Chapter 1
Introduction

1.1. Background

Parallel processing of tasks is essential to meet the demand of increasing computing requirements of complex applications. Modern embedded-systems usually include multiple general-purpose CPUs, specialized processors like DSPs, peripherals, communication buses and memory. Applications need to be suitably scheduled on to these embedded devices to optimize the overall performance.

An application is typically modeled as a precedence constrained task graph [75] and represented as a *Directed Acyclic Graph* (DAG), where the nodes represent tasks and the directed edges represent execution dependencies as well as the amount of communication. Scheduling of tasks on the processors (*task scheduling*) and edges on communication channels (*edge scheduling*) adhering to the precedence constraints is referred to as the general *Multiprocessor Scheduling Problem*. The primary scheduling objective is usually to *minimize the overall execution time* of the task graph under various constraints. Typically, in SoCs with more than one processors, power consumption is also an important concern. Nowadays programmable interfaces for dynamic voltage scaling and switching off idle hardware to reduce the overall power consumption are available. Hence, *optimizing power consumption* has become a high priority scheduling objective.

The existing solutions to the scheduling problem are mainly of two types: *static* and *online*. In static scheduling [4, 61, 75], scheduling decisions such as individual task mappings on processors and edge mappings on communication subsystems are taken at compile-time. In online scheduling [37, 92], the above scheduling decisions are taken at the run-time. In the literature, static scheduling has usually been preferred over online scheduling for precedence constrained task graphs to reduce the scheduling overhead at run-time.

Copyright IIT Kharagpur

A *static task graph* is usually referred to as a task graph whose structure or execution time of individual tasks are known a-priori and remain constant at run-time. Traditionally significant research has focussed on static task graph models. A *dynamic task graph* is referred to as a task graph whose graph structure or execution time of nodes may vary at run-time. A conditional task graph (CTG) [33, 122] is a special DAG in which control flow is captured at the conditional nodes. After execution of a conditional node, depending on the condition evaluated, one of the execution paths is selected for further execution and other alternative paths are discarded. Thus, the final sub-graph that gets executed is decided at run-time and hence, it belongs to the class of dynamic task graphs. Moreover, the exact execution time of a task at run-time is hard to predict [35, 98]. This is because each task in a task graph may be a macro task, it may contain loops and conditions which change the execution time of the task at run-time. Such task graph models [98, 119] with variable execution time also qualify as dynamic task graphs.

The general multiprocessor scheduling problem to minimize the execution time is known to be NP-complete. Further, parameters such as arbitrary communication/computation cost [61, 74], presence of conditional tasks [33, 122], contention in communication channels [114], homogeneous/heterogeneous nature in the processors [12], task duplication [3, 11], arbitrary connected processor models [31] and storage constraints [99] increase the solution space by many-fold. Amongst a variety of such assumptions and architecture models that have emerged in the literature, this thesis focusses on *non-preemptive* task graphs and *homogeneous* processor models.

Due to the intractable nature of the problem, heuristic based solutions are usually preferred by researchers and these solutions focus on solving the scheduling problem efficiently and providing near-optimal results. Traditionally, *List Scheduling* [61, 74] based heuristic solutions have been preferred because of their low complexity nature. In list scheduling, each task is assigned a priority and an ordered list is prepared in decreasing order of their priorities. Then the tasks are sequentially assigned to favorable processors. At each scheduling step the communication edges of the scheduled task are mapped on to the communication channels by using a suitable edge scheduling technique. The key aspects of a list scheduler include priority calculation and processor selection for a task using schemes such as dynamic critical path based priority [74]. In some studies additional optimization techniques [55, 98, 125] such as genetic algorithm and simulated annealing are combined with list scheduling.

Copyright IIT Kharagpur

In the domain of scheduling with the objective to minimize power consumption, there are multiple power consumption factors that are studied in the literature [124]. *Static Power Management* (SPM) [49, 129] and *Dynamic Power Management* [131, 107] techniques are used to reduce the overall power consumption. In the context of dynamic task graph scheduling, *Dynamic Power Management* techniques become more relevant. In dynamic task graph schedules, slacks are produced continuously during execution due to early finish of tasks or conditional paths that are not executed. These slacks can be effectively used to reduce the *Dynamic Power* consumption by a suitable *Dynamic Voltage Scaling* (DVS) [124, 131] methodology. Since there is a quadratic dependency of dynamic power consumption on the supply voltage, the reduction in supply voltage decreases the energy consumption considerably. The lowering of supply voltage also decreases the maximum operating frequency and that results in increase in the task execution time. Thus deadline guarantee becomes an important constraint in DVS techniques.

