Project Description

1 Career Development Plan: Research

With advances in technology, modern embedded systems interact more closely with human beings and the physical world to form complex cyber-physical systems (e.g., smart cars [169]), where additional functionalities (e.g., collision avoidance) and higher data quality requirements create ever-increasing performance demands [94, 105, 130, 144]. Although the performance of embedded processors has been significantly improved, such increasing performance demands of embedded applications make it infeasible for a single processor and multiple processing units have been widely adopted in modern embedded systems [141, 172]. Moreover, in addition to various timing constraints, energy efficiency and high reliability are also of utmost importance, especially for safety-critical embedded systems with limited energy budget. Therefore, considering these multidimensional and often conflicting requirements on performance, energy efficiency and system reliability, the design of real-time embedded systems poses serious technical challenges.

The emergence of multicore processors, where a single chip integrates multiple processing cores with various shared resources (such as L2 cache) [74, 126], quickly leads us into a new multicore computing era. As powerful computing engines, multicore processors provide great opportunities and could alleviate the design difficulties for real-time embedded systems, especially considering the parallel nature of modern embedded applications (e.g., HDTV, military radars and encoding of multi-channel video/audio [105, 130, 141]). However, the problem of how to effectively exploit multicore processors in real-time embedded systems is not trivial and viable solutions involve many aspects of computer science research, such as architecture (cache policies), operating systems (scheduling) and compiler techniques (program analysis and parallelization). Note that, the performance of multicore based embedded systems relies largely on what applications/tasks are executed where, when and how. Although real-time scheduling has been studied for decades with well developed scheduling theories for both uniprocessor and multiprocessor real-time systems, the scheduling for multicore based real-time systems just caught the attention of researchers recently with many new challenges, especially considering the tightly-coupled on-chip resources (e.g., L2 cache) and the parallel nature of the problem. As the main theme of this proposed research, the scheduling issues in multicore based real-time embedded systems will be studied systematically.

In the last decade, energy-aware computing has moved to the forefront of research in computer science and engineering. Despite the significant progress reported, where many energy management schemes have been studied for various real-time system models and scheduling policies, the problem remains a grand challenge [77]. In particular, not much work has been done on exploiting multicore processors for energy savings, in real-time settings [29, 147]. With the continued scaling of CMOS technologies and adjustment of design margins for higher performance, it is expected that, not only the systems that are traditionally operated in electronics-hostile environments (such as those in outer space), practically all digital systems will be much more vulnerable to transient faults [14, 166]. Moreover, recent studies show that, dynamic voltage and frequency scaling (DVFS), which is an efficient and widely exploited energy saving technique, has an alarmingly negative effect on system reliability due to increased transient fault rates in semiconductor circuits at lower supply voltages [56, 145, 192]. Therefore, the solutions that consider only one dimension in isolation of others have limited applicability and a comprehensive scheduling framework becomes a necessity.

The basic tenet of this proposed research plan is to study and develop an integrated scheduling framework for multicore based real-time embedded systems, which will incorporate the flexible performance/power characteristics [11, 61] and inherent redundancy in multicore processors [74, 126]. Such an initiative is crucial as multicore processors will be the computing engines for...
the next-generation real-time embedded systems, where the demands for high performance, energy efficiency and high system reliability are ever-increasing. With his strong research background in real-time scheduling [134, 195, 200], energy management [15, 133, 117, 171, 184, 190, 191, 196] and fault tolerance [180, 182, 185, 188, 189, 192, 193, 194, 197, 198, 199], the PI will tackle the problem of designing and developing the proposed scheduling framework in two steps.

First and the most important, to guarantee various hard and soft timing constraints of different (e.g., periodic, sporadic and aperiodic) real-time tasks in multicore based embedded systems, the novel component-based execution and resource models will be studied systematically. Based on these models, both offline and online multicore-aware real-time scheduling algorithms will be developed. Then, considering different system requirements on energy efficiency and reliability, efficient energy-aware and flexible reliability-aware real-time scheduling algorithms will be developed, which will take into account all the power consuming components (e.g., memory and I/O devices) in embedded systems and the inherent redundancy in multicore processors.

1.1 Multicore-Aware System Models and Real-Time Scheduling Algorithms

The fundamental requirement for real-time embedded systems is to ensure that applications/tasks complete their executions in the desired timely manner. For different real-time tasks to be executed on systems with a single or multiple processing units, the scheduling problem of how to guarantee various hard and/or soft timing constraints has been studied extensively in the last few decades [149]. As the well-known scheduling algorithms, rate monotonic scheduling (RMS) and earliest deadline first (EDF) have been shown to be optimal in uni-processor periodic real-time systems, for static and dynamic priority assignments, respectively [111]. However, neither of these algorithms is optimal for real-time systems with multiple processing units [60].

Traditionally, two major approaches have been studied for the scheduling problem in multiprocessor real-time systems: partitioned and global scheduling [58, 60]. In partitioned scheduling, each task is assigned to a specific processor and processors can only execute tasks that are assigned to them. With such partitioning, different well-developed uniprocessor scheduling algorithms (e.g., RMS and EDF) can be applied to the subset of tasks on each individual processor. Global scheduling, on the other hand, puts all tasks into a shared single queue and all processors fetch the next ready task for execution from the global queue. That is, tasks have more flexibility to migrate and execute on different processors depending on their run-time behaviours. Recently, as a hierarchical approach, cluster scheduling has been proposed, which partitions tasks among different groups of processors (i.e., clusters) and global scheduling is adopted for the tasks within each cluster of processors [19, 34, 150]. Note that, cluster scheduling is a general approach, which will reduce to partitioned scheduling when there is only one processor in each cluster. For the case of a single cluster consisting of all processors, it will resemble global scheduling.

