SCHEDULING HEURISTICS FOR MAXIMIZING OUTPUT QUALITY OF IRIS TASK GRAPHS IN MULTIPROCESSOR ENVIRONMENT WITH TIME AND ENERGY BOUNDS

A Thesis Presented

by

RAJESWARAN C RAVINDRAN

SCHEDULING HEURISTICS FOR MAXIMIZING OUTPUT QUALITY OF IRIS TASK GRAPHS IN MULTIPROCESSOR ENVIRONMENT WITH TIME AND ENERGY BOUNDS

A Thesis Presented

by

RAJESWARAN C RAVINDRAN

Approved as to style and content by:

_____

C Mani Krishna, Co-chair

_____

Israel Koren, Co-chair

_____

Michael Zink, Member

_____

Christopher V. Hollot, Department Chair
Electrical and Computer Engineering

ABSTRACT

SCHEDULING HEURISTICS FOR MAXIMIZING OUTPUT QUALITY OF IRIS TASK GRAPHS IN MULTIPROCESSOR ENVIRONMENT WITH TIME AND ENERGY BOUNDS

MAY 2012

RAJESWARAN C RAVINDRAN

B.E., MADRAS UNIVERSITY, CHENNAI, INDIA

M.S.E.C.E., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor C Mani Krishna and Professor Israel Koren

Embedded real time applications are often subject to time and energy constraints. Real time applications are usually characterized by logically separable set of tasks with precedence constraints. The computational effort behind each of the task in the system is responsible for a physical functionality of the embedded system. In this work we mainly define theoretical models for relating the quality of the physical functionality to the computational load of the tasks and develop optimization problems to maximize the quality of the system subject to various constraints like time and energy. Specifically, the novelties in this work are three fold. This work deals with maximizing the final output quality of a set of precedence constrained tasks whose quality can be expressed with appropriate cost functions. We have developed heuristic scheduling algorithms for maximizing the quality of final output of embedded applications rather

than intermediate quality which has not been dealt with before. This work also deals with the fact that the quality of output of a task in the system has noticeable effect on quality of output of the other dependent tasks in the system. Finally run time characteristics of the tasks are also modeled by simulating a distribution of run times for the tasks, which provides for averaged quality of output for the system rather than un-sampled quality based on arbitrary run times.

Many real-time tasks fall into the IRIS (Increased Reward with Increased Service) category. Such tasks can be prematurely terminated at the cost of poorer quality output. In this work, we study the scheduling of IRIS tasks on multiprocessors. IRIS tasks may be dependent, with one task feeding other tasks in a Task Precedence Graph (TPG). Task output quality depends on the quality of the input data as well as on the execution time that is allowed. We study the allocation/scheduling of IRIS TPGs on multiprocessors to maximize output quality. The heuristics developed can effectively reclaim resources when tasks finish earlier than their estimated worst-case execution time. Dynamic voltage scaling is used to manage energy consumption and keep it within specified bounds.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

An embedded system is a collection of computational and physical components that coordinate with each other to achieve a given objective by a specified deadline. The deadline has a fundamental impact on the design and operation of the embedded system. The common concerns in both hard and soft real time systems are quality of result and resource allocation. In case of a hard real time system, the quality of result is zero once the deadline expires whereas in the case of a soft real-time system the quality of output degrades more gracefully after the deadline expires. An effective embedded system requires a superior resource allocation strategy which tries to maximize the quality of result. This work specifically deals with systems which are characterized by imprecise computations or IRIS (Increased Reward with Increased Service) tasks. IRIS tasks are characterized by mandatory and optional portions. The system has option of terminating the optional portion early at the price of a less accurate output. The resolution of this trade-off between quality of result and allocated resources is the focus of this thesis.

A typical embedded system or a real time application in the real world is assumed to consist of logically separable computational tasks with precedence constraints, each of which has a specific functionality. The initial requirement of this work is to construct a task model which accommodates physically meaningful cost functions that reflect the quality of the result as a scalar value allowing optimization problems to be modeled with desired constraints. The reward accrued increases with increase in computation or service. Moreover this model also paves the way for an ideal blend

of soft and hard real time characteristics into a single unified frame-work in which the quality of the result can be expressed and optimized.

1.1   Examples of IRIS Tasks

IRIS tasks can be found for a wide variety of applications. Some of them are explained below.