The work presented in this thesis primarily aims at scheduling problems of dynamic task graphs on multiprocessor systems. Schedule length or Makespan minimization and online slack distribution for power reduction are the two primary objectives in this work. Experimental results on a large set of task graphs and on multiprocessor systems in the range of $2-8$ processors have been presented. The proposed algorithms also intend to be efficient and incur low overhead at run-time.

The rest of the chapter is organized as follows. Section 1.2 explains the representations of static and dynamic task graphs with an example. Motivation and objectives of the thesis are explained in Section 1.3. The contributions of the thesis is given in Section 1.4. Finally, rest of the thesis organization is provided in Section 1.5.

1.2. Static and Dynamic Task Graphs

We describe the basic representations of static and dynamic task graphs with an example. As discussed earlier, a dynamic task graph can be a conditional task graph and/or it contains tasks whose execution times vary at run-time. Figure 1.1(a) shows a conditional task graph with a conditional node v_3. Depending on whether v_3 is evaluated to A or A', v_6 or v_7 is selected for further execution. If we consider v_3 to be a normal node instead of a conditional node, the representation reduces to that of a static task graph. Figure 1.1(b) shows a dynamic task graph with 4 nodes with two nodes v_2 and v_3 having variable

Copyright IIT Kharagpur

execution times at run-time. All communications are assumed to take 3 units of time. The execution times of v_1 and v_4 are fixed (to be 5 each), where as, v_2 and v_3 can complete their execution within the execution time $(7, 10)$ and $(4, 7)$ at run-time, respectively. If we assume that the tasks v_2 and v_3 always execute at their worst-case execution times which are shown to be 10 and 7, respectively, it becomes the representation of a static/deterministic task graph.

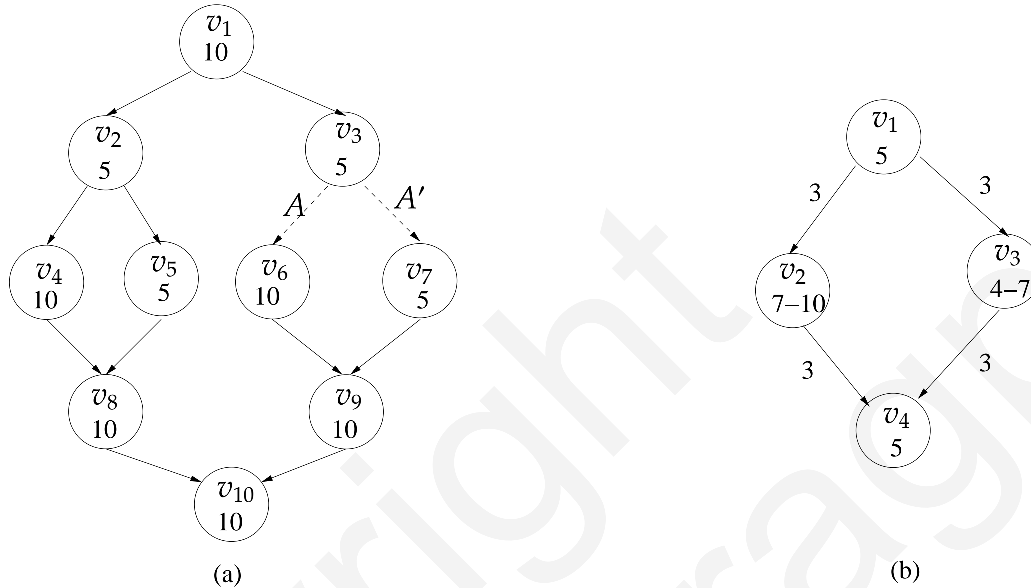

Figure 1.1: a) An Example Task graph with a conditional node v_3 b) An Example task graph two variable time execution nodes v_2 and v_3

1.3. Motivation

Static task graph scheduling techniques have evolved to many classes of sub-problems such as scheduling without communication, scheduling with communication and contention, scheduling under duplication schemes, etc. However, the focus on *dynamic task graphs* has been limited and the existing solutions are mainly extensions of the scheduling algorithms of static task graphs. Since applications and architectures have become complex over time, dynamism in task graphs at execution-time is sometimes unavoidable. This has been the primary motivation towards developing scheduling solutions for dynamic task graphs. Observations stated below encouraged us to pursue this research.