In multicore processors, such as Intel Core2 Quad processors [81] and AMD Quad-Core processors [4], the processing cores on a chip generally share (various levels of) on-chip caches. Such shared cache architecture can significantly mitigate the task migration overhead, which has been the traditional concern against global scheduling. Therefore, there is a reviving interest in global scheduling and some interesting results have been reported in recent years [8, 10, 19, 20, 21, 40, 41, 59, 73, 90, 91, 92, 132]. Despite the recent progress, many problems in parallel real-time scheduling are still open, especially for systems with multicore processors. Due to the implicit sharing of the tightly-coupled system resources (e.g., on-chip caches [74, 100]), Cazorla et al. have identified the problem of performance non-predictability for applications being executed on systems with multicore processors [46, 47]. Although several techniques have been studied to address the performance interference problem (such as cache quota management [83, 135], hardware priority
enforcement mechanisms [172] and performance isolation schemes [69]), such performance non-predictability has become the major barrier deterring the exploitation of multicore processors in real-time embedded systems.

Therefore, to guarantee various hard and soft timing constraints of different real-time tasks in multicore based embedded systems, the PI will first investigate novel execution and resource models and explore efficient offline and online multicore-aware real-time scheduling algorithms, which will form the foundation of the proposed scheduling framework. In particular, the PI plans to undertake research in the following specific directions:

- **Component-based execution and resource models for multicore embedded systems**

  In traditional real-time embedded systems, processing units are generally modeled as single-dimension resources. That is, a processor is allocated to real-time tasks in a time multiplexing manner and a task has exclusive access to the processor during its allocated time intervals [111]. Moreover, in classical real-time scheduling theory, it is assumed that a real-time task is a single-threaded process that can only occupy one processor at any given time (that is, tasks cannot be executed in parallel on more than one processor) [60]. Based on these conventional execution and resource models, timeliness of real-time tasks is assumed to be guaranteed through worst case analysis [65, 55, 168] and appropriate resource allocation/reservation mechanisms [136, 142, 143].

  For embedded systems with multicore processors, when tasks are scheduled during the same time interval, they will run in parallel physically on different processing cores and compete for the shared on-chip resources (e.g., L2 cache [100, 156]). As the key step to ensure timeliness in embedded systems, worst case analysis needs to take such execution interferences among tasks due to accessing the shared resources into consideration. However, for general embedded systems with many tasks and processing cores, such worst case analysis could be extremely difficult considering different schedules and execution orders of tasks. Note that, there is a dilemma between the worst case analysis and the schedule of tasks due to their circular dependencies: accurate worst case analysis relies on the exact schedule of concurrently running real-time tasks, whereas the exact schedule of tasks will depend on the results of worst case analysis (i.e., WCETs of tasks) as well as the adopted scheduling policy and algorithm. Therefore, to facilitate the worst case analysis and simplify the scheduling steps in multicore based real-time embedded systems, the development of new execution and resource models becomes a necessity.

**Proposed Work:** Component oriented design [21] and compositional real-time scheduling have been studied recently to address the increasing complexity in embedded systems [120, 151, 152]. The key idea is to organize a subset of (component) tasks in an embedded system as a compositional task, where the timing constraints and resource requirements of the component tasks will be guaranteed if the aggregated resource requirement of the compositional task can be ensured through appropriate resource allocation and scheduling algorithms. Inspired by these work, in this project, the PI will study the component-based execution and resource models for multicore based real-time embedded systems. Here, to separate the concerns of worst case analysis from those of scheduling algorithms, a few real-time (component) tasks are grouped into a compositional super task. For a given mapping and schedule of the component tasks on the desired number of processing cores, the worst case analysis can be carried out (for example, by extending the approach in [173]) for the super task, which will have a single unified timing and resource requirement. In the scheduling phase, a super task will compete for system resources (i.e., one or more processing cores as in the gang-scheduling [89]) simultaneously for its component tasks. Accordingly, processing cores sharing tightly-coupled resources (e.g., L2 cache) are bounded together as a virtual processor, which will be allocated collectively to a super task.
Figure 1: Composition of virtual processors and super tasks.

For example, Figure 1a shows an 8-core processor with two cores sharing L2 cache. Here, four virtual processors (\(VP_1\), \(VP_2\), \(VP_3\) and \(VP_4\); represented by the dotted rectangles) with each one having two cores are used to model the processor. Note that, considering the parallel nature of their functionalities, real-time applications/tasks could be implemented in a multi-threaded fashion and run on more than one processing core for better performance [105]. Suppose that there are four tasks \(T_1\), \(T_2\), \(T_3\) and \(T_4\), which have the same period \(P\) and form a subset of tasks for an application. Here, \(T_1\) has two threads and other tasks are single-thread processes. If the application will be executed on a system with an 8-core processor as shown in Figure 1a, to restrain execution interferences within a super task, it is desired to construct super tasks that require two cores simultaneously. Figure 1b shows two super tasks (\(ST_1\) and \(ST_2\)) for different compositions (i.e., mapping and scheduling) of these four tasks. We can see that the mapping and scheduling of the component tasks, which can utilize the rough estimation of tasks’ computation requirements, are very important for the resource requirement of a super task. For a given schedule of its component tasks, the size (i.e., WCET) of a super task can be obtained through accurate worst case analysis. Therefore, a super task can be represented by its parallelism (i.e., number of cores required simultaneously), its WCET and period (which can be the common period of its component tasks). For the super tasks in Figure 1b, either of them can be scheduled on a virtual processor with two cores as shown in Figure 1a (other virtual processors could execute other super tasks of the application). If a scheduling algorithm can ensure the resource requirement of a super task, the timing constraints of its component tasks will be guaranteed.

From the above example, it is easy to see that different compositions for super tasks and virtual processors have direct effects on the schedulability of a set of real-time tasks in an application and the system utilization. Intuitively, to confine the execution interferences due to shared resources and to facilitate the worst case analysis, the processing cores that share resources should belong to one virtual processor (as shown in Figure 1a). For the composition of super tasks, to simplify the worst case analysis and scheduling process, it is necessary for the component tasks within a super task to have the same periodicity as shown in the above example. For tasks with different periods, the idea of sub-tasks can be explored to first divide the tasks according to the greatest common divisor (GCD) of their periods. Then the sub-tasks can be composed into super tasks. In addition, the composition should also consider the number of threads in each component task, the number of cores in a virtual processor and the utilization of component tasks in a super task. The idea is to ensure that a super task only needs one virtual processor at any given time, which will simplify the scheduling algorithms. Note that, super tasks with low utilization may share a virtual processor in a way similar to periodic tasks sharing a conventional processor.

