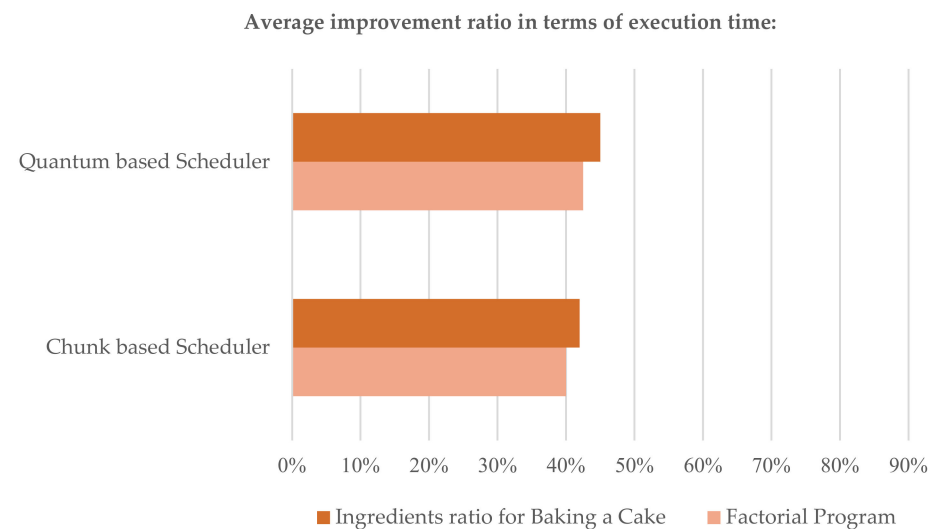


Figure 8. Average improvement ratio in execution time by comparison with the traditional OS scheduler.

Table 8 shows a comparison for average improvement in throughput using different task scheduling techniques in comparison with the schedulers used by the running operating system.

**Table 8.** Comparison of average improvement in throughput.

Approach or Technique	Processed Data	Core Count	Average Improvement
Song [5]	CFD Kernel and Cholesky Factorization	4, 8, 16, 32	25% to 42%
Ozturk [24]	WL1 and WL2 (containing multiple queries)	2 Sockets (6 cores each)	15% to 19%, 25% for individual query
Regueira [43]	Heat Transfer, Workflow and Quick Sort	6, 24, 32, 64	11.1% to 11.3%
CBS and QBICTM	ILP, Factorial Program	4	30% to 55%

Results show the effectiveness of our proposed schedulers in comparison with other existing state-of-the-art schedulers. The average improvement in the throughput using our proposed schedulers can further be improved if the proposed technique is applied on a multicore machine having a higher core count.

#### 4. Discussion

The proposed schedulers in this study aim to distribute the workload according to each core's maximum capacity or efficiency. As the cores in the heterogeneous systems usually have different computing speeds, the equal division of the workload might not be as effective as in homogeneous systems. Most of the previous works focus on distributing equal and even workloads among cores but they do not clearly address the load balancing mechanism on heterogeneous systems.

The experimental results conducted in Section 3 prove the effectiveness of our proposed schedulers. We have presented two different scheduling techniques to ensure load balancing on a heterogeneous system. The load balancing mechanism is based on fair distribution of workload by keeping the processing power of multiple cores in consideration rather than just equal distribution of tasks.

By comparison with other proposed techniques, our affinity-based schedulers have several limitations which direct the future study of our work. The affinity-based scheduling of tasks maximizes the data reuse potential in shared caches. In future work, we will focus on improving the cache hit ratio thus reducing the cache miss penalty. In the QBICTM scheduling technique, the context switching rate significantly increases whenever a quantum at the fastest core expires and the migration of tasks between the cores takes place. We will focus on minimizing the context switching of tasks, which will further improve the average execution time of an overall program.

Despite the mentioned limitations, our proposed schedulers applied load balancing fairly on the heterogeneous system. With the aim of achieving the maximum performance of the system on offer, our proposed schedulers focus on maximizing the utilization of all the processing cores in multicore architectures.

#### 5. Conclusions

We proposed two techniques for scheduling tasks on heterogeneous multicore systems. All the dependent tasks are listed together using a Bayesian analysis model in which the non-informative uniform prior is used. All the input parameters are attached with the prior and passed onto the generative model as an input. The posterior data generated by the Bayesian model contains all the dependent tasks in a list in a contiguous manner and those tasks are passed on to the schedulers. Our CBS lists all the tasks together where dependent tasks are present in a contiguous manner. We then create a number of chunks of tasks of variable sizes, equal to the number of cores available. The largest chunk is assigned to the

fastest core and the smallest chunk is allocated to the slowest core in the heterogeneous architecture.

