



Article

Energy-Efficient Task Partitioning for Real-Time Scheduling on Multi-Core Platforms

Manal A. El Sayed ^{*}, El Sayed M. Saad, Rasha F. Aly  and Shahira M. Habashy

Department of Computer and Systems, Faculty of Engineering, Helwan University, Cairo 11111, Egypt; Alsayed_mustafa@h-eng.helwan.edu.eg (E.S.M.S.); rashaarm@yahoo.com (R.F.A.); Shahira_heikal@h-eng.helwan.edu.eg (S.M.H.)

* Correspondence: eng.manalahmed@gmail.com

Abstract: AbstractMulti-core processors have become widespread computing engines for recent embedded real-time systems. Efficient task partitioning plays a significant role in real-time computing for achieving higher performance alongside sustaining system correctness and predictability and meeting all hard deadlines. This paper deals with the problem of energy-aware static partitioning of periodic, dependent real-time tasks on a homogenous multi-core platform. Concurrent access of the tasks to shared resources by multiple tasks running on different cores induced a higher blocking time, which increases the worst-case execution time (WCET) of tasks and can cause missing the hard deadlines, consequently resulting in system failure. The proposed blocking-aware-based partitioning (BABP) algorithm aims to reduce the overall energy consumption while avoiding deadline violations. Compared to existing partitioning strategies, the proposed technique achieves more energy-saving. A series of experiments test the capabilities of the suggested algorithm compared to popular heuristics partitioning algorithms. A comparison was made between the most used bin-packing algorithms and the proposed algorithm in terms of energy consumption and system schedulability. Experimental results demonstrate that the designed algorithm outperforms the Worst Fit Decreasing (WFD), Best Fit Decreasing (BFD), and Similarity-Based Partitioning (SBP) algorithms of bin-packing algorithms, reduces the energy consumption of the overall system, and improves schedulability.

Keywords: dynamic voltage/frequency scaling; energy-aware partitioning; multi-core real-time systems; shared resources; task allocating and scheduling



Citation: El Sayed, M.A.; Saad, E.S.M.; Aly, R.F.; Habashy, S.M. Energy-Efficient Task Partitioning for Real-Time Scheduling on Multi-Core Platforms. *Computers* **2021**, *10*, 10. <https://doi.org/10.3390/computers10010010>

Received: 16 December 2020

Accepted: 30 December 2020

Published: 8 January 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

Embedded systems have become omnipresent, with the number of just mobile devices now nearly reaching the world population. Embedded systems implementations embrace, for example, home applications, pacemakers, cell phones, satellites, energy generation and distribution, industrial automation, and many other kinds of systems. The process of managing their energy consumption has become extremely challenging. Embedded systems extremely affect the layout and development restrictions of their respective surrounding systems and inversely. Some embedded systems communicate with the physical surrounding and must ensure that a certain action is carried out successfully and that it is terminated within a determined time frame. Some eminent examples of these devices are airbags in cars, medical pacemakers, and autopilots in airplanes, and they are called real-time embedded systems.

Multi-core processors are now the current architecture for recent real-time embedded systems. To achieve both efficiency and speed, CPU architectures have evolved multi-core processor units in which two or more processors have been used to perform a task. Multi-core technology provided better response times when running massive applications, improved power management, and provided faster execution times. Multi-core processors are specially designed to run tasks in parallel. Parallelism can be at two levels in multi-core

processors—one at the hardware level and another one at the software level. The proposed software has been designed to take benefit of available parallelism.

Now, several multi-core processors are used with a dynamic voltage/frequency scaling (DVFS) mechanism to save extra energy, where the voltage or frequency for every core can be set by the CPU. As a consequence, each core in the processor may have a different processing power and energy exhaustion. Numerous strategies were suggested, recently, for energy-aware real-time task scheduling on multi-core processors that enable DVFS mechanisms [1,2].

Despite dependent tasks being widespread in many real-life applications, there are few studies that have been carried out on them. When considering dependent real-time tasks, shared resources must be accessed through a mutually exclusive approach that guarantees that a task must accomplish its critical section execution before the next task tries to access the same resource. The allocating of tasks and the scheduling in multi-core embedded real-time systems have become vital problems in curtailment of energy consumption while still satisfying the needed performance. There are several scheduling algorithms suggested to completely utilize the computing resources from multiple cores to attain high efficiency. Traditionally, multi-core real-time task scheduling can be divided into global scheduling, Global Earliest-Deadline-First (G-EDF), and partitioned scheduling, Partitioned-EDF (P-EDF) [3–5]. Tasks are scheduled by one scheduler in the global scheme, and every task is permitted to move from one processor to another during the execution. The jobs are arranged in a single global queue and an individual job of a task can be preempted on a processor and continued on a different processor. Conversely, within the partitioning scheme, task migration is prohibited and all the task instances are performed on the same processor. Every processor will have a different queue prepared to schedule task jobs. Therefore, run-time performance is enhanced and tasks can only intervene on the local processor.

Partitioned scheduling protocols are widely used and extensively endorsed in their performance and usability through commercial real-time operating systems [6]. Besides, excellently studied single-processor scheduling and synchronization mechanisms could be adapted for multiprocessors with little adjustment (or without alters). Nonetheless, partitioning tasks to the processors is believed to be a bin-packing problem that is an NP-hard problem in the strong sense; finding the optimal mechanism in polynomial time is, therefore, not likely in the generic state. So, scheduling protocols should be configured with suitable partitioning algorithms to utilize the efficiency provided by multi-cores. Heuristic techniques and adequate practicability studies have been developed to discover a near-optimal partitioning for bin-packing algorithms [3,7]. Nonetheless, scheduling protocols and existing multiprocessors (multi-cores) partitioning algorithms primarily presume independent tasks while tasks typically share resources in real applications. The classic bin-packing algorithm does not consider the blocking time, and therefore, the tasks can suffer much blocking. When tasks access shared resources such as Input/Output ports and shared memories, locking protocols are used to avoid uncontrolled priority inversions [8,9] and to preserve data coherence. To comprehend the full capabilities of embedded multi-core systems, resource-aware partitioning techniques are needed.

This study presents a heuristic partitioning algorithm, the blocking-aware-based partitioning (BABP) algorithm, to assign tasks that may access the same shared resources to the same core (beginning with the task that has the longest blocking time). Therefore, the BABP algorithm makes the best of the available parallelism in these multi-core systems, as much as possible, because it guarantees that the parallel tasks—not having shared resources—are dispatched to different cores so as to execute them in parallel.

The proposed algorithm partitions a collection of real-time tasks on a non-ideal DVS processor of a multi-core architecture. According to DVFS methods, the BABP uses a Two-Speed Strategy (TSS)-based approach known as the Dual-Speed (DS) algorithm [10], which is initially used to carry out tasks at a low level of speed and then shifts to a high-level speed immediately when the tasks are blocked.

Partitioned Earliest-Deadline-First (P-EDF) [3] is used as the dynamic priority task-scheduling strategy for each processing core of a multicore system. Upon considering dependent real-time tasks, the BABP algorithm uses the Multiprocessor Stack Resource Policy (MSRP) [11] to synchronize the access of tasks to shared resources. By using MSRP, a limited blocking time is ensured for tasks when accessing the global resources, and local resources are synchronized using SRP. When using the P-EDF algorithm to schedule tasks [3], the DS algorithm computes the low level of speed and the high level of speed based on the EDF-sufficient condition of schedulability. Therefore, while energy consumption is decreased, the timing restrictions of tasks can be guaranteed. Particularly, when tasks arrive, the DS algorithm allocates the low-speed level for executing them, while at the moment the tasks are blocked, the processor speed will shift to the high-speed level. With the DS algorithm, a high-speed interval begins when the blocking starts and terminates at the blocking task deadline. The capabilities of the proposed approach were appraised by using a simulation platform named the multi-core real-time scheduling simulator (MCRTsim) [12].

The key contributions of this study are: (1) A BABP heuristic algorithm is proposed to effectively exploit the available parallelism, balance the workload in these multi-core systems, and assign tasks which can run in parallel to different cores as much as possible. For example, as shown in Figure 1, the tasks τ_1 , τ_5 , and τ_7 can be dispatched to one core and the others to another core. (2) The suggested algorithm is implemented with a simulation platform called MCRTsim. (3) An assessment of the suggested algorithm in conjunction with the blocking-agnostic bin-backing partitioning algorithm and the (SBP) algorithm, as a reference, is done. Within the framework of this study, the blocking-agnostic algorithm points to a bin-packing algorithm that does not include blocking parameters to improve the efficiency of partitioning while the schedulability check comprises blocking times. In particular, this research presumes that tasks are periodic, preemptive (only of non-critical sections), and dependent because of the synchronous access to shared resources. By using the BABP algorithm as a partitioning strategy and P-EDF as a scheduling algorithm, the simulation results indicate that the BABP algorithm achieves more energy savings than other partitioning techniques.

The remainder of this paper is arranged as set out below. Section 2 sums up the previous research on real-time systems scheduling and synchronization with a uniprocessor or a multi-core processor. Section 3 depicts the system model and problem formulation. Section 4 discusses the proposed BABP algorithm and its implementation, with the schedulability analysis. Section 5 reports on the simulation assessment and outcomes analysis. The conclusion is reported in Section 6.