Extending these preliminary results, the PI plans to further investigate different composition approaches (for instance, to address tasks’ precedence constraints) for various (periodic, sporadic and aperiodic) real-time tasks on multicore based embedded systems.
Efficient multicore-aware real-time scheduling algorithms for periodic tasks

For the scheduling problem, the PI will first focus on periodic tasks, which are the classical and important real-time tasks. Note that, scheduling algorithms for systems with multicore processors will be inherently parallel. As an important research area, parallel real-time scheduling has been studied for decades with many results having been reported [5, 60, 148, 95, 108, 137, 162], including the ones from recent studies [8, 10, 19, 20, 21, 27, 40, 41, 59, 73, 90, 91, 92, 132]. However, it is still an evolving field and many problems remain open, especially for the real-time embedded systems with multicore processors.

In the pioneering work that addressed the scheduling problem for real-time embedded systems with multicore processors, Jain et al. have listed and investigated two important problems: task co-scheduling and resource sharing of co-scheduled tasks, for soft real-time multimedia applications [86]. More recently, for different architectures of multicore processors, Anderson et al. have also investigated several scheduling algorithms, again, for soft real-time systems [7, 8, 40, 41, 42]. However, considering the tightly-coupled shared resources (e.g., on-chip L2 cache) in multicore processors and performance interferences among tasks due to resource sharing, it is still a great challenge on how to efficiently utilize the available resources and provide hard real-time guarantees in multicore based embedded systems.

Assuming that the scheduler can be invoked at any time instance (i.e., continuous time domain) and processors can be allocated in arbitrarily small share, a few optimal parallel real-time scheduling algorithms have been studied, which can achieve full system utilization and guarantee all the timing constraints of tasks [44, 51, 70]. However, due to the limitation in real systems, processors are generally allocated to tasks in discrete time quanta (e.g., 10ms in Linux). Based on the discrete time model, the proportional fair (Pfair) scheduling algorithm has been proposed for periodic real-time tasks, which is also optimal in terms of achieving full system utilization [25]. The basic idea of Pfair is to enforce proportional progress (i.e., fairness) for each task at every time unit, which actually puts a more strict requirement for the problem. That is, any Pfair schedule for a set of periodic real-time tasks will ensure that all task instances can complete their executions before the deadlines.

More specifically, assuming that a periodic task $T_i$ has WCET $c_i$ and period $p_i$ (which are both integers), the weight of $T_i$ is defined as $W_i = \frac{c_i}{p_i}$. The allocation error of $T_i$ at time $t$ is defined as $\delta_i(t) = X_i(t) - t \cdot w_i$, where $X_i(t)$ is the number of time units allocated to $T_i$ from time 0 to time $t$. To obtain the Pfair schedule, the algorithm ensures that $|\delta_i(t)| < 1$ for every task $T_i$ at any time $t$ [25]. Extending this idea, several variations of Pfair scheduling algorithm have been studied [8, 9, 26, 119]. However, by making scheduling decisions at every time unit, these algorithms could lead to quite high scheduling overhead, which is undesirable for embedded systems.

**Proposed Work:** Observing that a real-time task can only miss its deadline at its period boundary, as the preliminary work, the PI has studied an optimal boundary fair (Bfair) scheduling algorithm for multiprocessor real-time systems [195]. Extending the ideas of Pfair scheduling, Bfair makes scheduling decisions and ensures fairness for the tasks only at their period boundaries. Specifically, define the period boundary time points for a set of periodic real-time tasks as $B = \{b_0, \ldots, b_f\}$, where $b_0 = 0$, $b_j < b_{j+1}$ ($j = 0, \ldots, f - 1$), $\forall j, \exists (i, k)$, $b_j = k \cdot p_i$ and $b_f = \text{LCM}$. Here, due to the periodicity of the problem, the schedule is considered only up to LCM (least common multiple) of all tasks' periods. That is, at time $b_j$, Bfair allocates processors to tasks for the time units between $b_j$ and $b_{j+1}$. Bfair ensures that, at any period boundary $b_j \in B$, $|\delta_i(b_j)| = |X_i(b_j) - b_j \cdot w_i| < 1$ for every task $T_i$. That is, the allocation error for any task $T_i$ at any boundary time $b_j$ is within one time unit and the resulted schedule is boundary fair [195].
For illustration, consider an example task set with 6 periodic tasks running on two processors: $T_1 = (2, 5)$, $T_2 = (3, 15)$, $T_3 = (3, 15)$, $T_4 = (2, 6)$, $T_5 = (20, 30)$ and $T_6 = (6, 30)$, where $\sum_{i=1}^{6} w_i = 2$ and $LCM = 30$. Figure 2a shows the schedule generated from the Bfair algorithm [195]. For comparison, the schedule obtained from the Pfair algorithm [25] is shown in Figure 2b. The numbers in the rectangles of the schedules denote that the corresponding tasks will be executed on the specific processor during those time units. The dotted lines in the figures are the period boundaries of the tasks. From these schedules, we can see that there are only 10 scheduling points for Bfair, while the number of scheduling points for Pfair is 30 (the number of time units) within one LCM. The preliminary study shows that, compared to Pfair scheduling algorithms, Bfair can reduce 75% scheduling points [195]. For tasks with more regular periodicity, our recent results show that such reduction can be up to 94% [200].

Moreover, by allocating multiple time units together, the execution of tasks can be aggregated together under the Bfair scheduling algorithm and the number of context switches can be significantly reduced compared to that of the Pfair schedule. For the above example, there are 45 context switches in the Bfair schedule and the number is 52 for the Pfair schedule within one LCM. Our recent study shows that, compared to that of Pfair, Bfair can reduce the number of context switches by up to 82% [200]. For the same reasoning, the number of task migrations under Bfair can also be reduced by up to 85% [200]. Such reduction in context switches and task migrations is very important to reduce the run-time overhead of real-time embedded systems.