Simulated Annealing is a heuristic search algorithm which can be used for generating optimal or near-optimal solutions for the well known symmetric traveling salesman problem [14]. A key control parameter is depreciation factor, which has a direct relationship with amount of time given to the algorithm versus the quality of results. The depreciation factor is a real value in the range (0,1) which is multiplicative factor for the cooling temperature during successive iterations of the algorithm. The higher the depreciation factor the slower the decrease in temperature, the greater the algorithm execution time. The depreciation factor for the temperature can be adjusted based on the time available for making decision offline. The lowerThis cooling schedule forms the basis of the the annealing, which enables the algorithm to get out of local minimas and progress towards global minimas. the depreciation the longer the search. Thus, at the start of SA, most worsening moves may be accepted, but at the end only improving ones are likely to be allowed. Accepted worsening moves can help the procedure jump out of a local minimum. The algorithm may be terminated after a specified number of jumps in temperature or at a given minimum temperature. The cost is an indication of the quality of result and the algorithm looks to minimize the cost. Better costs are achieved when the algorithm runs for a longer time, thereby validating the IRIS nature of this algorithm.

The quadratic assignment problem [9] is also an iterative task. Quadratic assignment is a basic optimization problem that generalizes TSP, clustering etc., The premise is set of n facilities and a set of n locations. For each pair of facilities a

weight or flow is specified (e.g., the amount of supplies transported between the two facilities) and for each pair of locations, a distance is specified. The problem is to assign all facilities to different locations with the goal of minimizing the sum of the distances multiplied by the corresponding flows. The input to the quadratic assignment problem consists of two $n \times n$ matrices $W = w_{(i,j)}$ (the weight or flow matrix between facilities) and $D = d_{(i,j)}$ (the distance matrix between locations) . Given matrices W,D and a permutation $\phi : n \rightarrow n$ the objective function is to minimize

$$Q(\phi) = \sum_{i,j \in n} w_{(i,j)} \cdot d_{\phi(i),\phi(j)} \tag{1.1}$$

In these algorithms the cost improves with increase in number of iterations of the algorithm but flattens out to a near-optimum value after a considerable number of iterations.

Some other applications which could use such a model include audio and video processing, multimedia data streaming, real-time image tracking, scalable multimedia processing and transmission, network traffic management, decision making under uncertainty, anytime learning in evolutionary robotics, motion planning and robot control and multi-target tracking. Multi core real-time scheduling algorithms may use the imprecise/IRIS model for grouping and scheduling a set of imprecise/IRIS tasks. In virtualized platforms, a group of IRIS applications may run as virtual programs on a single machine. In such a case, effective imprecise algorithms can be used by Hypervisor schedulers, such as Xen Hypervisor, in allocating the CPU resource to multiple virtual programs.

1.2 Examples of IRIS feeding IRIS

Consider the open source video codec tool, xvid [18]. This tool has a two-pass option for video encoding. The first pass analyzes the video clip; the second pass uses

the results of that analysis to obtain a high-quality encoding. Algorithm settings allow one to control the time spent in first-pass analysis; one can trade off the precision of the motion search against the time taken. The second pass takes the first pass results to efficiently encode the video clip. Controlling the allowed bitrate allows us here to trade off the quality against the computational work of this step. The quality of the first pass affects the range of possibilities for the second pass; the quality of both passes depends on the length of time devoted to them.

A second example is path planning in robotics [21]. Path planning includes sensing and planning modules, both of which have the IRIS property. The sensing module builds up an awareness of the environment; this is then used by the planning module to complete path planning.

A third example is developing control inputs for cyber-physical systems. Suppose a linear control system has multiple control variables. One approach is to calculate these variables one at a time in order of their perceived impact on the quality of control provided; when control input k is calculated, the values of control inputs $1, \cdots, k-1$ are already available. Depending on the amount of time available, we may only calculate the first N control inputs, leaving the others at 0. Gupta has shown this to be a viable strategy in an environment where the amount of time available for computation is variable [7].

Our final example is the task structure for the control of a planetary rover [22]. The task is composed of a sequence of processing levels l_i and each level contains alternative modules m_i^1 , m_i^2 , $\cdots$. Each alternative module has a different resource requirement in return for which it provides a certain quality output. By selecting the modules appropriately, we can trade off the quality of control provided against the resources (e.g., time) consumed.

CHAPTER 2

PREVIOUS WORK

2.1   Models and Heuristic Scheduling algorithms

Existing models for imprecise computation can be mainly classified under the following divisions - the basic imprecise computation model, the extended imprecise computation model, IRIS model and anytime algorithms. All these motives deal with splitting the task into *mandatory* and the *optional* parts. It is necessary that the mandatory part is complete in order to obtain a meaningful or lowest quality acceptable result while the optional part enhances the quality of the result produced by the mandatory part. The optional part can be prematurely terminated or omitted in its entirety as per user discretion. Most work on scheduling IRIS tasks has focused on independent tasks. Also, the common assumption is that all tasks run up to their estimated worst-case execution times.

