

Chapter 2

System Model

Abstract It is obvious that energy-aware scheduling problems largely depend on the tasks and platform under consideration. This chapter provides the system models that we consider in this book. Task models are presented in Sect. 2.1, which includes four types of tasks, namely, frame-based tasks, tasks with precedence constraints, periodic tasks, and sporadic tasks. Uniprocessor power consumption models are provided in Sect. 2.2. Multiprocessor platform models are presented in Sect. 2.3. Section 2.4 discusses other concepts and assumptions related to energy-aware scheduling on multiprocessor platforms. These concepts and assumptions are also important for energy-aware scheduling problems.

2.1 Task Models

Actually, various types of tasks/applications are considered for energy-aware scheduling. In this book, four typical task models are included; they are *frame-based tasks*, *tasks with precedence constraints*, *periodic tasks*, and *sporadic tasks*.

Frame-based Tasks: Frame-based tasks are a set of tasks, $\mathcal{T} = \{\tau_1, \tau_2, \dots, \tau_n\}$ that are released at the same time 0 and share a common deadline D . The execution requirement of task τ_i is denoted by C_i , which is defined as the Worst Case Execution Time (WCET) at the processor's maximum frequency. Denote $WCEC_i$ as the Worst Case Execution Cycles (WCECs) of task τ_i ; then, $C_i = WCEC_i / f^{max}$. For tasks with precedence constraints, C_i is defined with the same meaning. The utilization of task τ_i is defined as $u_i = C_i / D$.

Tasks with Precedence Constraints: Tasks in the set $\mathcal{T} = \{\tau_1, \tau_2, \dots, \tau_n\}$ that have precedence constraints are modeled by a weighted Directed Acyclic Graph (DAG). Let the DAG be (V, E) ; the node set V of this graph corresponds to tasks $\tau_1, \tau_2, \dots, \tau_n$. The edge set E corresponds to precedence constraints. The weight of each node, C_i represents the execution requirements of task τ_i . The overall

Fig. 2.1 An example of tasks with precedence constraints

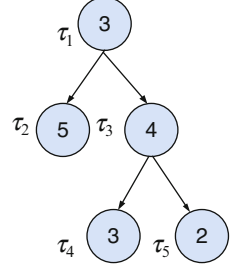
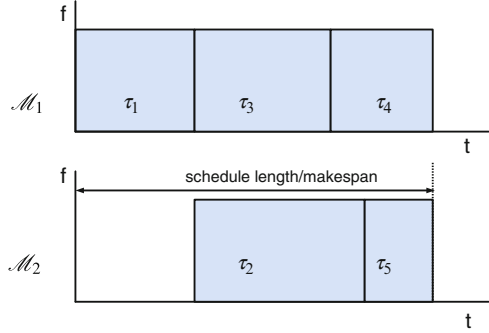


Fig. 2.2 A valid schedule on two processors



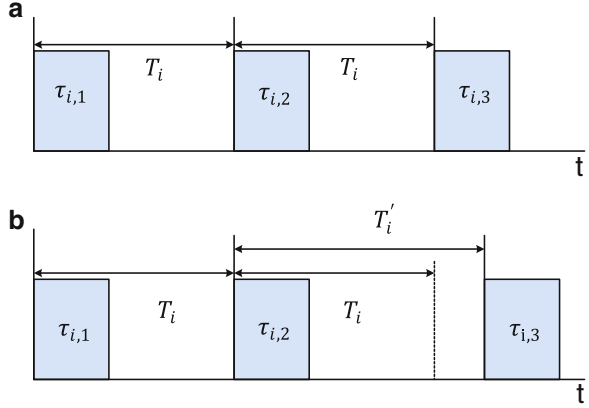
completion time of scheduling these tasks on multiple processors is called the *makespan* or *schedule length*, denoted by D . The utilization of task τ_i is defined as $u_i = C_i/D$.

Consider an example of five tasks with worst case execution times, $C_1 = 3$, $C_2 = 5$, $C_3 = 4$, $C_4 = 3$, and $C_5 = 2$. Their precedence constraints are shown in Fig. 2.1. A valid schedule with schedule length of 10 is shown in Fig. 2.2.

With the energy-aware consideration, two problems are usually associated with scheduling framed-based tasks and tasks with precedence constraints: minimizing the schedule length with an energy consumption constraint and minimizing energy consumption with a schedule length constraint.