Following this line of research, the PI plans to further investigate efficient parallel real-time scheduling algorithms for periodic tasks with less overhead, which is specially valuable for embedded systems. For instance, to reduce the migration overhead of tasks, it is desired to keep some fixed tasks on each processor while limiting the number of tasks that can migrate among processors. Although several studies have been reported recently based on the concept of portion task (or task split) [90, 91, 92], the solutions cannot achieve full system utilization. It has been proved that Pfair schedules with such mapping constraints do exist [112, 119]. Based on the maximum flow graph, the PI has developed an offline algorithm to generate a Pfair schedule for periodic tasks with mapping constraints. Note that, for the tasks with fixed mapping to the same processor, the time units allocated to them in the Pfair schedule can be rearranged follow-
ing the EDF policy to reduce the number of context switches. For the online efficient scheduling algorithm, which is still open, the PI has investigated several approaches to extend Pfair/Bfair algorithms based on task aggregation/split and will continue to work in this direction.

Considering the fact that a few processing cores in a multicore processor may share on-chip resources (e.g., L2 cache) and even power supply (as discussed later), it is desirable to group them together as a cluster (or virtual processor) for easy management. As another research venue, the PI will extend the cluster scheduling [19, 34, 150] where Bfair will be adopted within each cluster of tasks and processing cores. Intuitively, larger clusters with more processing cores and tasks can improve system utilization while incurring more scheduling points (thus higher scheduling overhead) and, vice versa. Following the similar reasonings for the utilization bound of partitioned scheduling [5, 114], our preliminary study shows that, for a system with $n_k$ clusters and each cluster having $k$ processing cores, the worst case utilization bound is $[134]:$

$$UB_{k,n_k} = \frac{k}{k+1} (k \cdot n_k + 1)$$  

(1)

Here, if $k = 1$, the bound will effectively reduce to be the one for partitioned scheduling [5, 114]. For the case of a single cluster ($n_k = 1$) that consists of all $k$ processing cores, the bound will be $k$, which is actually the maximum utilization for the conventional Bfair algorithm [195]. The PI plans to extend the Bfair algorithm and the cluster scheduling approach to incorporate the novel component-based execution and resource models for multicore based real-time embedded systems. Furthermore, as another important category, non-preemptive scheduling will also be explored to address the complication of cache effects in multicore systems. Here, as the basic scheduling element, a super task (with all its component tasks) will run to complete whenever it is scheduled. The corresponding schedulability analysis will be investigated in detail.

- **Flexible multicore-aware real-time scheduling algorithms for mixed task sets**

Different from periodic tasks, aperiodic real-time tasks can arrive at any time and, for sporadic tasks, the consecutive arrivals need to be separated by a minimal interval [113]. Many techniques (such as, server based approaches [2, 157, 158], slack stealing [102] and CASH queue [39]) have been studied to handle aperiodic and sporadic real-time tasks with various soft or hard timing constraints in uniprocessor systems [110] and some of the techniques have been extended for multiprocessor systems [22, 23, 28, 33, 93, 128]. However, the problem of how to effectively support aperiodic and sporadic tasks and to improve their response times and/or the acceptance ratio in systems with multicore processors remains open.

**Proposed Work:** Extending the scheduling algorithms to be developed for periodic tasks in the above section, the PI will investigate different mechanisms to efficiently support sporadic and aperiodic real-time tasks. The straightforward approach would conserve a few dedicated processing cores to handle aperiodic and sporadic tasks. Here, the number of dedicated cores is critical for the system performance and needs be carefully evaluated. Moreover, although this approach avoids execution interferences with periodic tasks at run-time, it excludes the possibility of exploring the vast amount of dynamic slack that would be available online to improve the performance of aperiodic and sporadic tasks. Following the ideas of server-based schemes [22, 28, 93], in the second approach, certain capacity on each core or virtual processor can be reserved for aperiodic and sporadic tasks while the remaining capacity is utilized for serving periodic tasks. Furthermore, slack reclamation mechanisms (such as CASH queue [128] and wrapper tasks [187]) will be explored to enhance the performance of aperiodic and sporadic tasks at run-time. The feasibility of these approaches will be analyzed and proved theoretically and empirical evaluations will be performed in detail.
1.2 Energy- and Reliability-Aware Multicore Real-Time Scheduling Algorithms

Research for real-time embedded systems in the last decade has also been marked by a growing interest in power/energy management as most embedded systems have limited energy budget. The basic idea for power management is to run systems at low performance and thus low power consumption states whenever possible. Considering that processors consume significant amount of power in embedded systems and dynamic power dissipation dominates [36], most of the previous work has targeted managing the power consumption of processors [16, 72, 131, 191, 167, 174]. However, with the increased static/leakage power due to scaled feature sizes and increased integration levels [32, 63, 123] as well as other power consuming components (such as main memory [57, 101], secondary memory [53, 80, 106] and I/O devices [160, 161]), system-wide energy management is gaining increasing importance [15, 50, 52, 68, 87, 88].

An important design principle in multicore processors is power efficiency. Justified by the convexity of the power dissipation function with processing frequency and supply voltage, for the same performance, instead of being executed on a complex super-fast processor, an application can be executed in parallel on multiple simple lower speed processing cores to save energy. Although many static and dynamic power management schemes have been proposed for uniprocessor real-time embedded systems [16, 17, 82, 121, 129, 131, 167, 174, 72, 175, 179] as well as multiprocessor systems [3, 18, 24, 191, 196], only a few studies have exploited the power efficiency of multicore processors recently, in the setting of real-time embedded systems [29, 147].