The basic imprecise computation model by Lin *et al.* [11] introduces this kind of simple logical segregation of the task. The quality of result is denoted by the amount of error produced by the task, which depends on the unexecuted fraction of the optional part of the task. This error decreases as more and more of the optional part is executed and reaches zero when the optional part is completed. The IRIS (Increased Reward Increased Service) Model [3] is similar to the basic model except that the metrics for quality measurement is given by reward accrued by the tasks. The reward accrued is higher when the error is small and vice versa.

The computation models adopted in anytime algorithms are a group of algorithms [20] that can return a meaningful result at any time and are developed for real-time

artificial intelligence problems which allow computation to be terminated prematurely. They differ from other models in the fact that the mandatory portion requirement of the tasks are almost negligible compared to the optional portion. In many cases the mandatory portion is just a preliminary assignment of a group of parameters used in the computation.

Chung, *et al.* consider periodic task sets running on multiprocessors [1]; the task set is known ahead of time and a schedule can be set up offline. A first-fit approach is taken to allocating tasks to processors; following this, uniprocessor scheduling is carried out on each processor. The Rate Monotonic (RM) algorithm [13] is used to assign static priorities to the mandatory portions of each task based. The optional portions of all tasks have lower priority than the mandatory portion of any task. Various simple heuristics have been studied for scheduling the optional portions, including static priorities inversely related to the task utilization and dynamic priorities favoring the optional portion with the least execution time provided or the one with the least slack time. It is assumed that the error associated with premature termination of an optional portion is proportional to some positive power of the fraction of uncompleted work.

An online approach is discussed in Shih and Liu [17]. The workload consists of a set of tasks known ahead of time together with tasks that arrive during system operation. The error model is linear, the output error being equal to the amount of unfinished work. As tasks arrive, time is reserved for their mandatory portions using the latest-ready-time-first order. Optional tasks can execute as long as there is enough time.

Dey *et al.* presented three heuristic scheduling algorithms for online scheduling of aperiodic workloads [4] [2]. Their reward function is a concave non-decreasing function of the execution time. Two of the algorithms take a two-level approach. The top level is executed whenever a new task arrives and is responsible for deciding

the allocation of service time to that task such that the reward is maximized. The lower-level algorithms decide the order in which tasks execute. Their third algorithm takes a greedy approach. The two metrics used for evaluating performance are the reward rate and average number of task preemptions using each scheduling policy. They have developed an analytical model for an IRIS task system and obtained the upper-bounds on the reward-rate that is achievable by any scheduling policy adopted. This work concludes that with the appropriate lower-level scheduling policy, the performance of their algorithm approaches quite close to its upper bound. The average number of preemptions is very small when the Earliest Deadline First (EDF) scheduling algorithm is used at the lower level.

Mej a-Alvarez *et al.* [23] presented the INCA server that incrementally searches within a set of feasible solutions to maximize reward or value. Every task is assigned a criticality value and consists of a mandatory part and an optional part. The aim is to execute the most critical tasks in the system so that the total value of the system output is maximized. In the case of overload, the first move of the INCA server is to disable some optional parts to eliminate the overload. Selecting the optional parts to discard involves searching through a number of combinations and is time consuming. Hence in such cases the INCA server iteratively executes a quick approximate online algorithm to select a set of optional parts to execute to maximize the value. Hence further iterations of the approximate algorithm refine the quality of the initial solution.

A very good analysis of the independent task set problem is presented in [17], where every independent task follows the mandatory/optional model and the aim is to minimize the total error incurred by all the tasks. This work takes into account three distinct scenarios based on the presence of offline tasks (ones that arrive before the processor starts execution) and the ready time of the arriving tasks. They have developed three algorithms one for each case mentioned above to minimize total error

incurred. Their algorithm describes various events that can occur when a task is executing online and appropriate handlers for each of those events such that the objective of minimizing the total error is achieved.

A hierarchical approach to scheduling is taken by Tchamgoue, *et al.* [19]. The overall workload is divided into components; each component is guaranteed to obtain a certain minimum amount of resources over every specified period. Each component can then be scheduled with this guarantee in mind. A hierarchical approach allows the scheduling of one component to be decoupled from the scheduling of another.

The above mentioned works all deal with independent tasks. By contrast, Feng and Liu consider *composite tasks*, each of which consists of linearly dependent tasks [5,6]. That is, each task (except for the first and last) in a composite task has exactly one parent and one child; a task receives input from its parent, carries out some processing, and then forwards the output to its child. The first task receives inputs from the application; the final task produces output to the application. The quality of output of a task depends both on the quality of its input as well as on the amount of time it executes for. An interesting assumption is that inaccuracies in the input can cause the mandatory and optional portions to require more time to execute.