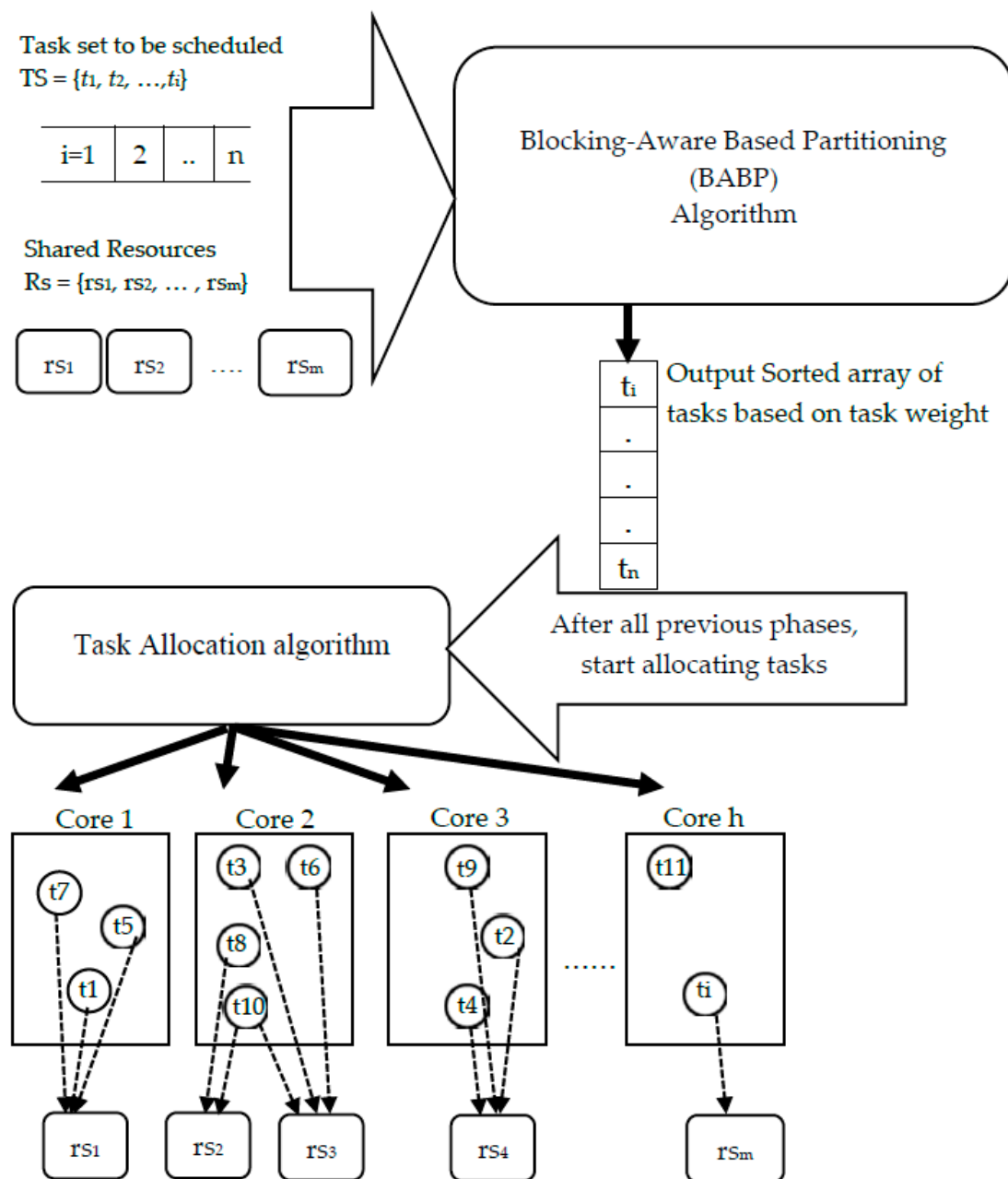


Figure 1. Block diagram of the proposed algorithm.

2. Related Work

Many studies have focused, within recent years, on energy-aware scheduling of embedded systems in real time. In uniprocessor environments, there are several research papers in the domain of energy-aware scheduling of independent real-time tasks, and an extensive survey can be found in [1]. Very little research has discussed the problem of dependent real-time tasks within the context of task synchronization [13,14]. The DVFS mechanism that works to slow the processing speed is a widely used energy-saving technique due to the convexity of the power consumption function [15–17].

The interest in multiprocessor techniques has increased as a result of the growth in multi-core architectures. The article [18] tackled the problem of energy-aware static partitioning of periodic real-time tasks on asymmetric multiprocessor (multi-core) embedded systems. It formulated the problem according to the platform-supported DVFS model and outlined optimal methods of reference partitioning for each case of the DVFS model.

The authors of [19], from the perspective of allocating workloads to cores, suggested a method for energy administration of applications in a multi-core partitioned architecture. They introduced the Energy Efficient Allocator (EEA) algorithm as an allocation method for assigning partitions to cores founded on bin-packing algorithms that consider the various frequencies at which a core can work. They also presented a variety of solutions to the problem of energy minimization. Every solution will provide an appropriate allocation of workload to cores with various levels of energy and system utilization. The EEA algorithm picks out the type of allocator (First Fit Decreasing Utilization (FFDU), WFDU, and BFDU) and the criteria (decreasing utilization (DU), increasing utilization (IU), or randomly (R)) under which partitions are chosen to minimize their frequency.

For hard real-time systems, the authors of the article [20] presented a study of energy-aware multi-core scheduling algorithms. They summed up several algorithms listed in the literature and grouped them by both homogeneous and heterogeneous multi-core processors, depending on Partitioned, Semi-Partitioned, and Global scheduling strategies. An Inter-task Affinity-aware Task Allocation (IATA) algorithm was proposed in [21] to nullify overheads in the WCET due to cache evictions. IATA collects the tasks considering their constraints, dependencies, preferences (shared resources, inter-core communication, and cache evictions) and assigns these groups to multiple cores to decrease the additive overheads in WCET.

A static mixed task scheduling (SMTS) algorithm has been proposed in [22] to solve the problem of scheduling mixed tasks that comprise of n hard real-time periodic tasks with shared resources and soft aperiodic tasks. They take into account two opposing objectives: decreasing the energy consumption and reducing aperiodic task response time. The SMTS algorithm schedules aperiodic tasks with the maximum processor speed and periodic tasks with the best speed. They have also introduced a dynamic mixed task scheduling algorithm (DMTS) capable of reclaiming dynamic slack time produced from periodic tasks and the constant bandwidth server to minimize energy consumption. Their results display that the DMTS technique outperforms the SMTS algorithm and the baseline algorithm, where DMTS decreases an average of 7.18% of energy consumption and 53.66% of response time compared with the other algorithms.

The authors of [23] suggested research on the maximum gains for volunteer computing platforms (VCPs). VCPs can be considered asymmetric multiprocessing systems (AMSS). The authors needed to pick tasks from users and assign the tasks to appropriate workers to solve the maximum benefit problem. They proposed a list-based task assignment (LTA) strategy and showed that the LTA strategy could complete the task with a deadline restriction as soon as possible. Then, based on the LTA technique, they proposed a maximum benefit scheduling (MBS) algorithm, a new task assignment algorithm aimed at optimizing VCP gains.

The authors of [24] implemented a comparison of 11 heuristics for mapping independent tasks on heterogeneous distributed computing systems. It has been shown that the relatively simple Min-min heuristic achieves minimum energy in comparison with the other strategies for the cases studied. The article [25] showed that the proposed Resource-Oriented Partitioned (ROP) scheduling with a distributed resource sharing strategy would achieve a significant speed-up factor guarantee. The authors of [26] aimed to reduce energy consumption under real-time and reliability constraints. They suggested that a formulation of an Integer Non-Linear Programming (INLP) performs task mapping by jointly addressing task allocation, assignment of task frequency, and duplication of tasks. The original INLP problem was safely converted to an analogous Mixed Integer Linear Programming (MILP) problem to provide an optimal solution. Appointing a real-time task group to the multi-core platform is a bin-packing problem that is understood to be an NP-hard problem in the powerful sense; therefore, finding the best solution in polynomial time is not pragmatic in the generic state. Given the unfavorable nature of the problem, numerous heuristics and their performance analyses were subject to various research papers, such as the First-Fit, Best-Fit, Next-Fit, and Worst-Fit methods [27,28]. A comparison was made

for homogeneous multi-core systems and periodic independent tasks between these four well-known heuristics behaviors [29].

Indeed, when the Earliest-Deadline-First scheduling technique was used, the problem had a near resemblance to bin-packing [30,31], and the results/heuristics that can be acquired in this vastly studied field show insights into partitioning-based scheduling. The suggested algorithm in [32] uses the Worst-First strategy to partition the collection of frame-based tasks (with the same period and deadline) and then scales the speed in accordance with the task characteristics in a certain instant. Although the method is represented by a rational approximation factor for optimum scheduling, some unrealistic assumptions were made by the author such as a continuous and infinitive frequency range ($s \in [0, \infty]$) and negligible in idle-state consumption. The problem of appointing a series of periodic real-time tasks in multi-core systems characterized by a single voltage island (where all processors share the same voltage and frequency) was considered in [33]. First, they examined the approximation upper bound for the classical Worst-First heuristic, and then they introduced their technique that overcomes many state-of-the-art limitations.

Resource control policies for single-processor systems are well recognized. The Priority Ceiling Protocol (PCP) [34], in particular, is one of the most attractive suggested protocols for synchronization of resource accesses. It avoids both deadlock and transitive blocking. Stack Resource Policy (SRP) [35,36] was defined as a refinement to PCP for EDF systems that strictly binds priority inversion and permits simple schedulability tests. Each task under SRP is assigned a preemption level that reflects the relative deadlines of the tasks. The shorter the deadline, the higher the preemption level. The authors of [37,38] subsequently developed multiprocessor and distributed versions of PCP. Hence, the protocol was targeted at distributed shared memory systems. There have been some versions of the Multiprocessor Priority Ceiling Protocol (MPCP) that extend PCP to multiprocessor systems and reduce the remote blocking. The authors of [39] extend the research for dynamic PCP. A dynamic priority multiprocessor version of the Priority Ceiling Protocol based upon EDF scheduling (MDPCP) was introduced in [40]. The authors of [11,41] extend SRP to the Multiprocessor Stack Resource Policy (MSRP), the first spin-lock protocol in multiprocessor real-time systems.