1. **Online Scheduling:** Static scheduling is generally preferred over online scheduling in static task graphs because in static scheduling all the scheduling decisions are known before program execution and actual

Copyright IIT Kharagpur

scheduling overhead is minimized. However, for dynamic task graphs, we have observed that a static schedule is not always sufficient and making schedule adjustments at run-time is required to provide a better schedule length. Advanced static scheduling techniques consider task variations at compile-time. However, they eventually provide a static schedule which cannot be adjusted at run-time. These static schedules fix the tasks on to the processors and thus limit the scope of schedule adjustments. We observed that there are cases when an online scheduler is essential for improvements in the schedule length of dynamic task graphs. We explain this with an example. Figure 1.2(a) shows a static schedule of the task graph shown in Figure 1.1(b) on a two processor system, pe_1 and pe_2. The worst-case schedule length of this static schedule is 23. (We may note here that communication time between two nodes scheduled on to the same processor is considered to be zero and therefore the corresponding edges are not shown in the diagram). It can be observed from Figure 1.2(b) that a schedule length of 20 is obtained which is optimal when v_3 finishes early by 3 time units. Similarly, Figure 1.2(c) shows that the a schedule length of 23 is obtained when v_2 finishes early by 3 time units. Figure 1.2(d) shows for the later case, by executing v_4 on pe_2, a better schedule of schedule length 20 can be obtained. However, v_4 cannot be statically moved to pe_2 because it will result in increase in schedule length for the case where v_2 executes at its worst-case time leading to a worse schedule length of 23 compared to the previously obtained schedule length of 20. It can be observed that no static schedule can be created that can work well for both the cases of v_2 finishing early and v_3 finishing early and hence, the processor assignment for the node v_4 needs to have some sort of online consideration.

2. **Overhead and Hybrid Scheduling:** Scheduling overhead is another reason for static scheduling getting preference over online scheduling. A purely online scheduler may incur significant overhead if it has to perform the scheduling as well as adjustments at run-time. We observed that an approach in which online scheduling is guided by prior offline processing can be an effective solution to minimize overall execution time. In a hybrid technique, processing intensive scheduling decisions can be made offline and the scheduling calculations that are critical for schedule adjustments can be made part of the online scheduler.

Copyright IIT Kharagpur

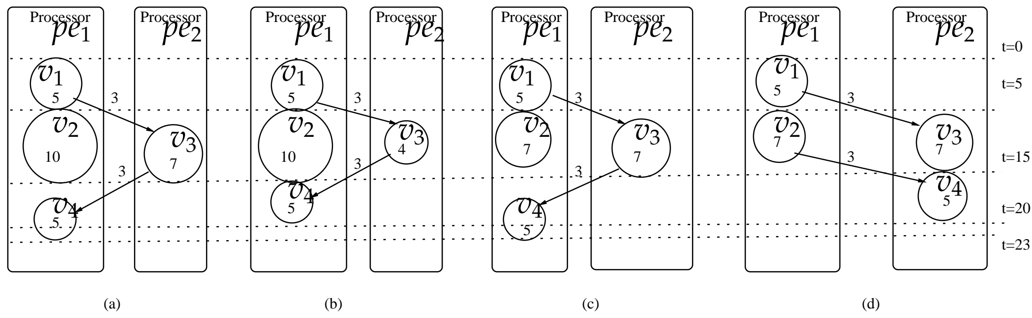

Figure 1.2: a) Static Schedule of 1.1(b) on two processors, b) Optimal Static Schedule with v_3 is finishing early, c) Static Schedule with v_2 finishing early, d) Online remapping with v_2 finishing early

3. **Slack Distribution for Power Reduction**: In dynamic task graphs, tasks may finish early or some tasks may not execute due to conditional paths leading to slack formation in the course of execution. In the presence of multiple processors, slack distribution algorithms such as [130] have been proposed. These methods use the slack to prolong the execution of a few tasks and reduce the supply voltage of the processors. We have observed that by analyzing a few properties of the dynamic task graph schedules, significant improvements can be achieved. Since the slack distribution by the above analysis can again lead to significant overhead at run-time, hybrid strategies are essential to keep the overhead of the online scheduler low.