QBICTM is the second proposed load balancing scheduler in our study. Here we have introduced a time quantum at the fastest core which enables each task to have a fair and equal chance of being processed at the fastest core. The migration of tasks is performed by defining a minimum threshold in each core's queue. When the total number of tasks in a certain queue falls below the defined minimum threshold, the tasks currently being processed are migrated to the queue of the next corresponding core. The only problem with the quantum based intra-core migration of tasks is that it increases the number of context switches, however, it still significantly out-performs the traditional operating system scheduler in terms of the execution and task waiting times.

Our load balancing schedulers outperformed the default OS scheduler in terms of execution time for the factorial program as it processed the whole program in significantly shorter time. The reduction in overall execution time for a process containing different range of numbers is between 30–55% while for an individual task the execution time is reduced by an average of 45–51% using our proposed affinity-based schedulers compared to the scheduler used by the operating system for the real-life example of baking a cake. For future work in this regard, the cache hit rate can be maximized while reducing the cache miss penalty by grouping the tasks into clusters on the basis of the maximum interactions between them.

**Author Contributions:** Conceptualization, S.I.A., S.K.; methodology, S.I.A., S.K., and M.G.; software, S.I.A.; validation, M.G., A.J. and K.K.; formal analysis, K.K. and M.G.; resources, A.J. and K.K.; writing—review and editing, M.G., and K.K.; funding acquisition, M.G., A.J. and K.K. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF), funded by the Ministry of Education (No. 2018R1D1A1A02085645). This work was also supported by the Korea Medical Device Development Fund grant funded by the Korean government (the Ministry of Science and ICT, the Ministry of Trade, Industry and Energy, the Ministry of Health & Welfare, the Ministry of Food and Drug Safety) (Project Number: 202012D05-02).

**Institutional Review Board Statement:** Not Applicable.

**Informed Consent Statement:** Not Applicable.

**Data Availability Statement:** Not Applicable.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Li, M.; Chen, C.; Zhu, G.; Savaria, Y. Local Queueing-Based Data-Driven Task Scheduling for Multicore Systems. In Proceedings of the 2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS), Windsor, ON, Canada, 5–8 August 2018; pp. 897–900.
2. Akram, N.; Zhang, Y.; Ali, S.; Amjad, H.M. Efficient Task Allocation for Real-Time Partitioned Scheduling on Multi-Core Systems. In Proceedings of the 2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST), Islamabad, Pakistan, 8–12 January 2019; pp. 492–499.
3. Guo, Z.; Zhang, Y.; Wang, L.; Zhang, Z. Work-in-Progress: Cache-Aware Partitioned EDF Scheduling for Multi-Core Real-Time Systems. In Proceedings of the 2017 IEEE Real-Time Systems Symposium (RTSS), Paris, France, 5–8 December 2017; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2018; Volume 2018, pp. 384–386.
4. Xu, M.; Phan, L.T.X.; Choi, H.-Y.; Lee, I. Analysis and Implementation of Global Preemptive Fixed-Priority Scheduling with Dynamic Cache Allocation. In Proceedings of the 2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Vienna, Austria, 11–14 April 2016; pp. 1–12.
5. Song, F.; Moore, S.; Dongarra, J. Analytical Modeling for Affinity-Based Thread Scheduling on Multicore Platforms. In Proceedings of the 2009 IEEE International Conference on Cluster Computing and Workshops, New Orleans, LA, USA, 31 August–4 September 2009; pp. 1–10.
6. Sibai, F.N. Performance Effect of Localized Thread Schedules in Heterogeneous Multi-Core Processors. In Proceedings of the 2007 Innovations in Information Technologies (IIT), Dubai, United Arab Emirates, 18–20 November 2007; IEEE: Piscataway, NJ, USA, 2007; pp. 292–296.