Partitioning-based real-time scheduling of multiprocessors finds feasibility as the primary aim. The problem occurs in two different patterns: to decrease the number of processors necessary to assure the feasibility of the task set, or, instead, to find sufficient schedulability (usually, utilization) limits given a fixed multiprocessor platform. In this research, the researchers also take into consideration the energy factor to this problem. Because generic bin-packing heuristics do not regard the blocking time caused by resource requests, they may not be efficient for task sets that have shared resources. To regard this extra blocking, the two well-known multi-core synchronization protocols, MPCP and MSRP, were presented. A partitioning heuristic adapted to the MPCP was introduced [6], a semaphore-based multiprocessor real-time locking protocol. The MSRP, spin-lock protocol [11], was proposed where tasks are busy waiting for shared resources once blocked.

The Similarity-Based Partitioning (SBP) algorithm [42] was presented. It is another partitioning heuristic for MSRP using the same methodology, which uses modern cost heuristics to more precisely classify group splits with low energy consumption. It appoints the tasks which can access the same collection of shared resources to the same core to avoid a number of blockings.

3. Research Model and Problem Formulation

3.1. Multi-Core DVFS Processor and Energy Model

Most recent processors allow variable levels for voltage and frequency, and this processor can perform dynamic voltage scaling (DVS) and its speed is proportional to the supply voltage. In the literature, DVS processors are classified as ideal and non-ideal. The ideal DVS processor will run at any speed, ranging from the lowest to the highest possible speed, whereas a non-ideal DVS processor possesses only separate speeds.

Recently, multiple DVS processors are non-ideal, whereas ideal DVS processors are for theoretical research purposes only. This study regards a multi-core platform P consisting of a set of z cores, i.e., $P = \{\text{core}_1, \text{core}_2, \dots, \text{core}_z\}$, and it supports h discrete speeds $S = \{s_1, s_2, \dots, s_h\}$, where $s_1 < s_2 < \dots < s_h$. The researchers presume that the platform P supports per-core DVFS capabilities where cores may run at different speeds at the run time.

The processor power model [43] used in this study has been greatly used in the literature [13,44]. The researchers suppose a DVFS-enabled multi-core processor is capable of operating at a variety of separate voltage levels. Commonly, the power exhaustion of a complementary metal oxide semiconductor (CMOS) system is known as dynamic and static power consumption [14]. The dynamic power dominates the total energy consumed by the processor core and the dynamic power dissipation is the most costly and time-consuming part. Therefore, this study is aiming only to detract dynamic power consumption during this study and the static power consumption is neglected [26].

The static power consumption is foremost caused by leakage currents (I_{leak}), and the static (leakage) power (P_{leak}) will be defined by:

$$P_{leak} = I_{leak} \cdot V_{dd} \quad (1)$$

The dynamic power consumption will be displayed as a convex function of the processor speed. The dynamic power consumption for CMOS circuits [45] depends on the processor operating voltage and frequency at speed S and it can be presented by:

$$P_{dynamic}(s) = C_{eff} V_{dd}^2 f \quad (2)$$

where C_{eff} is the effective switching capacitance, V_{dd} is the supply voltage, and f is the clock frequency of the processor (speed) that will be declared as:

$$f = k \cdot (V_{dd} - V_{th})^2 / V_{dd} \quad (3)$$

where k is a constant, V_{dd} is the supply voltage, and the threshold voltage V_{th} . To express the power consumption of a specified core $_i$ of processor P , the researchers use a function $PC^i(s)$ of the selected speed s . If a task keeps a processor throughout the implementation duration of $[t_1, t_2]$, then the energy exhausted by the processor throughout this period is given by:

$$\sum_{core_i \in P} \int_{t_1}^{t_2} PC^i(s(t)) dt \quad (4)$$

where $s^i(t)$ is the speed of the processor at time t .

3.2. Task and Resource Models

This study focuses on real-time systems consisting of a periodic task set with n tasks, $TS = \{t_1, t_2, \dots, t_n\}$. Each task t_i is presented by a tuple $(A_i, P_i, D_i, C_i, \text{ and } Z_i)$, where:

- The arrival time (A_i): the timing when the task is first issued.
- The period (P_i): the fixed time duration among jobs.
- The relative deadline (D_i): the maximum appropriate delay for task processing.
- The computation time (C_i): the worst-case execution time (WCET).
- The list of critical sections (Z_i) of task i .

This research regards well-formed tasks that meet the requirement $0 \leq C_i \leq D_i \leq T_i$. Each task t_i is a prototype of its instances and every instance can reach for every period T_i regularly. Let $t_{i,j}$ represent the j th instance of task t_i . Within this research, researchers are concerned about scheduling and synchronizing the dependent real-time tasks. The researchers presume these tasks are periodic, dependent (because of their access to shared resources), and preemptible (only in non-critical sections). Furthermore, they presume that a set of m shared resources (software objects, e.g., data structure, files, data objects, or shared variables)

$RS = \{rs_1, rs_2, \dots, rs_m\}$ may be accessed in a mutually exclusive method (simultaneous access is not allowable).

Researchers presume that a semaphore provides access control of shared resources to ensure mutual exclusion amongst competitive tasks. Task requests for shared resource access will happen at any moment during its implementation; a portion of code accessing a shared resource is classified as a critical section under mutual exclusion restrictions. A list that describes the critical sections of a task t_i is $Z_i = \langle z_{i,1}, z_{i,2}, \dots, z_{i,n} \rangle$, where $z_{i,j}$ is the j th critical section of t_i . This study presumes that the shared resource requests are not nested. Locks are freed in the opposite order in which they were acquired. A task t_i may request a shared resource $rs \in RS$ several times during its execution but just one job at a time will access a shared resource, i.e., binary semaphore. Real-time locking protocols assist to ensure mutual exclusion. For instance, if a task t_i asks for a shared resource rs already locked by another task, it must wait until rs is available. Besides, each shared resource rs can have a ceiling priority Ω , indicating the highest possible priority that it can have. Researchers declare u_i as the task utilization and it can be described by $u_i = \frac{C_i}{P_i}$.

The system utilization U_{tot} is equal to $\sum_{i=1}^n u_i$ and the periodic task set is scheduled by the P-EDF policy. According to P-EDF policy, priorities are appointed dynamically and are inversely proportional to the absolute deadlines of the active tasks, and the higher priority tasks are executed first.

3.3. Problem Description

Consider a workload set TS of n dependent periodic real-time tasks (dependency because of simultaneous access to shared resources) and a set RS of m shared resources. The idea is how to optimally schedule the TS and synchronize their access of RS on a multi-core processor P that supports the DVFS technique and allows h discrete speeds. The research aimed to find the optimum method of task-to-processor assignment (task partitioning) to minimize the total energy exhaustion of a real-time system. In this case, the tasks allocated to each processor can be feasibly scheduled, and the overall energy consumption of P is minimized (among all feasible task allocations). The problem of optimizing dynamic energy consumption using DVFS on a multi-core platform is an optimization problem, that is, to find feasible scheduling with minimal energy consumption [15,17]. Notice that scheduling is considered feasible if all scheduled task instances can be finished within their deadlines at the latest [33].

4. Task Scheduling and Synchronization in a Multi-Core Platform

In particular, this study uses P-EDF [3] as the scheduling algorithm and multiprocessor stack resource policy (MSRP) [11] as the synchronization protocol. By using the P-EDF scheduling algorithm, the priority-driven scheduling algorithm, tasks are partitioned offline at first among cores and are then scheduled on the allocated cores. Under MSRP, the resources are divided into two groups: local and global. Local resources are accessed only by tasks that execute on the same processor. Global resources are those which can be accessed by tasks running on different processors. There are two types of blocking: local blocking, which occurs when a task running on one core is blocked by another task running on the same core, and remote blocking, which occurs when this task is blocked by a task that is running on another core. Unlike SRP, global resources have different ceilings—one for each processor. Moreover, every processor has its own system ceiling. On processor P , tasks can only use global resources at the processor ceiling priority, that is, the highest preemption level of all the tasks on processor P . Global resources are shared across processors in a First-In-First-Out (FIFO) manner. To acquire a global resource, a task must be running at the processor ceiling which makes it non-preemptive. Whenever a task tries to access a shared resource that is already locked in the system by another task, the task performs a busy wait (called a spin-lock), and the task resumes when the shared resource is unlocked from the previously locked task.

This study use MSRP to ensure a mutual exclusion among the competing tasks from multiple cores and to maintain the data consistency of shared resources. Under the MSRP, every task has a fixed value, named preemption level λ_i of task t_i , to estimate the possible blocking in the presence of dynamic priority scheduling. Tasks with a shorter deadline will have a higher preemption level so the levels of preemption will represent the relative deadlines of the tasks. Resources are given a ceiling value during the run-time according to the maximum preemption level of the tasks accessing the resource. Whenever a task is issued, it can only preempt the currently performed task if its absolute deadline is lesser and its degree of preemption is greater than the highest ceiling of currently locked resources. The effect of this protocol is nearly identical to PCP; tasks experience only one blocking, deadlocks are avoided, and a simple formula can be obtained to compute the blocking time. The MSRP lets tasks use the local critical resources under the SRP policy. As a result, SRP saves redundant context switches by blocking earlier [11].