Periodic Tasks: A periodic task is an infinite sequence of task instances (or called jobs), where each job/instance of a task comes in a regular period. Each task τ_i in a periodic task set $\mathcal{T} = \{\tau_1, \tau_2, \dots, \tau_n\}$ is described by (A_i, C_i, D_i, T_i) , where, A_i is its initial arrival time, C_i represents the worst cast execution time of its one job/instance, D_i represents its relative deadline, and T_i represents its period. If, initially, all of the tasks are released at the same time 0, which means $A_i = 0, \forall i, 1 \leq i \leq n$, then A_i is omitted; task τ_i is simply denoted by (C_i, D_i, T_i) . The j th job of τ_i is denoted by $\tau_{i,j}$. Besides, some useful terms are defined for periodic tasks. The utilization of a periodic task is defined as $u_i = C_i/T_i$. The total utilization of a task set is denoted by $U^{total} = \sum_{\tau_i \in \mathcal{T}} u_i$. The tasks are considered to have *implicit* deadlines if $D_i = T_i$ and are considered to have *constrained* deadlines if $D_i \leq T_i$. Oftentimes, the hyper-period H of a task set is defined as the minimal common multiplier of all periods

Fig. 2.3 Examples of a periodic task and a sporadic task. **(a)** An example of a periodic task. **(b)** An example of a sporadic task



of the tasks. Then, the number of task τ_i 's jobs during the hyper-period is H/T_i . Besides, a task τ_i 's density is defined as $\lambda_i = C_i/D_i$. A typical example of a periodic task τ_i with period T_i is shown in Fig. 2.3a. In the research community, a set of periodic tasks are generally considered as a whole.

Sporadic Tasks: A sporadic task is also an infinite sequence of task instances/jobs, where job arrivals have a minimal inter-arrival time rather than a fixed period. Obviously, a periodic task is a special kind of a sporadic task. Because of this relation between periodic tasks and sporadic tasks, the representation of periodic tasks can also be used for sporadic tasks, with the slight difference that T_i represents the sporadic task's minimal inter-arrival time. Some research approaches and results for periodic tasks can also be used for sporadic tasks. A typical example of a sporadic task τ_i with minimal inter-arrival time T_i is shown in Fig. 2.3b. Notice that T_i is just the minimal inter-arrival time. Practical inter-arrival times can be greater than T_i . As shown in Fig. 2.3b, the inter-arrival time between $\tau_{i,2}$ and $\tau_{i,3}$ is $T'_i > T_i$. Generally, a set of sporadic tasks are considered as a whole.

2.2 DVFS Models

Our work in this book focuses on energy-aware scheduling on multiprocessor platforms. In some sense, multiprocessor platforms can be regarded as consisting of multiple uniprocessors. In this section, we will present the power consumption model for a single processor first. Both homogeneous and heterogeneous multiprocessor platform models will be described in the next sections.

The practical power consumption model of a DVFS processor is quite complex and varies among different manufacturing technologies.

Generally, a DVFS processor can be in three kinds of operation modes: *active* mode, also called busy mode or run mode, which refers to the mode in which the processor is executing tasks; *idle* mode, which refers to the mode in which the processor is on, but there is no task running on it; *shutdown* mode, also called dormant mode or sleep mode, which refers to the mode in which the processor is not running any tasks, is put in a very low power state, and can be recalled to active mode when new tasks arrive. If a processor can be put into a shutdown mode, it is called a *shutdown-enabled* processor; otherwise, it is called a *shutdown-disabled* processor. A shutdown-disabled processor has to stay in idle mode and cannot be put in shutdown mode, even when the processor has no task to execute.

In **active** mode, we use the general model which has been verified with the SPICE simulation in [1]. In this model, the active power consumption is given by:

$$P^{act} = P^{dyn} + P^{sta} + P^{on}$$

where P^{dyn} is the dynamic power consumption due to switching activity, P^{sta} is the static power consumption due to leakage current, and P^{on} is the intrinsic power consumption needed to keep the processor on and is assumed to be a constant.

The dynamic power, P^{dyn} , is given by:

$$P^{dyn} = C_e V_{dd}^2 f$$

where C_e is the average switched capacitance per cycle, V_{dd} is the supply voltage, and f is the clock frequency.