1.4. Contributions of this work

The main contribution of this work is to address the dynamic task graph scheduling problem for schedule length and power minimizations. The contributions include:

- **Hybrid Scheduling of Dynamic Task Graphs with Limited Duplication**: We develop a hybrid (mixing static and online) scheduling algorithm for task graphs without communication under memory constraints.

- **Online Scheduling of Dynamic Task Graphs with Communication and Contention**: In the presence of communication and contention, we develop an online scheduling strategy for dynamic task graphs for two types of communication models, namely, point-to-point and broadcast models.

Copyright IIT Kharagpur

- **Slack Distribution in Dynamic Task Graph Schedule**s **for Power Reduction**: We present an online slack distribution methodology to reduce power consumption using offline task graph mapping analysis.

We now provide an overview of the above methodologies.

1.4.1. Hybrid Scheduling of Dynamic Task Graphs with Limited Duplication

The first work in this thesis addresses the scheduling problem of dynamic task graphs on multiprocessors under memory constraints. We observed that due to dynamism in the tasks, the workload distribution in processors becomes imbalanced in static schedules at run-time. The objective of this work is to improve the schedule length in such cases by mixing static and online techniques.

The input model accepts a given conditional task graph $G(V_s, V_c, E_s, E_c)$ where each task $v_i \in G$ can take any value between $BCET_{v_i}$ and $WCET_{v_i}$ at run-time and has a storage memory requirement of cs_{v_i}. The communication cost between nodes has been neglected in this work. A given number of limited homogeneous processors are available and each processor pe_j has a local code memory of size M_{pe_j}. The scheduling problem in this work is to schedule the task graph on the processors and minimize the schedule length. The code memory constraint imposes restrictions on the number of tasks that each processor can hold and imposes a limit on the number choices among the processors on which a node can be scheduled at run-time.

We have presented the solution methodology as a three phase strategy. In the first phase (static mapping), a static scheduling is performed on the processors and a schedule graph (G_S) and a concurrency graph (G_C) are constructed. In the second (selective duplication) phase, G_s and G_c are analyzed and based on few derived properties, a set of nodes are eliminated from duplication which are not likely to produce schedule length improvement with a possible rescheduling. Each node v_i in the remaining set is assigned a weight which denotes a possible schedule length gain $W_{v_i \rightarrow pe_j}$, if a rescheduling is performed at run-time on pe_j. Then, each processor pe_j is assigned a set of nodes such that $\sum W_{v_i \rightarrow pe_j}$ is maximized and the memory constraint in each processor is met i.e $\sum cs_{v_i} \leq M_{pe_j}$. The third (online) phase is a run-time scheduling algorithm that maintains a list of free nodes and assigns them to suitable processors at run-time. We have developed online scheduling strategies for both deadline guarantee as well as without a necessary deadline guarantee.

Copyright IIT Kharagpur

We have compared the proposed strategy with the static scheduler proposed by Xie & Wolf [122] and critical path based algorithm [74] with necessary modifications for the comparison. Experimental results indicate that this technique provides better average schedule length (8% – 20%) compared to static schedulers in the presence of dynamism in task graphs. The effects of model parameters like number of processors, memory and various task graph parameters on performance are investigated in this work. Experimental results also indicate that the average schedule length improves with increasing dynamism in the task graphs and with increasing availability of local memory in each processor.

1.4.2. Online Scheduling of Dynamic Task graphs with Communication and Contention

The second problem in this thesis addresses the dynamic task graph scheduling problem with non-negligible communications associated with the edges. We observed that in the presence of communication and limited number of channels, online scheduling of tasks becomes even more important.

Similar to the first problem, a dynamic task graph and a set of limited number of processors are taken as input. In this task model, each edge e_{ij} is associated with communication $c(e_{ij})$. The processor model is specified by a set of homogeneous processors connected by a communication system B with $z(B)$ number of independent channels connected to all the processors. A channel can carry only one communication at a particular time so it may lead to channel contention at run-time. Two types of communication models have been assumed, namely, point-to-point and broadcast communication models. In point-to-point communication model, the target processor of a communication has to be identified and communicated with and in the broadcast communication model, a communication is broadcast to all the processors. The objective of this problem is to schedule the tasks on the processors and edges on the communication channels to minimize the schedule length.