As the traditional concerns, reliability and fault tolerance have been major factors in computer system design and there is an extensive literature on fault-tolerant real-time systems [30, 38, 99, 107, 109, 125, 127]. During the operation of a system, faults may occur due to various factors, which have to be carefully handled in a timely manner, especially for safety-critical real-time embedded systems. In general, permanent faults, which are caused by hardware defects and/or wear-out, are tolerated by providing space/hardware redundancy (e.g., triple modular redundancy (TMR)). In contrast, transient faults, which are normally caused by temporary interferences (e.g., electrical noises and cosmic ray radiations), can be recovered by re-executing the faulty computation with rollback recovery techniques [97]. As transient faults occur much more frequently than permanent faults [45, 84, 85], especially with the scaled technology sizes and reduced design margins [14, 166], various techniques have been proposed to detect and tolerate transient faults at different levels (e.g., self-correction circuit design [14, 118], architecture level redundancy [13, 122, 159, 165] and compiler-directed software duplication [48, 78, 139]).

Note that, the various forms (e.g., hardware/space and temporal) of redundancy needed in the commonly used fault tolerance techniques have implications on energy consumption. For instance, both DVFS-based energy management schemes and rollback recovery techniques inherently rely on (and potentially compete for) the active use of system slack. Moreover, recent studies showed that, although an effective and widely deployed technique for energy savings, DVFS has an alarming negative effect on system reliability [56, 145, 192]. Therefore, for safety-critical real-time embedded systems with limited energy budget, energy-efficient fault tolerance techniques will be of great interests and have caught researchers’ attention recently [64, 116, 163, 177, 178], including the PI’s previous work [180, 182, 185, 187, 188, 189, 192, 193, 194, 197, 198, 199].

Proposed Work: As the second part of the proposed scheduling framework, the PI plans to investigate: a). various energy-aware real-time scheduling algorithms that will incorporate the flexible performance/power characteristics of multicore processors; and b). flexible reliability-aware real-time scheduling algorithms that will exploit the inherent redundancy of multicore processors and address energy efficiency simultaneously. Specifically, the PI will explore the following directions:
Energy-Aware Scheduling Algorithms for Multicore Embedded Systems: To effectively investigate power management schemes for embedded systems with multicore processors, it is necessary to develop an **accurate but easy-to-use system power model** that can incorporate all the power consuming components (such as memory [57, 101] and I/O devices [160, 161]). As the number of cores on a chip continues to increase (with tens [164] or even hundreds of cores [76]), various organizations of the cores have been studied for high performance and/or energy efficiency [41, 61, 75]. Exploiting the **voltage island** technique [54, 62, 79, 96, 103], the PI has studied **block-partitioned** configurations for multicore processors, where the cores are partitioned into blocks with the ones in the same block sharing the same power supply voltage and thus having the same frequency [133]. Note that, due to energy efficiency and/or flexibility consideration, the number of cores in each block can be different (as shown in Figure 1a).

Following the same principle of a widely used simple system-level power model for single processor systems [15, 192], the PI has derived an easy-to-use power model for systems with block-partitioned multicore processors, where the system power consumption $P$ can be represented as [133]:

$$P = P_s + \sum_{i=1}^{n_b} x_i \cdot \left( P_{ind}^i + P_{dd}^i \right) = P_s + \sum_{i=1}^{n_b} x_i \cdot \left( P_{ind}^i + \sum_{j=1}^{n_c} y_{i,j} \cdot C_{cf} \cdot f_i^m \right) \quad (2)$$

Here, $n_b$ denotes the number of blocks in a multicore processor and $n_c$ presents the number of processing cores in the i’s block ($1 \leq i \leq n_b$). The system power $P$ has two major parts: **static power** ($P_s$, a constant) and **active power**. The active power for each block is further divided into two components: **frequency-independent active power** ($P_{ind}^i$) and **frequency-dependent active power** ($P_{dd}^i$), which depends on the common processing frequency ($f_i$) for the cores in that block. Note that, with advanced technologies, one idle core can be put into sleep state in only a couple of cycles [4, 81]. Thus, each core can be in either the **active** state (when it is actively executing some workload; denoted by $y_{i,j} = 1$) or the **sleep** state (with $y_{i,j} = 0$). When all cores in a block are in the sleep state, the block can be powered off (denoted by $x_i = 0$) to further save its $P_{ind}^i$. Moreover, $C_{cf}$ is the switch capacitance and $m$ is a system dependent constant [15, 36, 52].

Despite its simplicity, the above model captures the essential components of power consumption in embedded systems with multicore processors and supports both **on/off** and **DVFS** techniques for effective system-level energy management. The PI will extend the model to consider new architectures (e.g., heterogeneous cores [42]) and study its effects on energy management in detail.

Incorporating the above system power model and the component-based execution and resource models, the PI will investigate **static power-aware real-time scheduling algorithms** for multicore-based embedded systems. Note that, there are two important factors that affect the static schedule and the amount of available (global and local) static slack: the composition of the tasks and processing cores as well as the scheduling policy adopted. Therefore, for different architectures of multicore processors and a given scheduling policy, the PI will first investigate the effects of various compositions of super tasks (from both independent and dependent tasks) and virtual processors (where cores belong to the same or different blocks) on the energy savings. Although it is optimal for uniprocessor systems to scale down the processing of all tasks uniformly within the interval considered [16, 131], the PI has previously discovered that the single speed setting is sub-optimal for parallel systems due to different degrees of parallelism within the schedule of an application [117]. As another avenue of this research, the PI will investigate energy-efficient static mapping algorithms for the component-based super tasks and virtual processors. Considering the inherent parallelism of super tasks and the number of cores within each virtual processor, a super task may require more than one virtual processors at the same time, or more than one super tasks...
can share a virtual processor at a given time. Such facts should be taken into consideration in the static mapping schemes to achieve the best energy efficiency, in the real-time settings.