Schedulability Analysis of the MSRP

For a multi-core platform, researchers propose a partitioning algorithm for appointing tasks onto processors; then, the tasks will be scheduled by EDF as a scheduling algorithm and will use MSRP as a synchronization algorithm. When tasks are scheduled to be carried out on a uniprocessor [37], a group of n real-time tasks are schedulable by EDF and SRP if:

$$\left(\sum_{k=1}^i \frac{C_k}{T_k} \right) + \frac{B_i}{T_i} \leq 1, \quad \forall_i, 1 \leq i \leq n \quad (5)$$

where B_i is the worst-case blocking time of t_i . Tasks can access resources in a mutually exclusive technique, and therefore, the overheads due to blocking time must be considered whilst checking the schedulability of tasks assigned to the core. Under MSRP, if a task t_i tries to request a global resource rs , it becomes non-preemptive. If the resource rs is free, it locks the resource, but if rs is already locked by another task t_j running on a different processor, t_i performs busy wait (spinning state). The worst-case blocking time B_i^{glob} can be calculated by considering busy wait time as follows:

$$B_i^{glob} = \max \left\{ \left| z_{(t_i, rs)} \right| + spin(P_c, rs) \right\} \text{ where } (t_j \text{ is not on } P_c) \wedge (rs \text{ is global}) \quad (6)$$

where $spin(P_c, rs)$ is the upper bound of busy wait time that any task can wait on processor P_c to access a global resource rs , which can be expressed as follows:

$$spin(P_c, rs) = \sum_{\forall P_l \neq P_c} \max(|Z_{t_j, rs}|) \quad \forall t_j \text{ On } P_l \quad (7)$$

where $|Z_{t_j, rs}|$ refers to the length of any critical section of task t_j requesting to access the resource rs .

B_i^{local} is considered to be the worst-case blocking time of task t_i when accessing a local resource. By using the synchronization protocol MSRP, B_i^{local} can be calculated as follows:

$$B_i^{local} = \max \{ |Z_{t_j, rs}| \} \text{ where } (t_j \text{ is on } P_c) \wedge (rs \text{ is local}) \wedge (\lambda_i > \lambda_j) \wedge (\lambda_i \leq ceil(rs)) \quad (8)$$

where λ_i is preemption level of task t_i , and $ceil(rs)$ is the ceiling of local resource rs which is the highest preemption level of all the tasks that may access rs in core P_c .

The worst-case blocking time B_i of task t_i executing on processor P_c is calculated as follows:

$$B_i = \max(B_i^{local}, B_i^{glob}) \quad (9)$$

Based on the schedulability analysis of multiprocessor environments [11], a set of n real-time tasks on processor P_k , ordered by decreasing preemption level, is schedulable under EDF and MSRP if:

$$\sum_{k=1}^i \frac{C_k + spin_k}{T_k} + \frac{B_i}{T_i} \leq 1 \quad \forall P_k \in P, T_{Pk} = \{t_1, \dots, t_{nk}\}, \forall i = 1, \dots, n_k \quad (10)$$

The proposed algorithm aims to reduce the overall blocking overhead in the system that may exceed the schedulability of a task set.

5. Proposed Approach for Task Partitioning

This section introduces the proposed BABP algorithm for task allocation on a homogeneous uniform multi-core platform. Figure 1 presents the general idea of the proposed algorithm. The algorithm BABP can be used under partitioned RMS (Rate-Monotonic Scheduling) and partitioned EDF scheduling schemes along with MSRP. The algorithm uses the uni-core synchronization protocol SRP when dependent tasks are assigned to the same core of the processor. The BABP algorithm aims to partition the periodic real-time tasks amongst the cores to (1) decrease the overall remote blocking times of tasks due to shared resources, (2) balance a load of multiple cores, and (3) reduce inter-core communication. This usually improves the schedulability of a task set. Considering the blocking factors of tasks under MSRP, more blocking times occur by tasks with additional and extended global critical sections [11]. Algorithm 1 lists the specifics of the proposed BABP algorithm.

5.1. Blocking-Aware-Based Partitioning (BABP) Algorithm

In the initial setup, BABP calculates each task utilization $eachU_i = \frac{C_i}{P_i}$, for each task (line 5) and then calculates the *num of $z_{i,q}$* parameter which is the number of critical sections where t_i asks for resource rs_q and the *longest $z_{i,q}$* parameter which signifies the longest critical section of t_i demanding rs_q (lines 7–14). The BABP algorithm uses the previously computed parameters (in the initial step) as inputs to compute a weight for each task t_{i_w} by using (11). The proposed partitioning technique aims to minimize the blocking times, so higher weights should be given to the tasks which can lead to longer blocking times. Therefore, the calculation of the weight of task t_i depends on its utilization in addition to the number of its critical sections multiplied by the duration of its longest critical sections for all resources that it will access as follows:

$$t_{i_w} = eachU_i + \left[\frac{\sum_{rsq=1}^{RS} (num\ of\ z_{i,q} \times longest\ z_{i,q})}{P_i} \right] \quad (11)$$

The BABP algorithm generates a resource usage table (lines 16–21) (sorts the tasks in non-increasing order based on the preemption value of the task λ_i) to determine the maximum blocking time for a task t_i [37,38], denoted by local blocking time calculated by (8) or global blocking time calculated by (6), depending on the partitioning strategy. Then, it orders the tasks according to their weights in a non-increasingly order. Depending on the partitioning strategy, the calculated task weight t_{i_w} signifies the significance of the task.

The BABP algorithm picks the task pairs, starting with the first task (that has the maximum weight). Then, it calculates the proposed cost function $V_{ij,q}$ by (12), which is a function that calculates the number of critical sections and the duration of the longest critical sections for each task pair (t_i and t_j) that shares a similar resource q :

$$V_{ij,q} = (num\ of\ z_{iq} \times longest\ z_{iq}) + (num\ of\ z_{jq} \times longest\ z_{jq}) \quad (12)$$

Then, the algorithm arranges tasks in non-increasing order based on their cost function V , grouped by shared resources (lines 23–32). The output of BABP is the task set where

the tasks are sorted depending on their cost function. Figure 2 introduces the flowchart of proposed BABP algorithm.

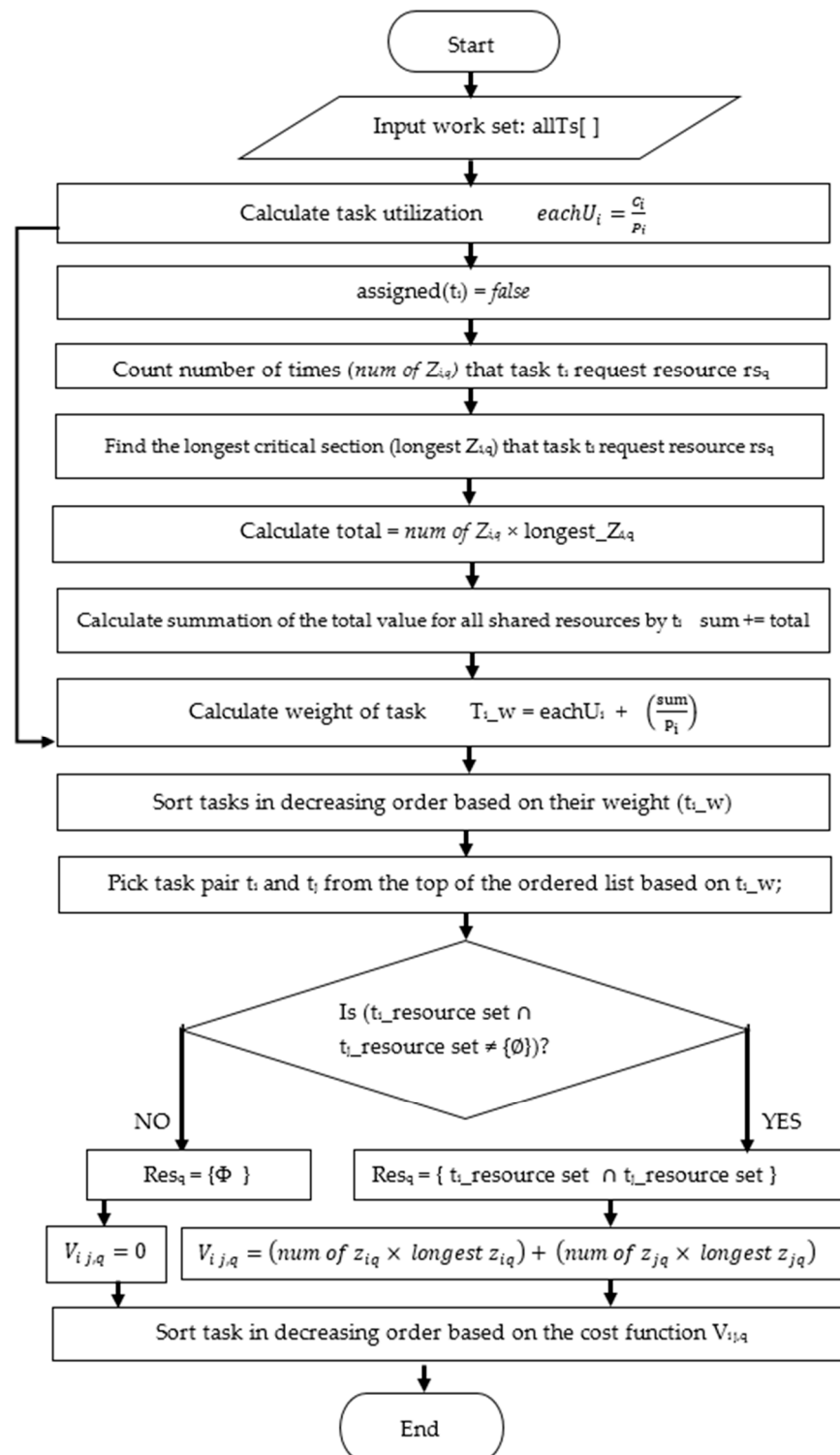


Figure 2. Blocking-Aware-Based Partitioning (BABP) algorithm flowchart.