Feng and Liu introduce a two-level scheduler. The first level schedules the composite tasks using a modified EDF approach which treats the entire composite task as optional and cuts off tasks at the deadline, even if they have not been given their full execution time. If it manages to find full execution time for each composite task, we are done. If not, it augments the execution time allocation to composite tasks with relatively small optional parts. In the second level, the time allocated for each composite task at the first level is distributed to its subtasks such that the output error of the composite task is minimized. They have developed and compared the performance of five second-level heuristic scheduling algorithms.

8

2.2 Implementation of IRIS type tasks

Lin *et al.* proposed three kinds of practical implementations of imprecise/IRIS computations [11]. The main motivation behind the milestone approach is taking a backup of the imprecise output in systems with the assumption that the output from the system increases monotonically in correctness as it progresses towards completion. Therefore, the longer a procedure executes, the more accurate is the result. They argue the correctness of a system can be more correctly represented by a staircase function where each step indicates completion of a intermediate phase. The assumptions behind the sieve approach is that the output corresponds in number and type to the inputs and sieves are functions which refine the inputs in terms of correctness. The sieves perform computation in order to increase precision, and hence choosing not to execute them leads to imprecision but faster execution. This method is based on using a staircase function where execution of a sieve corresponds to a step in the staircase. The third method is a multi-version method in which each version delivers a different quality result. The version that has to be executed is selected based on which one would give the best results under the given deadline constraints. Lin *et al.* [12] also proposes the Concord system which is a client server structure depicting imprecise computation based on the milestone approach. An imprecise computation is started on the server side when the client requests one to be started. A supervisor is a handler on the server side which is responsible for saving the intermediate results of the computation. If the server completes its execution before the deadline, then the precise result is sent back to the client via the supervisor. Otherwise, the computation is terminated on deadline and the imprecise result is returned to the client.

Marty and Stankovic [8] developed a kernel thread package for the Spring real time system. The thread package allows safe premature termination of computations by killing threads at logical boundaries such that the state of the thread is determinate.

A request/release pair provides for mutual exclusion which facilitates safeguarding the thread from being killed when a computation is in progress. This is quite similar to cancellation points in POSIX threads.

CHAPTER 3

MODEL AND PROBLEM STATEMENT

3.1 Task Model

We are given a task precedence graph (TPG) indicating the dependence between tasks. This may well consist of multiple connected components. A task is assumed to require inputs from all its parents before it starts executing; it delivers output only at the end of its execution. The quality of the result of any task has a noticeable effect on the quality of result of its dependent tasks. Hence the factors that decide the quality of result of a task are the cost function associated with the task, the amount of service it gets, the quality of result(s) that is passed on to it and its sensitivity to that quality. This is modelled as follows.If $\vec{\sigma}_i$ denotes the vector of inputs to task T_i and ϕ_i the fraction of its optional portion that has been executed, its output error is given by $E_i(\vec{\sigma}_i, \phi_i)$. As a practical matter, unless we instrument the code to monitor and output the progress of the execution, ϕ_i is never known exactly except when the optional portion finishes, i.e., when $\phi_i = 1$. At all other times, we must use our best estimate of this value based on profiling and on the number of cycles consumed so far in its execution.

3.2 Processor Model

The IRIS workload runs on a set of processors which use dynamic voltage scaling [16] to trade off clock frequency (and hence rate of execution progress) and energy consumed. In this work, we assume that there are two discrete voltage levels, V_{high} and V_{low}. It is quite easy to extend this algorithm to account for a larger number of

Notation	Explanation
d_i	Deadline of Leaf Task i
FT_i	Finish time of Leaf Task i
E_b	Energy bound for the TPG
c_i^{low}	Number of low voltage cycles spent executing Task i
c_i^{high}	Number of high voltage cycles spent executing Task i
c_i^{wm}	Mandatory worst case cycles of Task i
c_i^{wo}	Optional worst case cycles of Task i
e_{high}	Energy consumed by one high voltage cycle
e_{low}	Energy consumed by one high voltage cycle
ν^{low}	Step size at low voltage
ν^{high}	Step size at high voltage
P_{swap}	Swap Probability
χ	A mapping of tasks to processors $(1..n) \rightarrow (1..m)$
Λ	A set $\{c_{high}^i\}$ where $i \in \{1..n\}$
Ω	A set $\{c_{low}^i\}$ where $i \in \{1..n\}$
Π	A schedule given by three tuple $\langle \chi, \Lambda, \Omega \rangle$
Δ_{online}	Online time granularity
$\vec{\sigma}_i$	Input Vector to Task i
$E_i(\cdot)$	Output Error function of Task i
Γ	Final error of task graph
$F(\cdot)$	Recursive application of $E_i(\cdot)$

Table 3.1. Some Notations

voltage levels; however, with maximum supply voltages dropping every semiconductor generation, it is increasingly unlikely that more than two voltage levels will be useful. We assume that voltage switching costs are negligible: given that each task undergoes at most one voltage switch in our algorithm, this is a reasonable assumption. Moreover the overhead of voltage switching is typically a few tens of microseconds, which is very small in comparison to the execution time of complex control algorithms and the task periods in cyber-physical systems. A detailed discussion on the time overhead models can be found in [15]. The processor consumes e_{high} and e_{low} energies per clock cycle at V_{high} and V_{low}, respectively; the corresponding frequencies are f_{high} and f_{low}. The energy spent in communication is folded into the cost of execution and is not accounted for separately.