In this work, we have presented an online scheduling methodology as a solution. The global online scheduler is modeled as an event based scheduler that acts on a *task completed event* or a *free processor event*. In the broadcast model, in a task completed event, the outgoing communications of the completed task are scheduled if the child task is not likely to be scheduled on the same processor. The actual allocation of the child task is deferred to a later time. In each free processor event, among the ready tasks, the highest priority task is

Copyright IIT Kharagpur

assigned to the processor. In the point-to-point communication model, in a task completed event, additionally, a suitable processor for each of the child nodes of the task is identified and communicated with. This processor allocation of a child task is done by analyzing the already scheduled tasks on each processor pe_j as well as on the data ready time of the task on that processor ($drt(v_i, pe_j)$). The methodology has been designed to maintain a low overhead of scheduling specifically for limited number of processors and channels.

We compared the proposed strategy with the contention-aware static scheduler proposed by Sinnen and Sousa [114] combined with modifications for conditional task graph scheduling proposed by Xie & Wolf [122]. Experimental results indicate that this technique adapts better to variation in task graphs at run-time and provides better schedule length ($12\% - 22\%$) compared to a static scheduling methodology. The effects of model parameters like number of processors, memory and other task graph parameters on performance are investigated in this work. We show that this technique works well for dynamic task graphs with high dynamism, number of processors in the range of $2 - 6$ and moderate communication to computation ratio.

1.4.3. Slack Distribution in Dynamic Task Graph Schedules for Power Reduction

In the final work, we address the problem of slack distribution in dynamic task graph schedules with power reduction objective. We observe that due to dynamism in the task graph, slacks are continuously produced throughout the execution of the task graph. In the work described in the earlier two subsections, slacks are used to improve the schedule length. In this work, the slack produced at run-time is used to dynamically scale supply voltage but with the constraint of meeting the overall deadline. Though several online slack distribution strategies [130] exist in the literature, we observe that an offline analysis with knowledge of a few properties of the dynamic task graph may assist the online slack distribution strategies and improve on the energy savings while maintaining low overhead.

The processor model assumes a set of voltage scalable processors with continuous ($V_{min} - V_{max}$) or discrete voltage levels. This work uses the currently available slack (s_t) as well as estimated future slack FS_{τ_i} and estimated future workload FW_{τ_i} to distribute the slack at run-time. In the context of a dynamic task graph mapped on to multiprocessors, these estimations are complex compared to that of its single processor counter part.

Copyright IIT Kharagpur

We present an inter-task slack distribution scheme for this problem and describe it as a two phase strategy. In the offline phase, we calculate the FS_{τ_i} and FW_{τ_i} of each task τ_i with a novel computation chain extraction process. The procedure for this calculation is derived from effective slack and workload calculations of simple task chains in series and parallel combinations. In the online phase, a local online scheduler uses the values obtained from the offline phase and makes a simple calculation $s_\tau = min(s_t, \frac{e^w_\tau}{FW_\tau} \cdot (s_t + FS_\tau))$ to decide on the slack allocated (s_τ) to each task τ and rest of the slack is forwarded. This ensures $O(1)$ complexity at run-time as the other deciding parameters of slack distribution are calculated in the offline phase.

We compare the proposed technique with greedy and path based slack distribution techniques. Results indicate up to $6\% - 13\%$ more energy saving can be achieved with this methodology over such methods.

1.5. Thesis organization

The rest of the thesis is organized as follows:

Chapter 2: A literature survey on multiprocessor scheduling algorithms is provided in this chapter. We have discussed basic list scheduling techniques along with advanced algorithms for both schedule length and power minimization.

Chapter 3: In this chapter, we present a hybrid scheduling algorithm for dynamic task graphs under memory constraints. We present a three phase strategy with limited duplication in this work.

Chapter 4: An online scheduling algorithm is presented that is intended for the scheduling problem of a dynamic task graphs with communication cost on a target architecture consisting of limited number of communication channels leading to possible contention at run-time. Two types of communication models, namely, point-to-point and broadcast models are considered.

Chapter 5: A new slack distribution strategy that can be used to adjust the supply voltage at run-time is described in this chapter which is guided by an offline task graph mapping analysis.

Chapter 6: Finally a conclusion of the work is presented with a summary of the work done in this thesis. We discuss the possible extensions and future scope in this chapter.

Copyright IIT Kharagpur