As real-time tasks only take a small fraction of their worst case execution times [66], the dynamic slack obtained at run-time provides additional opportunities for energy savings. Based on global scheduling, the PI has previously studied a slack sharing mechanism, which enables processors to share dynamic slack with each other for better energy savings [190, 191]. Following this direction, the PI will investigate **dynamic power-aware real-time scheduling algorithms** with slack sharing for embedded systems modeled by component-based super tasks and virtual processors. Note that, different from the systems with conventional single-core processors, a processing core may not be able to adjust its running frequency freely since the cores in multicore processors may share a common supply voltage and have the same frequency. Moreover, the slack from early completion of a super task (when all its component tasks finish their executions) will appear on all processing cores occupied by that super task. Depending on how the super tasks and virtual processors are composed, such slack may need to be shared and/or reclaimed simultaneously.

Note that, instead of being used for energy management, spare capacity and dynamic slack can also be utilized for aperiodic real-time tasks to improve their response times or to increase the acceptance ratio. Considering embedded systems with mixed task set, as another research direction, the PI plans to study the tradeoff between energy efficiency and the performance of aperiodic tasks for different policies of slack usage, for embedded systems with multicore processors.

**Reliability-Aware Scheduling Algorithms for Multicore Embedded Systems:** Exploiting the inherent space redundancy in multicore processors, many architecture-level fault detection/recovery mechanisms have been studied [71, 98, 104, 122, 138, 181], where the central idea is to execute multiple copies of the same thread at the hardware level to provide transparent fault tolerance. However, all these techniques require some form of hardware modifications (i.e., adding different hardware comparators) and fault detection/recovery are enforced for all applications without considering their various reliability requirements. Moreover, with shared data and program segments among threads, **thread-level duplication (TLD)** can only detect/recover from transient faults in the logical/functional units as on-chip caches are generally not protected [170].

To take the faults occurred in memory sub-systems into consideration, the PI has proposed the concept of **process-level duplication (PLD)** [186]. Considering the available process management facilities in operating systems, PLD can be implemented in software [154] and exploits the commodity multicore processors. With the novel component-based task model, it will be interesting to investigate how to compose super tasks from the duplicated processes (i.e., tasks) for flexible reliability management. For instance, the duplicated processes can be grouped into the same super task for the ease of scheduling algorithms considering their same timing properties. However, to avoid correlated faults among the duplications, it may be necessary to put them into different super tasks and/or schedule them on different processing cores. Following this direction, the PI will study various compositions of duplicated processes/tasks for efficient fault detection and recovery, while providing better flexibility for energy management.

Moreover, by exploiting the inherent space redundancy and implicit temporal redundancy in real-time embedded systems with multicore processors, the PI will investigate flexible energy efficient reliability-aware real-time scheduling algorithms, in the following specific directions:

- **Energy-efficient fault tolerance scheduling with space and temporal redundancies**

In real-time systems, space redundancy is widely exploited in safety-critical systems (e.g., avionics systems) to tolerate both transient and permanent faults. However, it also implies double (or even triple) acquisition and operation costs (e.g., power consumption) with concurrent execution
for all the replicas. In contrast, with temporal redundancy, backup replicas are executed only when the primary execution is faulty (normally on the same system). As real-time systems are generally over-designed to accommodate the peak load conditions, excessive amount of slack time is expected at run-time [66], which could be exploited as temporal redundancy to enhance system reliability. For a real-time application being executed on a triple modular redundant (TMR) system, the PI has previously studied the optimal frequency/voltage settings for an optimistic-TMR (OTMR) scheme [194], where one processing unit runs at lower frequency for energy savings provided that it can catch up and finish the computation before the application deadline in case the results from other two units do not have an agreement [64]. For multicore based highly reliable embedded systems, the PI plans to extend the idea to the general concept of optimistic modular redundancy (OMR) for detecting and tolerating more than one fault.

Moreover, the PI plans to investigate energy-efficient fault tolerance schemes for various real-time applications. For instance, for a real-time application with a set of dependent tasks, to increase system reliability without violating timing constraints, space redundancy can be used for duplicated executions of the tasks on the critical path [146], while the recovery of other tasks may explore temporal redundancy for energy efficiency. For systems where excessive amount of dynamic slack is available at run time, temporal redundancy may be exploited to dynamically substitute space redundancy and save more energy. Furthermore, considering that different forms of real-time tasks (e.g., original code vs. the one with fault detection and/or recovery) can be generated with the help of advanced compilers [48, 139, 140], it will be crucial for a scheduler to choose the appropriate form of tasks and determine where (i.e., on which core) to execute them, after carefully evaluating the tradeoff between performance, reliability and energy efficiency.

- Parallel reliability-aware scheduling algorithms with negative effects of DVFS

Figure 3: Sequential and Parallel Reliability-Aware Power Management.

Most of the existing energy-efficient fault tolerance techniques did not consider the negative effects of DVFS on system reliability [64, 116, 163, 177, 178]. However, as the system supply voltage (and processing frequency) is scaled down through DVFS techniques for energy savings, the average arrival rate of transient faults (assuming a Poisson distribution [37, 176]) could increase exponentially [56, 145, 192]. Taking such effects into consideration, the PI has studied one reliability-aware power management (RAPM) scheme [183]. Instead of utilizing all the available slack to slow down the execution of a real-time task for energy savings, the RAPM scheme reserves part of the slack for one recovery task while exploiting the remaining slack to save energy. For example, as shown in Figure 3a, suppose that the next ready task $T_k$ has its worst-case execution time (WCET) as $c_k = 2$ and there are 3 units of slack available in the system at time $t$. Without
considering reliability, the *ordinary* greedy power management scheme would use all the slack and reduce the execution of task $T_k$ at frequency $0.4 \cdot f_{\text{max}}$ (where $f_{\text{max}}$ is the maximum available frequency) exploiting the DVFS technique as shown in Figure 3b, which could lead to drastically reduced system reliability [183]. Assuming that the recovery ($b_k$) of task $T_k$ takes the form of re-execution, the RAPM scheme will reserve 2 units of slack for the recovery. The remaining 1 unit of slack can be used to slow down the execution of task $T_k$ to frequency $\frac{2}{3} f_{\text{max}}$ for energy savings (shown in Figure 3c). With the assumption that the recovery task (i.e., re-execution) is invoked at the highest frequency ($f_{\text{max}}$), it has been proved that the system reliability can be preserved regardless of the degree of negative affects of DVFS on transient fault rates [183].