3.3 Optimization Objective and Constraints

The only output that is visible to the application is that from the leaves of the TPG. Denote by $\mathcal{L}$ the set of leaves of the TPG, by d_i, FT_i the deadline and finishing time of leaf task T_i respectively, and by c_j^{low}, c_j^{high} the number of low-voltage and high-voltage clock cycles spent executing *any* T_j. Number the tasks from 1 to n; let E_b be the upper bound of the energy consumption (set it to ∞ if no such bound exists).

The overall optimization problem is to minimize the weighted sum of the leaf errors:

$$\Gamma = \sum_{i \in \mathcal{L}} \kappa_i E_i \left(\vec{\sigma}_i, \phi_i \right) \tag{3.1}$$

subject to the following constraints for all $j \in \mathcal{L}$ and $i \in \{1, \cdots, n\}$:

$$FT_j \leq d_j \tag{3.2}$$

$$\sum_{i=1}^{n} \left(c_i^{high} \cdot e_{high} + c_i^{low} \cdot e_{low} \right) \leq E_b \tag{3.3}$$

where κ_j is the weight given to the error in the output of leaf task T_j, and reflects the scale of values of the application.

There are two sources for task input: the external world and other tasks. We assume that the error from the external world input is zero. It is not difficult to relax this assumption to account for say sensor errors. This can be accomplished by adding another set of variables which can account for them. With this assumption and applying the error function E_i recursively, we can write the overall error as some function of the number of clock cycles consumed by each task. That is, if $c_i = c_i^{low} + c_i^{high}$ is the number of clock cycles consumed by T_i, we can write

$$\Gamma = F(c_1, c_2, \cdots, c_n) \tag{3.4}$$

where $F(\cdot)$ can be obtained by recursive application of the $E_i(\cdot)$ functions.

13

Figure 3.1. Task Graph Example

As a simple example, consider the task graph shown in Figure 3.1. Tasks T_2 and T_3 receive inputs from T_1, T_4 receives inputs from both T_2 and T_3. We wish to derive $F(\cdot)$ from the error functions, $E_i(\cdot, \cdot)$, $i = 1, \cdots, 4$. Based on our profiling of these tasks, suppose our best estimate of the mandatory and optional cycles used by these tasks are given by μ_i, ω_i respectively for $i = 1, \cdots, 4$. Therefore, if c_i is the number of cycles allocated to task T_i, our best estimate of the fraction of the optional portions completed is given by $\phi_i = \max\left\{0, \frac{c_i - \mu_i}{\omega_i}\right\}$.

Hence, we can write

$$\begin{aligned} \vec{\sigma}_2 &= \vec{\sigma}_3 = (E_1(0, \phi_1)) \\ \vec{\sigma}_4 &= (E_2(\sigma_2, \phi_2), E_3(\sigma_3, \phi_3)) \end{aligned}$$

Hence, the output error, which is the error in the T_4 output is given by

$$E_4(\vec{\sigma}_4, \phi_4).$$

14

Based on the above expressions, we can obviously express E_4 in terms of c_i, $i = 1, \cdots, 4$.

The problem is complicated by the fact that, as mentioned above, the actual total number of execution cycles required to finish a task is not known precisely (except when the task finishes). At best, we only know its probability distribution based on workload profiling. We therefore have to use an *estimate* of ϕ_i as a function of c_i, based on the information available. We do know the worst-case cycles, c_i^{wm}, c_i^{wo}, required for the mandatory and optional portions, respectively, of each task T_i.

CHAPTER 4

SCHEDULING ALGORITHMS

4.1   Two Level Scheduling

The scheduling heuristic developed for this task model is divided into two stages offline and online. Offline scheduling involves schedule-building that is performed before the system is put into operation, while online scheduling is that performed during system operation . The offline algorithm uses heuristic search techniques to search through a finite solution space of configurations with a view to minimizing the final error of the TPG for the given deadline and energy constraints. The offline algorithm makes sure that each task in the TPG receives the mandatory worst case requirement which guarantees that the mandatory portion of the task is always completed. When the deadlines are so tight that the mandatory worst case cannot be provided to a task , the algorithm keeps searching through different configurations to find a valid schedule within the time allocated for the scheduler to make a decision. Since offline decisions are based on the worst case assumptions regarding their execution cycles, the tasks sometimes finish earlier and does not use all the allcoated cycles. The online algorithm distributes such released cycles to other tasks in the system with a view to minimize the final error of the TPG without violating the end-to-end deadline constraints.