Algorithm 1. Blocking-Aware-Based Partitioning (BABP).

Input: 1. a task set $TS = \{t_1, t_2, \dots, t_n\}$
 2. a set of resources $Rs = \{rs_1, rs_2, \dots, rs_m\}$

Initialize:

3. Generate all $Ts[]$ // array of all tasks t_i from task set TS ;
4. **foreach** $t_i \in$ all Ts
5. Calculate $eachU_i = \frac{C_i}{P_i}$; // task t_i utilization
6. $assigned(t_i) = false$;
7. **foreach** $rs_q \in t_i$ resources
8. **foreach** $z \in t_i$ -critical sections set
9. calculate num of $z_{i,q}$; // count number of times
 // the rs_q used by t_i
10. calculate $longest$ $z_{i,q}$; // longest critical section
 // of rs_q with t_i
11. $total = (num$ of $z_{i,q} \times longest$ $z_{i,q})$; // total number of
 // critical sections z_q of resource rs_q multiply by its longest critical
 // section for task t_i
12. **end for**
13. $sum + = total$; // summation of total value for all resources
 // that shared by task t_i
14. **end for**
15. Calculate $t_i-w = eachU_i + \left(\frac{sum}{P_i}\right)$; // proposed task_weight
16. Calculate $\lambda_i = \frac{1}{D_i}$; // task preemption level
17. **end for**
18. Sort tasks in descending order based on λ_i ;
19. **foreach** $t_i \in$ all Ts
20. Calculate max blocking time B_i of each task according to MSRP rules;
 // by constructing a resource usage table
21. **end for**
22. Sort tasks in descending order based on t_i-w ;
23. **for** $i = 0$ to $n-1$ // n is the number of tasks ordered by t_i-w ;
24. **for** $j = 1$ to n
25. Pick t_i and t_j from the top of the ordered list based on t_i-w ;
26. $Res_q = \{\Phi\}$;
27. **if** $(t_i$ -resource set $\cap t_j$ -resource set $\neq \{\emptyset\})$ **then**
 // check t_i and t_j have shared resources
28. $Res_q = \{t_i$ -resource set $\cap t_j$ -resource set};
29. Calculate cost fun $V_{ij,q}$ of tasks i, j for each resource in Res_q ; // by (6)
30. **end if**
31. **end for**
32. **end for**
33. Sort tasks in decreasing order based on $V_{ij,q}$, Grouping by shared resource_id;

return: List of Tasks, all Ts , sorted based on the cost function value

End

5.2. Task Allocation Algorithm

In this step, the Task Allocation algorithm picks the tasks, starting with the first task (which has the highest cost function V value), checks the task $assigned(t_i)$ value if false (line 5), and then tests the schedulability condition defined by (13). It allocates the tasks that directly or indirectly share resources to the same core based on the value of the cost function. For example, if the tasks t_i and t_j share resource R_q and tasks t_j and t_k share the same resource R_q , all three tasks will allocate to the same processor if the schedulability test is satisfied. If not, the algorithm allocates the task pairs with the maximum value of the cost function V of this shared resource to the same core at first.

If task t_i satisfies the schedulability condition for core_g and the task assignment is completed, the Task Allocation algorithm updates the $assigned(t_i)$ value to true for task t_i

and updates $core_g$ utilization considering all previous tasks' utilization allocated to that $core_g$ (lines 8–11):

$$Core_{g_{utilization}} = 1 - \left[\forall k_{i=1..n} \left(\sum_{k=1}^i \frac{C_k + spin_k}{D_k} \right) + \frac{B_i}{D_i} \right] \quad (13)$$

where B_i is the worst-case blocking time of t_k .

After the Task Allocation algorithm is finished, the task set TS is partitioned into $\{T^1, T^2, \dots, T^h\}$, and each partition is assigned to the proper $core_g$ for $g = 1$ to h . Every core takes advantage of EDF to schedule its allocated tasks. Algorithm 2 lists the specifics of the proposed Task Allocation algorithm. Figure 3 shows the flowchart of the proposed Task Allocation algorithm.

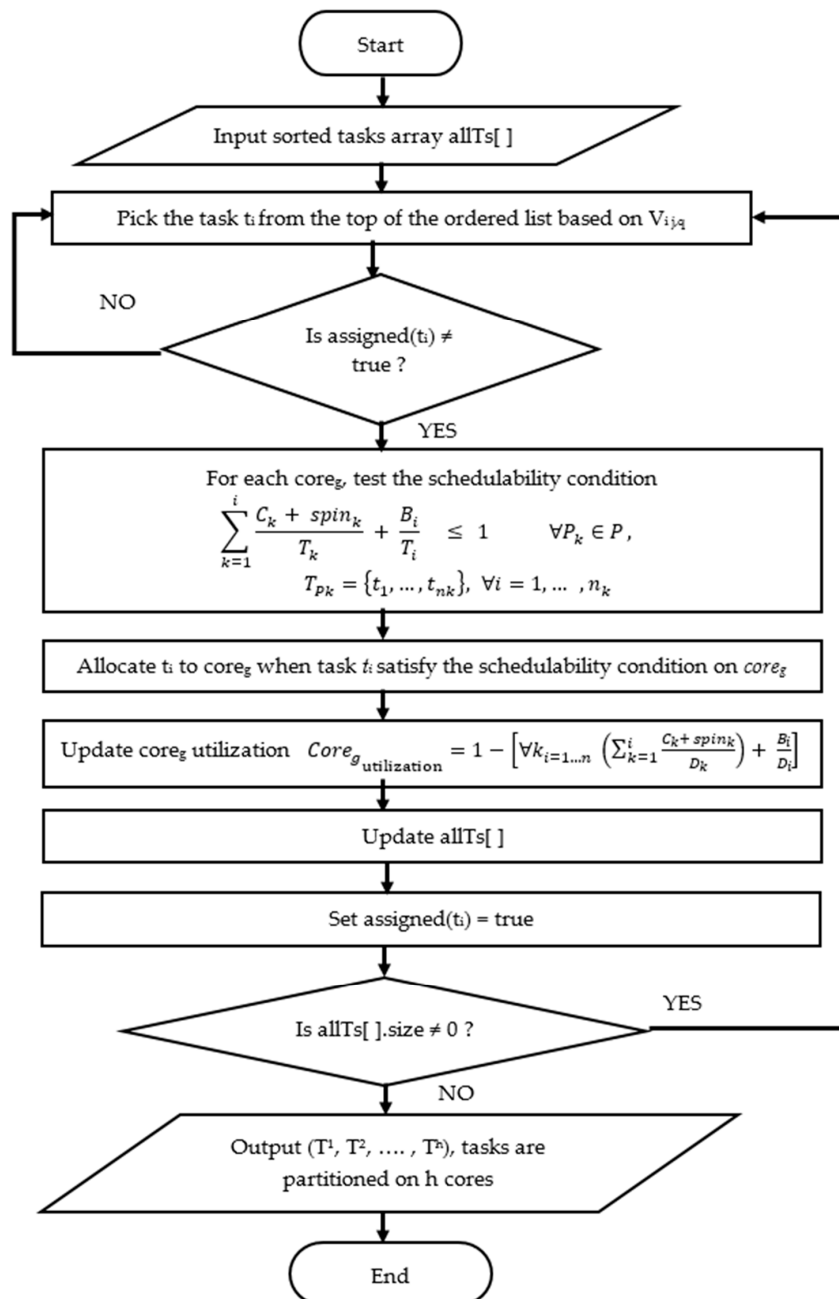


Figure 3. Task Allocation algorithm flowchart.

Algorithm 2. Task Allocation.

Input: 1. List of Tasks, allTs[], sorted based on the cost function value
 2. a multi-core processor $P = \{core_1, core_2, \dots, core_h\}$

Begin:

3. **while** (allTs.size > 0) **do**
4. Pick the task t_i from the top of the ordered list based on $V_{ij,q}$;
5. **if** assigned(t_i) \neq true **then**
6. **for** $g = 1$ **to** h // h cores
- // Test schedulability condition by (10)
7. **if** task t_i satisfy the schedulability condition on $core_g$ **then**
8. allocate t_i to $core_g$;
9. update $core_g.uti$; // using (13)
10. assigned(t_i) = true;
11. allTs.size $- = 1$;
12. break;
13. **end if**
14. **end for**
15. **end if**
16. **return** (T^1, T^2, \dots, T^h) // the output $\{T^1, T^2, \dots, T^h\}$

End

6. Experimental Evaluation and Analysis

The experimental assessment of the BABP algorithm was performed on a simulator named the multi-core real-time scheduling simulator, MCRTsim [12]. A realistic environment was used in the experiments from Marvell's XScale technology-based processor PXA270 [43] and a non-ideal DVS platform. The PXA270 processor provides six voltage frequency levels, indicated in Table 1. Through the simulator MCRTsim, the researchers set up a dual-core processor with the setting of its available processor speeds. The primary performance measure of interest in the experiments was the energy consumption of tasks, named Energy_Consum. Assuming that $s(t)$ is the speed of the processor in time t , the energy consumption Energy_Consum can be determined by $\int_0^{simTime} PC(s(t))dt$, where simTime is the time of the simulation.

Table 1. Voltage frequency levels of Marvell's XScale Pxa270 Processor [43].