The RAPM scheme has been extended to consider various real-time task models and scheduling policies [180, 185, 187, 188, 189, 198, 199], all of them have focused on uniprocessor real-time embedded systems. However, with the assumption that transient faults are recovered through backward recovery with re-execution, the existing RAPM framework has inherent limitations. For instance, if the WCET of the task $T_k$ in the above example is $c_k = 3$ instead (as shown in Figure 3d), there will be no enough slack for $T_k$’s recovery on the same processor.

Considering the availability of multiple processing cores, the PI strongly believes that multicore processors will provide new opportunities for designing flexible and efficient parallel RAPM schemes. For the above example, the recovery $b_k$ can be scheduled on a separate processing core while allowing the primary task $T_k$ to exploit the slack for scaling down its execution to save energy (as shown in Figure 3e). However, to maximize the energy savings, the amount of slack reclaimed by task $T_k$ (and thus its scaled frequency) has to be carefully evaluated to reduce the overlapped execution with its recovery (which runs at $f_{\text{max}}$) as shown in Figure 3f. Moreover, for highly reliable systems, idle processing cores can be used to enhance system reliability [49]. Based on the efficient scheduling algorithms to be developed in the first part, the PI will study flexible parallel RAPM schemes, which aim to address various constraints (e.g., timing, energy efficiency and system reliability) simultaneously and systematically.

### 1.3 Research Methodology and Validation Techniques

From an engineering perspective, the proposed research will involve development/implementation efforts for a set of scheduling algorithms to effectively utilize the special features of multicore processors in real-time embedded systems for performance, energy efficiency and reliability. At every stage, such efforts will be guided/supported by rigid theoretical analysis on the correctness and efficiency (in terms of computational complexity) of the algorithms and techniques under consideration, which has utmost importance as in other areas of Computer Science. For problems that are (or will be shown to be) intractable (most likely because of the parallel nature of multicore-based systems), efficient heuristics with provable performance bounds will be investigated.

The validation of the theoretical research findings and the evaluation/justification of the proposed schemes will be performed (in part) through experiments with real-world applications and related benchmarks [105, 124, 153] on physical platforms equipped with multicore processors. For this purpose, a platform with embedded multicore processors (such as AMD quad-core Opteron [4], ARM11 MPCore [12], Broadcom BCM1455 [35] or Intel quad-core Xeon L5408 [81]) will be acquired. As a Linux testbed for multiprocessor scheduling in real-time systems, LITMUS$^{RT}$ has been designed and implemented as a soft real-time extension to Linux kernel with the focus on multiprocessor real-time scheduling and synchronization [6, 43]. The PI plans to port LITMUS$^{RT}$ to the platform to be acquired. Then, the proposed execution/resource models and multicore-aware scheduling algorithms/schemes will be implemented as kernel functions and evaluated accordingly.
Moreover, the system power consumption will be measured with a data acquisition (DAQ) system to examine the effectiveness of the proposed schemes on energy efficiency. To accommodate the necessary equipments and corresponding research activities of graduate research assistants, a Real-Time Embedded Systems Laboratory will be established within the Department of Computer Science at the University of Texas at San Antonio during the first year of this project.

Although the actual implementation and experiments in real systems are undeniably important, the design and development of accurate and fast simulators are also essential, especially considering the extremely time consuming reliability-related experiments and the high cost for accelerated reliability tests. Moreover, as the research on the support for multicore processors is still in the early stage, only a few simulators for multicore-based systems are publicly available (e.g., GEMS [115] and SimFlex [155]), most of which have very limited support for evaluating real-time systems. As another important effort in this proposed research, a simulator for multicore-based real-time embedded systems will be developed. It is expected that this simulator will provide complete and flexible support to evaluate various real-time scheduling algorithms on different system configurations for their performance, energy efficiency and impacts on system reliability. Once completed, the simulator will be available to the research community, which will facilitate the promotion of “low-power reliable computing” in the real-time embedded system research community, accomplishing another important goal of this proposed project.

2 Career Development Plan: Education

While research leads us to make breakthroughs in science and develop advanced techniques for solving real life problems, teaching is an important means of passing the accumulated knowledge to the next generations and inspiring them for pursuits in science. Such synergy between research and teaching has attracted me to pursue an academic career. In what follows, the PI will focus on elaborating the educational activities enabled by this proposed project within the University of Texas at San Antonio (UTSA), an officially recognized minority institution.

Positioned in areas where Hispanics form the dominant majority, about 58% of UTSA students were from minorities in Fall 2008. Recently, UTSA has been striving to make the transition to be a top-tier research institution. The proposed effort will be instrumental in providing more research facilities and opportunities for training both undergraduate and graduate students researchers, most of whom will come from underrepresented groups. The PI will also invest special efforts and time to increase the participation of under-represented students to the various phases of this research project. For example, in Fall 2007, the PI recruited Mr. Salvador Rodriguez to work on “LEGO robots for Data Collection in Wireless Sensor Networks” after his summer study on embedded systems. As a member of UTSA team, Mr. Rodriguez also participated in the Ciber-Mouse programming contest in conjunction with RTSS’07 held in Tucson, Arizona. Inspired by the exciting experience, Mr. Rodriguez is currently enrolled in our graduate program to pursue his advanced degree. Moreover, the PI has participated in the annual ExxonMobile Texas Science and Engineering Fair in the last few years [67] and plans to attend the San Antonio BEST (Boosting Engineering, Science, and Technology) robot competition [1], both of them are for high school students, to demonstrate the LEGO robots designed by students, expecting to expose the department and to attract more under-represented students to our program.