4.2   Offline Allocation and Scheduling Heuristic

It is not practical to obtain an algorithm to optimize Γ in Equation 3.1. To begin with, we do not have perfect information as to ϕ_i. Even if we had some oracle to

Figure 4.1. Offline Heuristic.

accurately divine the value of ϕ_i during execution, this would still be an NP-complete problem. We must therefore satisfy ourselves with a heuristic.

Our heuristic exploits the fact that in cyber-physical systems (our target application area), the computational tasks are known in advance, and can be profiled extensively before the system starts operation. Such advance information can be exploited by having separate offline and online phases in the scheduling process. In the offline phase, tasks are assigned to processors and a schedule is generated making assumptions about the tasks' running time. In the online phase, as tasks finish, we update our knowledge of their actual running time and reclaim whatever resources are released by early task completion. For obvious reasons, the online heuristic must be lightweight.

Our algorithm has the high-level structure shown in Figure 4.1. We start with a candidate allocation of tasks to processors. This allocation is assessed for its ability to meet time and energy constraints as will be described later. Simulated annealing is used to navigate through various allocations in a search to find one which offers good performance.

17

4.2.1 Simulated Annealing Module

The basic elements of the simulated annealing module are:

1. A finite space,S, of all possible configurations, where each configuration is a mapping of the entire task set to processor set.

2. A step function STEP() which returns a configuration after moving a random task from one processor to another or swaps two random tasks on two different processors based on P_{swap} which indicates the probability with which two tasks assigned to different processors are swapped. The higher the value of P_{swap}, the grater the chances of tasks getting exchanged between different processors.

3. A cooling schedule with an initial temperature $Temp_{initial}$ and a final temperature $Temp_{final}$, a depreciation factor d_f and N_{tries} a limit to the number of tries of the greedy algorithm at each temperature value.

4. An acceptance criterion which states that every new configuration is accepted with probability p , where p is based on δ the difference between the new final error and the best final error, k the Boltzmann constant and T is the current temperature.

5. The depreciation factor d_f for the temperature can be adjusted based on the time available for making the scheduling decision offline. The lesser the depreciation the the better the chance of finding better configurations.

6. An arbitrarily generated initial configuration $\chi_{initial}$ with a random mapping of {1..n}→{1..m}.

In what follows, δ_{ij} is the Kronecker delta, i.e., $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. The worst-case mandatory and optional execution cycles of task T_i are denoted by c_i^{wm} and c_i^{wo}, respectively. We assign cycles to tasks in steps where necessary: the step size at high and low voltage levels is denoted by ν^{high}, ν^{low}, respectively. These are chosen so as to take the same time, i.e., such that $\nu^{high} \cdot f_{high} = \nu^{low} \cdot f_{low}$.

```
Module SIMULATED-ANNEALING
Input  :  Temp_initial , Temp_final , d_f , N_tries , T_b , χ_initial
Output  :  Π_final
Begin
     temp = Temp_initial;
     χ = χ_initial ;
     Γ_offline^final = ∞;
     Π_final = INVALID;
     χ_final = INVALID;
     while (temp > Temp_final) do
          for i in (1 ..  N_tries)
               χ_new = STEP(χ);
               Π= GREEDYALLOC(χ_new);
               if (Π is valid)
                   δ = F(Π) - Γ_offline^final
                   if (δ < 0) OR (RANDOM(0,1) > e^(-δ/(k×temp)))
                        χ = χ_new;
                        if ( F(Π) < Γ_offline^final)
                             Γ_offline^final = F(Π)
                             Π_final = Π
                             χ_final = χ
          end for
     temp = temp /  d_f;
     end while
     return  Π_final
End.
```

```
Module GREEDYALLOC
Input :  Configuration χ
Output :  A valid Π or INVALID indication


Π =TIMEBOUND(χ)
If (Π! = INVALID)  {
        If ($\sum_{i=1}^{n} c_i^{high} \cdot e_{high} \leq E_b$)
                return Π
        else
                return ENERGYBOUND(Π)
}
return Π = INVALID
```

4.2.2 Greedy Allocator Module

The greedy allocator returns a schedule based on which one can estimate the offline final error $\Gamma_{offline}$, for the specified task assignment. The schedule is marked INVALID if it is unable to find one which does not satisfy the deadline and energy constraints. It first generates a time allocation taking only the deadlines into account and disregarding the energy bound, if any. If an energy bound is specified, it then modifies this schedule by swapping high-voltage and low-voltage cycles if this is needed to meet the bound. If no such feasible swap can be found, it declares failure and returns an INVALID result.