Parameter	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6
Voltage (V)	1.55	1.45	1.35	1.25	1.15	0.90
Frequency (MHz)	624	520	416	312	208	104
Active Power Consumption (mW)	925	747	570	390	279	116
Idle Power Consumption (mW)	260	222	186	154	129	064

Once the tasks are assigned permanently to the processors, a speed assignment scheme is chosen to reduce the energy consumption while preserving feasibility. The Dual-Speed (DS) algorithm, a Two-Speed Strategy (TSS)-based technique, is initially used to execute tasks at a low-speed level and then switches to a high speed instantly when the tasks are blocked. When using the P-EDF algorithm to schedule tasks, DS adjusts the low-speed level and the high speed based on the EDF-sufficient schedulability condition [3].

6.1. Simulation Settings

Different workloads, randomly generated task sets, were used. Task period values were uniformly generated to obtain short tasks (10~50) ms, medium tasks (50~100) ms, and long tasks (100~500) ms. The worst-case computation time of tasks in the three groups was (1~10), (1~20), and (1~100) ms, respectively. The task period and the worst-case computation amount were picked out randomly from the respective ranges for each workload. Every task set was composed of 5~20 tasks. Within this research, the number

of shared resources was adjusted to be around 4 and 6 to ensure sufficient competition between tasks. The number of resources which a task accesses was picked at random from 1 to 4. The duration and position of the critical sections within every task were chosen randomly. Remember that the utilization bound for EDF is $\sum_{i=1}^n \frac{C_i}{P_i} \leq 1$, where n is the number of tasks. Table 2 introduces the parameters for a set of 10 tasks with a short period.

Table 2. Ten short tasks example parameter set.

i	1	2	3	4	5	6	7	8	9	10
A_i	0	0	1	3	0	1	0	2	2	0
P_i	30	27	43	45	49	40	48	50	47	39
C_i	2	1	3	6	4	3	4	7	1	1
U_i	0.067	0.037	0.069	0.133	0.082	0.075	0.083	0.14	0.021	0.026
$Z_{i,q}$	$Z_{1,3}$ $Z_{1,4}$	$Z_{2,5}$ $Z_{2,4}$	$Z_{3,2}$ $Z_{3,1}$	$Z_{4,2}$	$Z_{5,1}$ $Z_{5,2}$ $Z_{5,3}$	$Z_{6,4}$ $Z_{6,2}$	-	-	$Z_{9,5}$ $Z_{9,1}$ $Z_{9,3}$	-

6.2. Experimental Results and Discussion

When the randomly generated task sets were partitioned by BABP, SBP [42], and classic bin-packing heuristics—BFD and WFD [30,31]—they were scheduled and synchronized by P-EDF and MSRP in MCRTsim. These generated feasible dynamic priority task sets were assessed under various values of the utilization factor. The BABP algorithm was evaluated under the partitioned EDF scheduling scheme along with the MSRP multi-core shared resources synchronization protocol. Regarding energy consumption, the results of BABP are compared with the following heuristic algorithms' results: SBP, BFD, and WFD. Figure 4 shows a graphical representation of the simulation results, such that the scheduling results of tasks' executions and the resource usages can be observed on different cores.

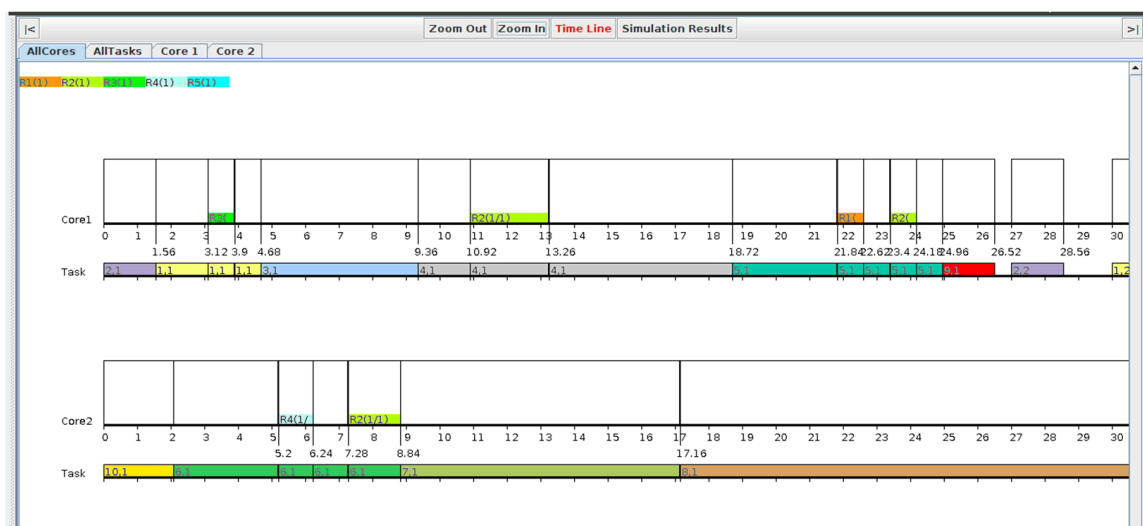


Figure 4. An output of the Visualized Execution Tracer for 10 short-period tasks.

Table 3 indicates the simulation results, energy consumption, for scheduling a set of 10 tasks with a short period. The results show that the blocking time caused by global shared resources is minimized significantly by allocating tasks using BABP as compared to other mentioned task allocation techniques. When a bin-packing algorithm allocates a task to a bin, it usually allocates the task in a bin that fits it better, and it does not consider the unallocated objects that will be allocated after the current object. The BABP approach

gathers all tasks with a higher value of the cost function $V_{i,j,q}$ and allocates them on the same core, which reduces the remote blocking time of the task due to shared resources, minimizes the inter-core communication, and exploits the parallelism of the multi-core architectures efficiently. These time-saving factors result in reducing the total processor busy time, which is consumed to schedule additional tasks.

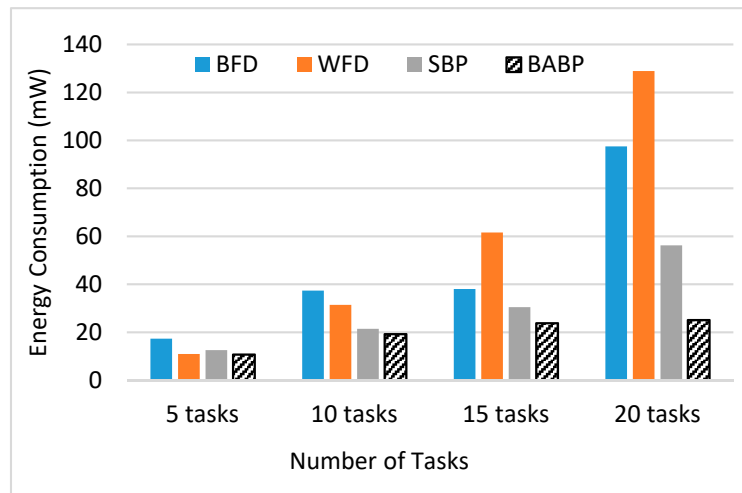
Table 3. The energy consumption for scheduling 10 example short tasks.

Partitioning Algorithm	BFD	WFD	SBP	BABP
Energy Consumption (mW)	37.38	31.45	21.43	19.19

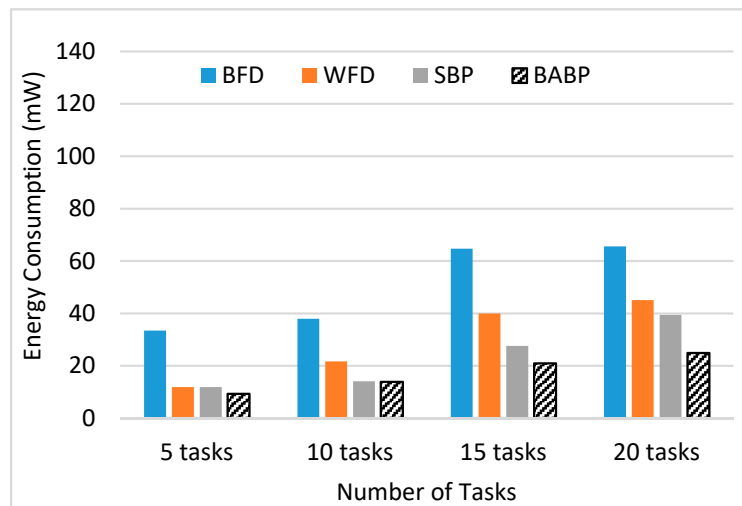
Considering the effect of variation in the number of tasks per task set, a simulation was performed by doubling, tripling, and quadrupling the number of tasks. Consider Figure 5 which plots the overall energy consumption of the system (the vertical axis) that the algorithms could schedule successfully versus the number of tasks 5, 10, 15, and 20 of the task set (the horizontal axis). Figure 5 indicates the energy consumption varied according to the number of tasks and the period of the task. Part (a) shows task sets with a short period (10~50) ms. Part (b) shows task sets with a medium period (50~100) ms. Part (c) shows task sets with a long period (100~500) ms. The results display that the increase in the number of tasks will increase energy consumption in all situations and the task set with fewer tasks will outperform the task set with more tasks. As a result of the competition among tasks for resources, the number of blockings and their time will increase as the number of tasks grows.