The teaching duty of the PI in UTSA involves instructing undergraduate and graduate lectures/classes, as well as supervising master and doctoral students in research. In the past few years, the PI had the opportunity to teach three undergraduate courses (Java Programming I, Java Programming II and Embedded Systems) and one graduate course on Operating Systems. From
these experiences, the PI realizes that how important it is to stimulate and inspire students with appropriate examples and how hands-on experiments help students with enhanced understanding of class materials. While striving to improve my teaching capabilities, in parallel with the line of research proposed in this project, the detailed curriculum developments and revisions that the PI intends to undertake are as follows:

- **developing a new graduate course on “Multicore and Real-Time Systems”**

  In order to attract more talented students to the proposed research area in this CAREER plan, the PI plans to offer a seminar-style course on “Multicore and Real-Time Systems” for doctoral and advanced master students starting Fall 2010. Beginning with a series of introductory lectures on multicore processors and real-time systems, the course will continue with paper reading and presentation by students. The topics of the papers to be selected will reflect the state-of-the-art and broad range of current research directions (e.g., real-time scheduling, energy management and fault tolerance schemes for multicore-based systems). Related to the proposed research plan, each student will choose a final project to enhance their understanding of the basic concepts and to enrich their hands-on research experiences. High quality research results from students’ projects will be published through journal articles, conference and workshop papers;

- **revising the curriculum for the graduate course on “Operating Systems”**

  As one of the four core courses, “Operating Systems” has the key importance in our graduate program and forms a prerequisite for many advanced graduate courses. Currently, the syllabus focuses (almost exclusively) on the topics of communication and distributed systems. Without questioning the importance of these topics, the PI believes that graduate “Operating Systems” should cover broader issues. As we are entering the multicore computing era, exposing the graduate students to the emerging technologies, while emphasizing the changing role of operating systems, is crucial in promoting research among graduate students. Consequently, eliminating the detailed discussions of specific middleware technologies (such as CORBA) while keeping the essential materials on distributed systems, the PI will augment the course with popular topics on multicore based systems (such as parallel scheduling and fault tolerance) and the related key concepts (e.g., process, thread and memory management) of operating systems;

- **enhancing the curriculum for the undergraduate course on “Embedded Systems”**

  As an effort to expose undergraduate students to hot research issues, the PI had developed an Embedded Systems course for Computer Science junior and senior students, which was taught in Summer 2007 and has become a regular upper elective course. Focusing on design principles and basic programming skills for embedded systems, the course adopted LEGO® MINDSTORM™ robots equipped with Handy Boards [31] as the programming platforms. The PI believes that robots are excellent platforms for students to enrich their knowledge in various Computer Science subjects (e.g., Computer Organization, Operating Systems, Networks and Programming). The required group projects could also cultivate the students to be responsible team-workers. The PI plans to enhance the course curriculum with discussions on parallel scheduling, real-time constraints and energy management. Through this course, the PI hopes to increase the motivation and awareness among the students towards research and advanced postgraduate studies.

  As an effective approach for integrating education and research, mentoring graduate students is another crucial activity in one’s academic career. Having graduated one master student, the PI is currently working with two talented PhD students on the research of “low-power reliable computing” and “multicore-based real-time systems”. In addition, the PI has served on three PhD dissertation committees and several master project committees.
With the involved theoretical/analytical research, simulator development and kernel-level coding, the proposed projects will provide the involved students with a variety of skills. Moreover, the development of the multicore-based system simulator for performance, energy and reliability should be a non-trivial contribution to the real-time systems research community at large. With the publicly available source codes and appropriate documentation, the simulator will facilitate the exploration/verification of multicore-related schemes/algorithms in both classes (such as the proposed new course on “Multicore and Real-Time Systems”) and research projects.

3 Project Timetable

The schedule for the research/teaching activities of this proposed project is planned as follows:

- **Year 1**: Establishment of the Real-Time Embedded Systems Laboratory; Acquire and embedded multicore platform and start to port LITMUS$^{RT}$; Investigate the component-based execution and resource modeling techniques for multicore-based systems; Develop a new graduate course on “Multicore and Real-Time Systems”;

- **Year 2**: Develop efficient multicore-aware real-time scheduling algorithms (including theoretical feasibility analysis) for multicore-based systems; Continue the work on porting LITMUS$^{RT}$ to the multicore platform; Revise the curriculums for the undergraduate course on “Embedded Systems” and the graduate course on “Operating Systems”;

- **Year 3**: Conduct research on energy management schemes for multicore-based real-time embedded systems and the corresponding feasibility analysis; Implement the multicore-aware real-time scheduling algorithms as kernel functions in LITMUS$^{RT}$ on the platform;

- **Year 4**: Develop flexible energy-efficient fault tolerance techniques in multicore-based systems and study the corresponding feasibility analysis; Design and develop a simulator incorporating the developed novel system models; Implement the energy-aware real-time scheduling algorithms for multicore systems in the platform and perform corresponding experiments to validate the energy model and evaluate the management schemes;

- **Year 5**: Implement the reliability-aware real-time scheduling algorithms on the platform and conduct corresponding experiments to evaluate the proposed scheduling algorithms; Finish the development of the simulator and as well as testing and documentation.

4 Results from Prior Support

Dr. Zhu is currently supported by one NSF grant (CNS-0720651) on “Towards an Integrated Framework for Low-Power Reliable Real-Time Embedded Systems” (a collaborative project with Dr. Hasan Aydin from George Mason University). The grant was awarded in 2007 for three years with the total amount of $284,983 (and UTSA share is $179,983). This project aims to develop the reliability-aware power management framework for uniprocessor real-time embedded systems, while taking the negative effects of DVFS on system reliability into consideration. Exploiting the backward recovery technique, various static and dynamic reliability-aware power management schemes have been developed for uniprocessor systems: such as for periodic real-time tasks scheduled with EDF [187, 188] and RMS [198]; tasks with variable reliability requirements [199]; systems with limited energy budget [180]; and optimistic schemes for tasks with probabilistic execution times [189]. Moreover, as the continuation of previous work, energy efficient redundant configurations have been studied for parallel embedded servers [197]. The grant currently supports one student towards his PhD at UTSA.