4.2.3 Time Bound Module

The time bound module generates a static offline schedule for the given configuration. The input to the algorithm is a configuration χ passed in by the SA module. The algorithm starts by assigning high cycles to meet the worst case mandatory requirement of all tasks. Next a check is done to analyze whether the schedule generated after this step violates the deadline. If this happens then the search heuristic is informed that this is an invalid configuration. If the deadline is not violated then the algorithm proceeds with the allocation of high cycles for the optional part of all tasks. The allocation is given to tasks in slices of ν^{high}. The task which gives the maximum improvement in final error at that instant is given the slice. Ties are broken arbitrarily. If by allocating the slice to the task the path on which it is placed becomes critical (the TPG violates end-to-end deadline) or if it exceeds total worst case requirement of the task, then the allocation is retracted and the task is marked for denying any allocations in the future. t_{af}^i is a flag used for indicating this. If t_{af}^i is zero then the task becomes unallocatable. This allocation continues until all the tasks are marked as unallocatable, at which point the valid schedule is returned to the greedy allocator module.

```
Module TIMEBOUND
Input :  Configuration χ
Output :  A valid Π or INVALID indication

Π = INVALID;
c_i = c_i^wm,   i = 1,···,n .
If a deadline is violated,
  return  Π = INVALID.
else
  Assign t_af^i = 1 for i = 1,···,n.

while (∃i s.t. t_af^i == 1):     {
  for each such i
        for each j ∈ {i,···,n}  c_j' = c_j + δ_ij ν^high
            Calculate B_i = F(c_1,···,c_n) − F(c_1',···,c_n')

Define i_max = arg max_{1≤i≤n} B_i .
Set  c_{i_max} + = ν^high

If a deadline is missed
  set t_af^{i_max} = 0
  revert allocation c_{i_max} − = ν^high
else
  update Π
}
Return Π
```

4.2.4 Energy Bound Module

The energy bound phase starts after the time bound phase arrives at a valid schedule with respect to deadline constraints. The offline energy bound phase starts by assigning low voltage cycles to all the tasks in the time frame allocated by the time-bound phase. Then it makes sure that all the tasks have enough cycles to satisfy their required worst case mandatory workload by converting low voltage cycles to high voltage cycles. Now after this stage, if the schedule has violated the energy deadline then low cycles are removed from the tasks which least affect the final error without violating their worst case mandatory work load requirement. If the algorithm runs out of tasks to remove low cycles and the energy deadline is still violated, the user is notified of this failure. If we are still under the energy bound, after completing the mandatory workload of the task, the low cycles of the tasks are converted into high cycles greedily until the energy barrier is hit or we run out of low cycles. When this condition is reached a valid schedule is returned.

```
Module ENERGYBOUND
Input :  Configuration χ
Output :  A valid Π or INVALID indication
```

1) $c_i^{\text{low}} = \lfloor c_i^{\text{high}} \cdot f_{\text{low}}/f_{\text{high}} \rfloor \quad c_i^{\text{high}} = 0$

2) for each $i \in 1, \cdots, n\}$

    while $(c_i < c_i^{\text{wm}})$

        $c_i^{\text{low}} - = \nu^{\text{low}}$

        $c_i^{\text{high}} + = \nu^{\text{high}}$

3) Calculate energy consumed, E_c.

4) while $(E_c > E_b)$

    $\text{task}_{\text{found}}^{\text{low}} = \text{FALSE}$

    for each $i \in \{1, \cdots, n\}$

        $\Delta_i = \infty$

        if $(c_i \geq c_i^{\text{wm}} + \nu^{\text{low}})$

            $\text{task}_{\text{found}}^{\text{low}} = \text{TRUE}$

            for each $j \in \{i, \cdots, n\} \quad c_j' = c_j - \delta_{ij}\nu^{\text{low}}$

            $\Delta_i = F(c_1', \cdots, c_n') - F(c_1, \cdots, c_n)$

    if $(\text{task}_{\text{found}}^{\text{low}} == \text{FALSE})$

        return INVALID

    else

        Find $i_{\min} = \min \arg_{1 \leq i \leq n}\Delta_i$

        $c_{i_{\min}}^{\text{low}} - = \nu^{\text{low}}$

        Recalculate E_c

5) while $(E_c < E_b)$

    for each $i \in \{1, \cdots, n\}$

        for each $j \in \{i, \cdots, n\} \quad c_j' = c_j + \delta_{ij}(\nu^{\text{high}} - \nu^{\text{low}})$

            $B_i = F(c_1, \cdots, c_n) - F(c_1', \cdots, c_n')$

    if $\forall i \in \{1, \cdots, n\}, \ c_i^{\text{low}} < \nu^{\text{low}}$

        return schedule Π

    if $\forall i \in \{1, \cdots, n\}, \ c_i \geq c_i^{\text{wo}} + c_i^{\text{wm}}$

        return schedule Π

    else

        Define $i_{\max} = \arg \max_{1 \leq i \leq n}B_i$.