The static power, P^{sta} , is given by:

$$P^{sta} = V_{dd} I_{subn} + |V_{bs}| I_j$$

in which I_{subn} is the sub-threshold leakage current, given by:

$$I_{subn} = K_3 e^{K_4 V_{dd}} e^{K_5 V_{bs}}$$

where K_3 , K_4 , K_5 are constants. V_{bs} is the voltage applied between body and source, and I_j is the reverse bias junction current. Besides, there is a relation between operating frequency, supply voltage, and threshold voltage:

$$f = (V_{dd} - V_{th})^\gamma / L_d K_6$$

where γ and K_6 are constants, and L_d represents the logic depth. The threshold voltage is given by:

$$V_{th} = V_{th1} - K_1 V_{dd} - K_2 V_{bs}$$

where all V_{th1} , K_1 , K_2 are platform-specific constants.

Assuming that the minimum and maximum operating frequencies are f^{min} and f^{max} , respectively, there are two types of processors: a processor is called *ideal* if it can operate at any frequency in the range $[f^{min}, f^{max}]$, and it is called *non-ideal* if it can only operate at a set of discrete frequencies in this range. If it can operate

on K different frequencies, denote $\{f^{(1)}, f^{(2)}, \dots, f^{(K)}\}$ as the available frequency set. Without loss of generality, assume that $f^{min} = f^{(1)} < f^{(2)} < \dots < f^{(K)} = f^{max}$. f^{min} can be or cannot be 0, and f^{max} can be or cannot be $+\infty$, according to different assumptions. For a processor executing at frequency f , its speed s can be defined as $s = f / f^{max}$.

This general active power consumption model is quite practical and complicated, but in most research, a lot of assumptions are made under different contexts, and the model is thus simplified based on this general model, as we will present later for separate papers.

In **idle** mode, the power consumption is denoted as P^{idl} , which may be much less than the active power. It's exact value is determined by different assumptions or by measuring practical platforms.

In **shutdown** mode, the power consumption is denoted as P^{sh} , which is much less than P^{idl} . It is assumed to be negligible in a lot of research literature or just a small percent of P^{idl} in other ones.

Since the processor can transit between different modes, there might be some transition overhead during the transitions. Generally, it is assumed that the transition between an active mode and an idle mode can be done immediately and requires no additional energy overhead. However, the transition between idle mode and shutdown mode may require both time and energy overhead, denoted by t^{ov} and E^{ov} , respectively.

2.3 Platform Models

This section will describe multiprocessor models, which consist of a homogeneous multiprocessor platform and a heterogeneous multiprocessor platform.

Homogeneous Platforms: Roughly speaking, if all of the processors on a multiprocessor platform are identical, this platform is regarded as a *homogeneous* platform. For homogeneous platforms, if all of the processors must operate at the same supply voltage and clock frequency at any time, it is called a *dependent* platform, which is the common case in a chip multiprocessor (CMP) or, in other words, a multi-core chip. For a dependent platform, although all of the processors must operate at the same supply voltage and clock frequency, they may transit into idle mode or shutdown mode independently. If processors on a platform can operate on different supply voltages and clock frequencies, and can adjust their values independently, it is called an *independent* platform. In addition, a new type of homogeneous platform is considered in previous and current researches. This platform is called a partitioned multi-core, where all cores on the platform are partitioned into different islands/blocks. Cores from the same block/island are dependent, while cores from different blocks/islands are independent.

Heterogeneous Platforms: If any processor is not identical to another one on a multiprocessor platform, this platform is deemed a *heterogeneous* platform. A typical example of a heterogeneous platform is one containing both DVFS processors and non-DVFS Processing Units (PUs), such as FPGA. For non-DVFS Processing Units (PUs), if its power consumption is dependent on the workload assigned to it, it is called a *workload-dependent* non-DVFS PU. Otherwise, it is called a *workload-independent* non-DVFS PU.

Also, a heterogeneous platform may consist of DVFS processors with different frequency ranges or different power consumption functions. Consider a heterogeneous platform with m DVFS processors, $\mathcal{M}_1, \mathcal{M}_1, \dots, \mathcal{M}_m$. Since processors may have different configurations and power consumptions, the previously presented power consumption model should indicate this aspect by simply adding a processor index. Namely, P_j^{act} , P_j^{dyn} , and P_j^{sta} represent the active power, dynamic power, and static power of processor \mathcal{M}_j , $j \in [1, \dots, m]$. P_j^{on} represents the intrinsic power to keep the processor \mathcal{M}_j on. The frequency range of processor \mathcal{M}_j is represented by $[f_j^{min}, f_j^{max}]$.