7. Sharma, R.; Kanungo, P. Dynamic Load Balancing Algorithm for Heterogeneous Multi-Core Processors Cluster. In Proceedings of the Proceedings—2014 4th International Conference on Communication Systems and Network Technologies—CSNT 2014, Bhopal, India, 7–9 April 2014; IEEE Computer Society: Piscataway, NJ, USA, 2014; pp. 288–292.
8. Markatos, E.P.; Leblanc, T.J. Using Processor Affinity in Loop Scheduling on Shared-Memory Multiprocessors. *IEEE Trans. Parallel Distrib. Syst.* **1994**, *5*, 379–400. [[CrossRef](#)]
9. Shende, S.; Malony, A.D.; Morris, A.; Spear, W.; Biersdorff, S. Encyclopedia of Parallel Computing. In *Encyclopedia of Parallel Computing*; Springer: New York, NY, USA, 2011; pp. 2025–2029.
10. Gollapudi, R.T.; Yuksek, G.; Ghose, K. Cache-Aware Dynamic Classification and Scheduling for Linux. In Proceedings of the 2019 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS), Yokohama, Japan, 17–19 April 2019; pp. 1–3.
11. Anderson, J.H.; Calandrino, J.M. *Parallel Real-Time Task Scheduling on Multicore Platforms*; IEEE: Piscataway, NJ, USA, 2006.
12. Calandrino, J.M.; Anderson, J.H. Cache-Aware Real-Time Scheduling on Multicore Platforms: Heuristics and a Case Study. In Proceedings of the Euromicro Conference on Real-Time Systems, Prague, Czech Republic, 2–4 July 2008; pp. 299–308.
13. Chattopadhyay, S.; Kee, C.L.; Roychoudhury, A.; Kelter, T.; Marwedel, P.; Falk, H. A Unified WCET Analysis Framework for Multi-Core Platforms. In Proceedings of the 2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium, Beijing, China, 16–19 April 2012; pp. 99–108.
14. Kim, N.; Ward, B.C.; Chisholm, M.; Fu, C.-Y.; Anderson, J.H.; Smith, F.D. Attacking the One-Out-Of-m Multicore Problem by Combining Hardware Management with Mixed-Criticality Provisioning. In Proceedings of the 2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Vienna, Austria, 11–14 April 2016; pp. 1–12.
15. Berna, B.; Puaut, I. PDPA: Period Driven Task and Cache Partitioning Algorithm for Multi-Core Systems. In Proceedings of the RTNS'12: 20th International Conference on Real-Time and Network Systems, Pont-à-Mousson, France, 8–9 November 2012; pp. 181–189.
16. Basavegowda, H.S.; Dagnev, G. Deep Learning Approach for Microarray Cancer Data Classification. *CAAI Trans. Intell. Technol.* **2020**, *5*, 22–33. [[CrossRef](#)]
17. Alguliyev, R.M.; Aliguliyev, R.M.; Sukhostat, L.V. Efficient Algorithm for Big Data Clustering on Single Machine. *CAAI Trans. Intell. Technol.* **2020**, *5*, 9–14. [[CrossRef](#)]
18. Jiang, R.; Mou, X.; Shi, S.; Zhou, Y.; Wang, Q.; Dong, M.; Chen, S. Object Tracking on Event Cameras with Offline-Online Learning. *CAAI Trans. Intell. Technol.* **2020**, *5*, 165–171. [[CrossRef](#)]
19. Keshtegar, B.; Nehdi, M.L. Machine Learning Model for Dynamical Response of Nano-Composite Pipe Conveying Fluid under Seismic Loading Machine Learning Model for Dynamical Response. *Int. J. Hydromechatronics* **2020**, *3*, 38–50. [[CrossRef](#)]
20. Ramesh Murlidhar, B.; Kumar Sinha, R.; Tonnizam Mohamad, E.; Sonkar, R.; Khorami, M. The Effects of Particle Swarm Optimisation and Genetic Algorithm on ANN Results in Predicting Pile Bearing Capacity the Effects of Particle Swarm Optimisation and Genetic Algorithm. *Int. J. Hydromechatronics* **2020**, *3*, 69–87. [[CrossRef](#)]
21. Jalal, A.; Sarif, N.; Kim, J.T.; Kim, T.S. Human Activity Recognition via Recognized Body Parts of Human Depth Silhouettes for Residents Monitoring Services at Smart Home. *Indoor Built Environ.* **2013**, *22*, 271–279. [[CrossRef](#)]
22. Jalal, A.; Khalid, N.; Kim, K. Automatic Recognition of Human Interaction via Hybrid Descriptors and Maximum Entropy Markov Model Using Depth Sensors. *Entropy* **2020**, *22*. [[CrossRef](#)]
23. Jalal, A.; Batool, M.; Kim, K. Stochastic Recognition of Physical Activity and Healthcare Using Tri-Axial Inertial Wearable Sensors. *Appl. Sci.* **2020**, *10*, 7122. [[CrossRef](#)]
24. Ozturk, O.; Orhan, U.; Wei, D.; Yedlapalli, P.; Kandemir, M.T. Cache Hierarchy-Aware Query Mapping on Emerging Multicore Architectures. *IEEE Trans. Comput.* **2017**, *66*, 403–415. [[CrossRef](#)]
25. Markatos, E.P.; Leblanc, T.J. Using Processor Affinity in Loop Scheduling on Shared-Memory Multiprocessors. In Proceedings of the Supercomputing'92: 1992 ACM/IEEE Conference on Supercomputing, Minneapolis, MN, USA, 16–20 November 1992; pp. 104–113.
26. Gochoo, M.; Akhter, I.; Jalal, A.; Kim, K. Stochastic Remote Sensing Event Classification over Adaptive Posture Estimation via Multifused Data and Deep Belief Network. *Remote. Sens.* **2021**, *13*, 912. [[CrossRef](#)]
27. Jalal, A.; Ahmed, A.; Rafique, A.A.; Kim, K. Scene Semantic Recognition Based on Modified Fuzzy C-Mean and Maximum Entropy Using Object-to-Object Relations. *IEEE Access* **2021**, *1*. [[CrossRef](#)]
28. Mahmood, M.; Jalal, A.; Kim, K. WHITE STAG Model: Wise Human Interaction Tracking and Estimation (WHITE) Using Spatio-Temporal and Angular-Geometric (STAG) Descriptors. *Multimed. Tools Appl.* **2020**, *79*, 6919–6950. [[CrossRef](#)]
29. Ahmed, A.; Jalal, A.; Kim, K. A Novel Statistical Method for Scene Classification Based on Multi-Object Categorization and Logistic Regression. *Sensors* **2020**, *20*, 3871. [[CrossRef](#)]
30. Liu, Y.; Kato, S.; Edahiro, M. Optimization of the Load Balancing Policy for Tiled Many-Core Processors. *IEEE Access* **2019**, *7*, 10176–10188. [[CrossRef](#)]
31. Murarasa, A.; Weidendorfer, J. Building Input Adaptive Parallel Applications: A Case Study of Sparse Grid Interpolation. In Proceedings of the 15th IEEE International Conference on Computational Science and Engineering, Paphos, Cyprus, 5–7 December 2012; pp. 1–8.
32. Jalal, A.; Quaid, M.A.K.; Ud Din Tahir, S.B.; Kim, K. A Study of Accelerometer and Gyroscope Measurements in Physical Life-Log Activities Detection Systems. *Sensors* **2020**, *20*, 6670. [[CrossRef](#)]