Figure 5 illustrates that the BABP algorithm performs significantly better than all other compared partitioning algorithms. The BABP algorithm can minimize the amount of remote blocking by partitioning tasks based on their resource usage likeness (directly or indirectly shared) and the longest blocking time. The BABP algorithm allocates the tasks that directly or indirectly share resources onto the same processor based on the importance of the cost function. Hence, BABP succeeds in minimizing the length of remote blocking. However, the SBP algorithm performs better than the blocking-agnostic algorithm in some situations only. The results indicate that in some cases, the increase in the task period contributes to more energy savings. They illustrate that task sets with a long period, even with a heavy workload, will profit from the proposed BABP algorithm to improve the system performance and save more energy.

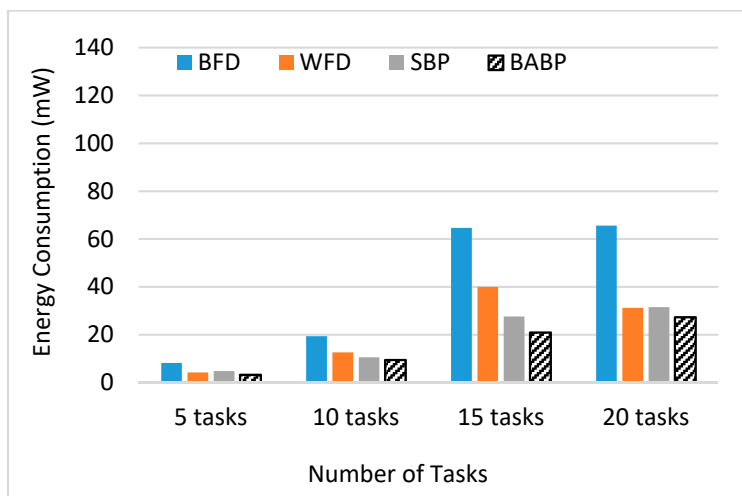
Considering the effect of variation in total utilization of tasks, numerous task sets have been randomly generated, with total utilization ranging from 0.4 to 1.0. Figure 6 plots the overall energy consumption of the system (the vertical axis) that the algorithms could schedule successfully versus the total utilization of the task set (the horizontal axis). It demonstrates the comparison of the results of the BABP, SBP, BFD, and WFD algorithms. These outcomes indicate that the variation in energy consumption for a given utilization is increased with increasing the utilization of tasks in some situations, as it is observed that BABP outperforms all other techniques in all situations.



(a) Task sets with a short period.

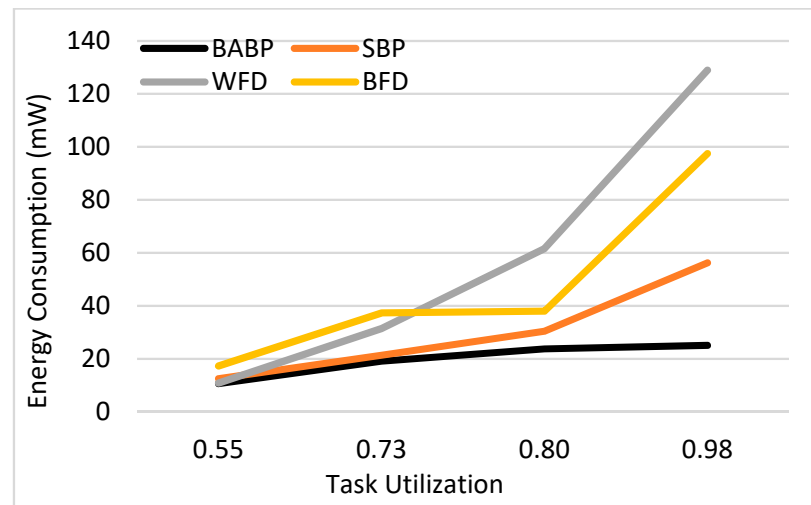


(b) Task sets with a medium period.

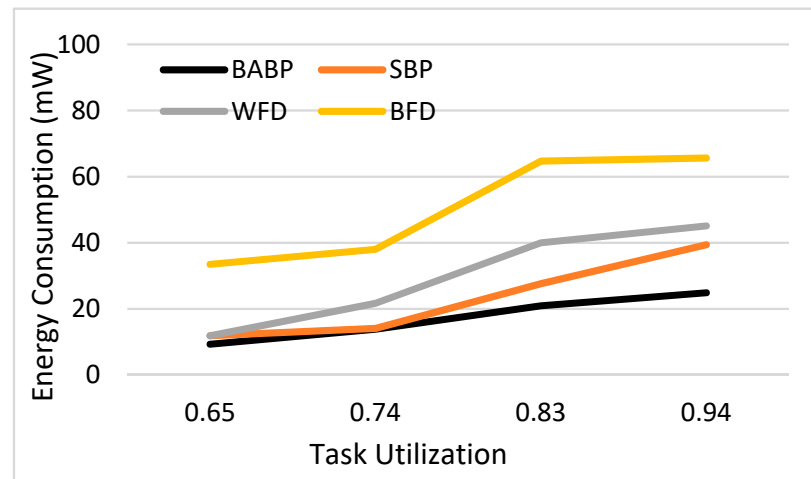


(c) Task sets a with long period.

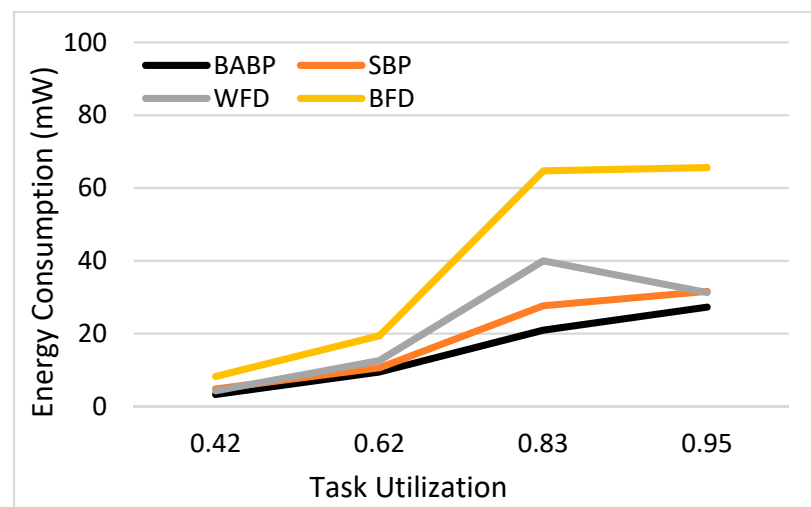
Figure 5. Energy consumption results in variation with a number of tasks and their period.



(a) Tasks sets with a short period.



(b) Task sets with a medium period.



(c) Task sets with a long period.

Figure 6. Energy consumption results with variation in tasks' utilization.

7. Conclusions

Obtaining better performance and meeting the hard deadlines of real-time tasks is a very critical problem. This research examines the problem of curtailment energy consumption for a dependent periodic real-time task set that share resources. The proposed BABP algorithm assigns a task set to the processor of a single-chip multiprocessor (multi-core) with shared memory. The proposed technique's goal is to minimize task blocking times by allocating tasks that share resources directly or indirectly to appropriate processors, beginning with tasks with the maximum estimated blocking time. Generally, this increases the schedulability of a task group and may result in fewer processors needed compared to blocking-agnostic bin-packing strategies. BABP supports the parallelism between tasks which do not have shared resources efficiently. The method can exploit the parallelism of the multi-core architectures efficiently. In this method, the tasks in one application can run on a different processing core in parallel.

Because so many systems use dynamic priority scheduling protocols in practice, the researchers implemented the proposed algorithm under MSRP, a standard synchronization protocol for multiprocessors (multi-cores) that operates under the dynamic priority scheduling algorithm P-EDF. The proposed technique performs notably better than the other conventional task allocation algorithms BFD, WFD, and SBP. The results reflect that the proposed algorithm reduces the globally shared resources and inter-core communication. The proposed partitioning algorithm will greatly reduce blocking times, improve overall system performance, and reduce energy consumption. The proposed approach abilities were evaluated using the MCRTsim simulator. For future work, the researchers will also concentrate on run-time task partitioning strategies for global and semi-partitioned schemes with various evolving synchronization protocols.

Author Contributions: Conceptualization of the study, methodology, algorithm development, optimization, original software, and drafting of the manuscript, M.A.E.S.; revising and approving the version of the manuscript to be published, E.S.M.S.; revising the manuscript, R.F.A.; revising the manuscript critically for important intellectual content and approved the version of the manuscript to be published, S.M.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: I choose to exclude this statement because the study did not require ethical approval.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author and will be made publicly available at a later time.

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

References

1. Chen, J.-J.; Kuo, C.-F. Energy-efficient scheduling for real-time systems on dynamic voltage scheduling (DVS) platforms. In Proceedings of the IEEE-RTCSA, Daegu, Korea, 21–24 August 2007; pp. 28–38.
2. Zhuravlev, S.; Saez, J.C.; Blagodurov, S.; Fedorova, A.; Prieto, M. Survey of energy-cognizant scheduling techniques. *IEEE Trans. Parallel Distrib. Syst.* **2012**, *24*, 1447–1464. [[CrossRef](#)]
3. Baker, T.P. *A Comparison of Global and Partitioned EDF Schedulability Test for Multiprocessors*; Technical Report. TR-051101; Department of Computer Science, Florida State University: Tallahassee, FL, USA, 2005.
4. Devi, U.C. *Soft Real-Time Scheduling on Multiprocessors*. Ph.D. Thesis, Department Computer Science, North Carolina University, Chapel Hill, NC, USA, 2006.
5. Shinde, V.; Biday, S.C. Comparison of Real Time Task Scheduling Algorithms. *IJCA* **2017**, *158*, 37–41. [[CrossRef](#)]
6. Lakshmanan, K.; de Niz, D.; Rajkumar, R. Coordinated task scheduling, allocation and synchronization on multiprocessors. In Proceedings of the IEEE-RTSS, Washington, DC, USA, 1–4 December 2009; pp. 469–478.
7. Carpenter, J.; Funk, S.; Holman, P.; Srinivasan, A.; Anderson, J.; Baruah, S. A categorization of real-time multiprocessor scheduling problems and algorithms. In *Handbook on Scheduling Algorithms, Methods, and Models*; Chapman Hall/CRC: Boca, FL, USA, 2004.
8. Brandenburg, B.B.; Anderson, J.H. Optimality results for multiprocessor real-time locking. In Proceedings of the IEEE-RTSS, San Diego, CA, USA, 30 November–3 December 2010; pp. 49–60.