Tasks' worst case execution requirements are also dependent on which processor the tasks are assigned to. Thus, $c_{i,j}$ is used to represent the Worst Case Execution Time (WCET) of task τ_i when it is assigned to processor \mathcal{M}_j at the maximum frequency f_j^{max} . $c_{i,j} = WCEC_i / f_j^{max}$. Similarly, $u_{i,j} = c_{i,j} / T_i$ is defined as the utilization of τ_i when it is assigned to \mathcal{M}_j . The utilization of processor \mathcal{M}_j is the sum of utilizations of all the tasks that are assigned to it, denoted by $U_j = \sum_{\tau_i \in \mathcal{T}_j} u_{i,j}$, where \mathcal{T}_j is the task set assigned to processor \mathcal{M}_j .

For shutdown-enabled processors, the time overhead and energy overhead of putting the processor into shutdown mode and returning it to active mode are denoted by t_j^{ov} and E_j^{ov} , respectively. Generally, heterogeneous platforms are assumed to be independent.

Notations that are consistently used in this book are provided in Table 2.1.

2.4 Other Related Concepts

This section will describe some important concepts that are related to energy-aware scheduling. They reflect some characteristics of practical platforms and tasks. When they are taken into consideration, more constraints and requirements should be included in the energy-aware scheduling algorithms.

Slack Reclamation: A processor may stay idle for a short period of time, even when it is allocated to execute some tasks, because of a task's late arrival or advanced completion. These idle periods of time are referred to as slacks. Generally, there are two kinds of slacks. One of them results from the scheduling scheme

Table 2.1 Notations used in this book

Notation	Description
\mathcal{T}	Task set $\{\tau_1, \tau_2, \dots, \tau_n\}$
τ_i	The i th task in task set \mathcal{T}
n	The number of tasks in \mathcal{T}
$\tau_{i,j}$	The j th job of a periodic/sporadic task τ_i
A_i	Initial arrival time of task τ_i
$WCEC_i$	Worst case execution cycles of task τ_i for frame-based tasks or precedence-constrained tasks or WCEC of a job of periodic/sporadic task τ_i
C_i	Worst case execution time of task τ_i for frame-based tasks or precedence constrained tasks or WCET of a job of periodic/sporadic task τ_i
D	The common deadline of a frame-based task set \mathcal{T}
D_i	Relative deadline of task τ_i
T_i	Period of periodic task τ_i ; or minimal inter-arrival time of sporadic task τ_i
u_i	Utilization of τ_i
U^{total}	Total utilization of task set \mathcal{T}
λ_i	Density of task τ_i
H	Hyper-period of a periodic task set \mathcal{T}
\mathcal{M}_j	j th processor/core or j th processor type
$p^{act}_j(p^{act})$	Active power consumption (of \mathcal{M}_j)
$p^{dyn}_j(p^{dyn})$	Dynamic power consumption (of \mathcal{M}_j)
$p^{sta}_j(p^{sta})$	Static power consumption (of \mathcal{M}_j)
$p^{on}_j(p^{on})$	Intrinsic power to keep processor (\mathcal{M}_j) on
$p^{idl}_j(p^{idl})$	Power consumption (of \mathcal{M}_j) in idle mode
$p^{sh}_j(p^{sh})$	Power consumption (of \mathcal{M}_j) in shutdown mode
$f^{min}_j(f^{min})$	Minimum frequency of processor (\mathcal{M}_j)
$f^{max}_j(f^{max})$	Maximum frequency of processor (\mathcal{M}_j)
s	Processor speed defined as f/f^{max}
$f^{(k)}(f^k)$	The k th frequency level of processor (\mathcal{M}_j)
$c_{i,j}$	WCET of τ_i when it is assigned to processor \mathcal{M}_j
$u_{i,j}$	Utilization of task τ_i assigned to \mathcal{M}_j
\mathcal{T}_j	Task set assigned to \mathcal{M}_j
U_j	Utilization assigned to \mathcal{M}_j
$t^{ov}_j(t^{ov})$	Switching time overhead (of \mathcal{M}_j)
$E^{ov}_j(E^{ov})$	Switching energy overhead (of \mathcal{M}_j)
m	The number of processors

itself, while the other one results from the difference between a task's Actual Case Execution Time (ACET) and Worst Case Execution Time (WCET). In energy-aware scheduling, the slacks can be used to slow down or shut down the processor to achieve the goal of saving energy.