33. Jalal, A.; Akhtar, I.; Kim, K. Human Posture Estimation and Sustainable Events Classification via Pseudo-2D Stick Model and K-Ary Tree Hashing. *Sustainability* **2020**, *12*, 9814. [[CrossRef](#)]
34. Jalal, A.; Batool, M.; Kim, K. Sustainable Wearable System: Human Behavior Modeling for Life-Logging Activities Using K-Ary Tree Hashing Classifier. *Sustainability* **2020**, *12*, 324. [[CrossRef](#)]
35. Torrellas, J.; Tucker, A.; Gupta, A. Evaluating the Performance of Cache-Affinity Scheduling in Shared-Memory Multiprocessors. *J. Parallel Distrib. Comput.* **1995**, *24*, 139–151. [[CrossRef](#)]
36. Jia, G.; Li, X.; Wang, C.; Zhou, X.; Zhu, Z. Memory Affinity: Balancing Performance, Power, Thermal and Fairness for Multi-Core Systems. In Proceedings of the 2012 IEEE International Conference on Cluster Computing, 2012, Beijing, China, 24–28 September 2012; IEEE Computer Society: Piscataway, NJ, USA, 2012; pp. 605–609.
37. Barrios Hernández, C.J.; Gitler, I.; Klapp, J. (Eds.) *High Performance Computing; Communications in Computer and Information Science; Springer International Publishing: Cham, Switzerland, 2017; Volume 697, ISBN 978-3-319-57971-9.*
38. Johari, S.; Kumar, A. Algorithmic Approach for Applying Load Balancing during Task Migration in Multi-Core System. In Proceedings of the 2014 International Conference on Parallel, Distributed and Grid Computing, Solan, India, 11–13 December 2014; pp. 27–32.
39. Muneeswari, G. Agent Based Load Balancing Scheme Using Affinity Processor Scheduling for Multicore Architectures. *WSEAS Trans. Comput.* **2011**, *10*, 247–258.
40. Holmbacka, S.; Lund, W.; Lafond, S.; Lilius, J. Task Migration for Dynamic Power and Performance Characteristics on Many-Core Distributed Operating Systems. In Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2013, Belfast, UK, 27 February–1 March 2013; pp. 310–317.
41. Gantel, L.; Layouni, S.; Benkhelifa, M.E.A.; Verdier, F.; Chauvet, S. Multiprocessor Task Migration Implementation in a Reconfigurable Platform. In Proceedings of the ReConFig'09—2009 International Conference on ReConFigurable Computing and FPGAs, Cancun, Mexico, 9–11 December 2009; pp. 362–367.
42. Van Dongen, S. Prior Specification in Bayesian Statistics: Three Cautionary Tales. *J. Theor. Biol.* **2006**, *242*, 90–100. [[CrossRef](#)] [[PubMed](#)]
43. Regueira, D.; Iturriaga, S.; Nesmachnow, S. *Communication-Aware Affinity Scheduling Heuristics in Multicore Systems; Barrios Hernández, C.J., Gitler, I., Klapp, J., Eds.; Communications in Computer and Information Science; Springer International Publishing: Cham, Switzerland, 2017; Volume 697, ISBN 978-3-319-57971-9.*