9. Yang, M.; Wieder, A.; Brandenburg, B.B. Global real-time semaphore protocols: A survey, unified analysis, and comparison. In Proceedings of the IEEE-RTSS, San Antonio, TX, USA, 1–4 December 2015; pp. 1–12.
10. Zhang, F.; Chanson, S.T. Blocking-aware processor voltage scheduling for real-time tasks. *ACM Trans. Embed. Comput. Syst.* **2004**, *3*, 307–335. [[CrossRef](#)]
11. Gai, P.; Lipari, G.; Natale, M.D. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In Proceedings of the IEEE-RTSS, London, UK, 2–6 December 2001; pp. 73–83.
12. Wu, J.; Huang, Y.C. MCRTsim: A Simulation Tool for Multi-Core Real-Time Systems. In Proceedings of the ICASI, Sapporo, Japan, 13–17 May 2017; pp. 461–464.
13. Jejurikar, R.; Gupta, R. Energy aware task scheduling with task synchronization for embedded real time systems. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2006**, *25*, 1024–1037. [[CrossRef](#)]
14. Wu, J. Energy-efficient scheduling of real-time tasks with shared resources. *FGCS* **2015**, *56*, 179–191. [[CrossRef](#)]
15. Choi, K. Dynamic voltage and frequency scaling for energy-efficient system design. Ph.D. Thesis, Department Electrical Engineering, Southern California University, Los Angeles, CA, USA, 2005.
16. Teodorescu, R.; Torrellas, J. Variation-aware application scheduling and power management for chip multiprocessors. *Int. Symp. Comput. Archit.* **2008**, *36*, 363–374. [[CrossRef](#)]
17. Yao, F.; Demers, A.; Shenker, S. A scheduling model for reduced cpu energy. In Proceedings of the IEEE Annual Symposium on FOCS, Milwaukee, WI, USA, 23–25 October 1995; pp. 374–382.
18. Elewi, A.; Shalan, M.; Awadalla, M.; Saad, E.M. Energy-Efficient Task Allocation Techniques for Asymmetric Multiprocessor Embedded Systems. *ACM Trans. Embed. Comput. Syst.* **2014**, *13*, 71. [[CrossRef](#)]
19. Guasque, A.; Balbastre, P.; Crespo, A.; Coronel, J. Energy efficient partition allocation in partitioned systems. *IFAC* **2018**, *51*, 82–87. [[CrossRef](#)]
20. Sheikh, S.Z.; Pasha, M.A. Energy-Efficient Multicore Scheduling for Hard Real-Time Systems: A Survey. *ACM Trans. Embed. Comput. Syst.* **2018**, *17*, 94. [[CrossRef](#)]
21. 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 IBCAST, Islamabad, Pakistan, 14–18 January 2019; pp. 492–499.
22. Zhang, Y.; Li, H. Energy aware mixed tasks scheduling in real-time systems. *Sustain. Comput. Inform. Syst.* **2019**, *23*, 38–48. [[CrossRef](#)]
23. Xu, L.; Qiao, J.; Lin, S.; Wang, X. Research on the Task Assignment Problem with Maximum Benefits in Volunteer Computing Platforms. *Symmetry* **2020**, *12*, 862. [[CrossRef](#)]
24. Braun, T.D.; Siegel, H.J.; Beck, N.; Boloni, L.L.; Maheswaran, M.; Reuther, A.I.; Robertson, J.P.; Theys, M.D.; Yao, B.; Hensgen, D.; et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Parallel Distrib. Comput.* **2001**, *61*, 810–837. [[CrossRef](#)]
25. Yang, M.; Huang, W.H.; Chen, J.J. Resource-Oriented Partitioned Scheduling in Multiprocessor Systems with Shared Resources. *IEEE Trans. Comput.* **2018**, *68*, 882–898. [[CrossRef](#)]
26. Cui, M.; Mo, L.; Kritikakou, A.; Casseau, E. *Energy-Aware Partial-Duplication Task Mapping Under Real-Time and Reliability Constraints*; Springer: Cham, Switzerland, 2020; pp. 213–227. [[CrossRef](#)]
27. Burchard, A.; Liebeherr, J.; Oh, Y.; Son, S.H. New strategies for Assigning Real-Time Tasks to Multiprocessor Systems. *IEEE Trans. Comput.* **1995**, *44*, 1429–1442. [[CrossRef](#)]
28. Lopez, J.M.; Diaz, M.G.J.L.; Garcia, D.F. Worst-Case Utilization Bound for EDF Scheduling on Real-Time Multiprocessor Systems. In Proceedings of the ECRTS, Stockholm, Sweden, 19–21 June 2000; pp. 25–33.
29. Aydin, H.; Yang, Q. Energy-aware partitioning for multiprocessor real-time systems. In Proceedings of the IPDPS, Nice, France, 22–26 April 2003.
30. Coffman, E.G., Jr.; Garey, M.R.; Johnson, D.S. Approximation Algorithms for Bin Packing: A Survey. In *Approximation Algorithms for NP-Hard Problems*; PWS Publishing: Boston, MA, USA, 1997; Chapter 2; pp. 46–93. Available online: <https://www.labri.fr/perso/eyraud/pmwiki/uploads/Main/BinPackingSurvey.pdf> (accessed on 16 December 2019).
31. Pardalos, P.M.; Du, D.-Z.; Graham, R.L. *Handbook of Combinatorial Optimization, Bin Packing Approximation Algorithms: Survey and Classification*, 2nd ed.; Springer: New York, NY, USA; Heidelberg, Germany; Dordrecht, The Netherlands; London, UK, 2013; ISBN 978-1-4419-7997-1.
32. Yang, C.-Y.; Chen, J.-J.; Kuo, T.-W. An approximation algorithm for energy-efficient scheduling on a chip multiprocessor. In Proceedings of the DATE, Munich, Germany, 7–11 March 2005; pp. 468–473.
33. Kandhalu, A.; Kim, J.; Lakshmanan, K.; Rajkumar, R.R. Energy aware partitioned fixed-priority scheduling for chip multiprocessors. In Proceedings of the IEEE-RTCSA, Toyama, Japan, 28–31 August 2011; pp. 93–102.
34. Sha, L.; Rajkumar, R.; Lehoczkzy, J.P. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Trans. Comput.* **1990**, *39*, 1175–1185. [[CrossRef](#)]
35. Baker, T.P. A Stack-Based Resource Allocation Policy for Real-time Processes. In Proceedings of the IEEE-RTSS, Lake Buena Vista, FL, USA, 2–4 December 1990; pp. 191–200.
36. Baker, T.P. Stack-Based Scheduling of Realtime Processes. *Real Time Syst.* **1991**, *3*, 67–99. [[CrossRef](#)]
37. Rajkumar, R. Real-time synchronization protocols for shared memory multiprocessors. In Proceedings of the IEEE-DCS, Paris, France, 28 May–1 June 1990; pp. 116–123.

38. Rajkumar, R. Synchronization in Multiple Processor Systems. In *Synchronization in Real-Time Systems: A Priority Inheritance Approach*, 1st ed.; Kluwer Academic Publishers: Amsterdam, The Netherlands, 1991; Chapter 3; pp. 61–177.
39. Chen, M.-I.; Lin, K.-J. Dynamic Priority Ceilings: A concurrency Control Protocol for Real-Time Systems. *Real Time Syst.* **1990**, *2*, 325–346. [[CrossRef](#)]
40. Chen, C.-M.; Tripathi, S.K.; Blackmore, A. A Resource Synchronization Protocol for Multiprocessor Real-Time Systems. In Proceedings of the International Conference on Parallel processing, Raleigh, NC, USA, 15–19 August 1994; Volume 3, pp. 159–162. [[CrossRef](#)]
41. Gai, P.; Natale, M.D.; Ferrari, G.L.A. A comparison of MPCP and MSRP when sharing resources in the Janus multiple-processor on a chip platform. In Proceedings of the IEEE-RTAS, Toronto, ON, Canada, 27–30 May 2003. [[CrossRef](#)]
42. Wu, J.; Hong, X.-J. Energy-Efficient Task Scheduling and Synchronization for Multicore Real-Time Systems. In Proceedings of the IEEE-Big Data Security on Cloud, Beijing, China, 26–28 May 2017; pp. 179–184.
43. *Marvell PXA270 Processor, Electrical, Mechanical, Thermal Specification*; no. MV-S104690-00, Rev. D; Marvell Semiconductor, Inc.: Santa Clara, CA, USA, April 2009.
44. Zhuo, J.; Chakrabarti, C. Energy-efficient dynamic task scheduling algorithms for DVS systems. *ACM Trans. Embed. Comput. Syst.* **2008**, *7*, 17. [[CrossRef](#)]
45. Burd, T.D.; Brodersen, R.W. Energy efficient CMOS microprocessor design. In Proceedings of the Hawaii International Conference on System Sciences, Wailea, HI, USA, 3–6 January 1995; Volume 1, pp. 288–297.