Task Preemption: If the current task under execution can be preempted by higher priority tasks, this task is considered to be *preemptive*; otherwise, it is considered to be *non-preemptive*. Schedulability conditions for preemptive scheduling and non-preemptive scheduling are quite different, resulting in the difference between energy-aware preemptive scheduling and non-preemptive scheduling.

Fixed/Dynamic Priority Scheduling: If priority values of tasks will not change during runtime, the scheduling is considered to be of *fixed priority*; otherwise, it is considered to be of *dynamic priority*. Rate Monotonic (RM) scheduling, which ranks the tasks according to tasks' arrival frequencies/rates, is a typical example of fixed priority scheduling. Earliest Deadline First (EDF), which dynamically assigns priorities according to the task job's deadlines during runtime, is a typical example of dynamic priority scheduling.

Partition-based/Global Scheduling: The most commonly used approach of scheduling tasks on multiprocessor platforms is *partition-based scheduling*, where each task is assigned statically to one processor. Partition-based scheduling allows schedulability to be verified by well-understood single-processor analysis techniques. Partition-based scheduling also requires less scheduling overhead on practical platforms. On the other hand, we have *global scheduling*, in which there is a single job queue from which jobs are dispatched to any available processor according to a global priority scheme. Global scheduling allows different instances/jobs of the same task to be executed upon different processors. Each instance/job can start its execution on any processor and may migrate during runtime from one processor to another if it gets preempted by a higher priority job.

Task Migration: Task migration means that a task need not be entirely executed on one processor. Given that task migration is allowed, for frame-based tasks and tasks with precedence constraints, one task's former part and the rest can be executed on more than one processor sequentially; for periodic tasks, a task's single job's former part and the rest can be executed on more than one processor sequentially; besides, jobs released early and later jobs can be executed on different processors. If task migration is not allowed, the whole part of a task must be executed on one processor. It is obvious that whether task migration is allowed or not has a great influence on a scheduling strategy. Consequently, it will also influence energy-aware scheduling.

Single Task Concurrent Execution (Parallel Task Execution): Traditionally, it is assumed that one job instance of a task is unable to be executed on more than one processor simultaneously, even when task migration is allowed. Unlike this traditional assumption, in some research, it is assumed that one job of a task is able to be executed on two or more processors concurrently and simultaneously; although, early jobs must be completed before the subsequent job can begin to be executed. This type of task is also called parallel tasks. Figure 2.4 provides a trivial example of scheduling a periodic task under this assumption. This assumption provides more space and the opportunity to use DVFS to achieve the goal of saving energy, and it makes the problem more complex and difficult at the same time.

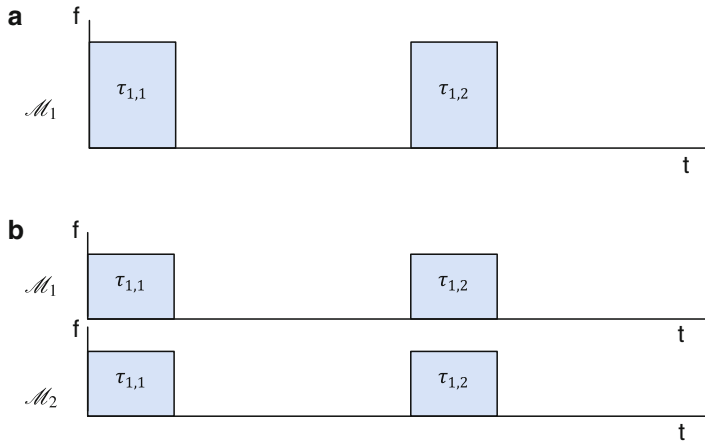


Fig. 2.4 An example of a single task being concurrently executed. (a) Executed on one processor. (b) Concurrently executed on two processors

Application-specific Power Consumption: Sometimes, it is assumed that the power consumption function of a processor is dependent on the tasks running on it, which is also the case in many practical systems. Thus, it is assumed that some constants in the power consumption model of a processor are dependent on the task(s) assigned onto it. When taking this aspect into consideration, problems become more complex than when a processor's power consumption is the same for any task.



<http://www.springer.com/978-1-4614-5223-2>

Energy-aware Scheduling on Multiprocessor Platforms

Li, D.; Wu, J.

2013, VII, 59 p. 28 illus., Softcover

ISBN: 978-1-4614-5223-2