        $c_{i_{\max}}^{\text{high}} + = \nu^{\text{high}}$

        $c_{i_{\max}}^{\text{low}} - = \nu^{\text{low}}$

        Recalculate E_c

4.3 Online Algorithm

As mentioned earlier, the actual execution times vary from one execution instance to another. The actual demand of a task is not known unless and until the task completes execution. At this point, we know that the entire optional part has been executed. Once task T_i completes execution, we know that $\phi_i = 1$, meaning that the T_i output error will be given by $E_i(\vec{\sigma}_i, 1)$. This then affects all tasks that are downstream from it and allows the error function $F(\cdot)$ to be updated appropriately. Also, if a task completes before its assigned time has been spent, additional time is released for other tasks to use. The job of the online algorithm is to reclaim this released time to improve on the offline schedule.

The algorithm makes sure that the tasks do not exceed their static finish times assigned by the offline algorithm while distributing the energy, thereby respecting the global deadline. The two parameters to control the amount of time the online scheduler has for distributing the released energy are the granularity of allocation Δ_{online} and the set of tasks considered for distribution. The lesser the granularity the more the time for calculating benefit for the tasks and hence more time will be taken for distributing the released energy.

The input parameter t_{level} controls the set of tasks considered for energy distribution when a task finishes: it can be regarded as a means to limit lookahead in an effort to reduce the algorithm overhead. We only consider tasks which are t_{level} levels away from T_f in the task graph. $depth(T_i)$ gives the shortest distance of task T_i from the root of TPG and $oncriticalpath(T_i)$ returns true if the allocation of additional energy to the task violates the deadline or energy constraint, $finished(T_i)$ returns true if the entire optional portion has finished.

```
Module ONLINE
Inputs:  Δ_online, t_level.
1)Calculate, t_reclaimed, the time reclaimed upon task T_f completion.
2) If (t_reclaimed == 0)
     return
else while (t_reclaimed > 0) {
3)   n_low^temp = t_reclaimed · f_low
4)   n_slice = Δ_online · f_low
5)   If n_low^temp < Δ_online return
6)   n_allocated = 0
7)   Identify task set OTS of tasks T_x such that
         depth(T_x) − depth(T_f) ≤ t_level
         finished(T_x) = FALSE
         T_x is not on a critical path to a leaf
         If OTS is empty, return
8)   for T_i ∈ {OTS}
         if (c_i + n_slice > c_k^wo + c_k^wm) remove T_i from OTS
         if OTS is empty, return
9)   Assign n_slice cycles at V_low to the task T_k
     in OTS which yields the greatest improvement in error:
     c_k = c_k + n_slice
     t_reclaimed− = Δ_online
10) If T_k now finishes later than in the offline schedule,
     reverse this:  {
         c_k− = n_slice
         Remove T_k from OTS
         t_reclaimed+ = Δ_online
     }
}
```

CHAPTER 5

SIMULATOR DETAILS

The simulator developed for this work The simulator environment for this work consists of the following major parts and some of their salient features are described below:

1. A task precedence graph generator

2. An offline and online scheduler

3. Result Analyzer

5.1   Task Precedence Graph Generator

The graph generator module generates task precedence graphs of varied characteristics for the Scheduler module to work on. The generated graphs are in the form of an adjacency matrix for the scheduler module to read. Apart from generating the TPGs this module also generates the worst case execution times of the mandatory and optional part of the tasks. It generates the online characteristics of the task. It generates suitable mean and standard deviation for the task run times based on the worst-case execution times. It generates the initial configuration for the TPGs i.e a mapping of the tasks to the processors which is used as the starting point for the search heuristics in the scheduler module. This module is also responsible for finding the depth of the TPGs that it generates; It also generates the communication cost that is accrued by the schedule for dependent tasks executing on different processors.

5.1.1   Inputs

1. Number of processors

2. Number of Tasks

3. Number of TPG

4. The maximum out degree of the tasks

5. A probability parameter for deciding the edge between two tasks before maximum out degree is reached for the parent task

6. A specification for the minimum run time of the tasks

5.1.2   Outputs

:

1. Adjacency matrix indicating the dependency relation among tasks

2. Communication matrix indicating the communication cost accrued by tasks

3. Sensitivity of each task to its parent task(s)

4. The depth of each TPG

5. An initial configuration for the search heuristics in the scheduler module

6. Worst case Mandatory and optional execution times for the tasks

7. Mean and deviation for the run times of the tasks describing the online characteristics

5.1.3 Generator

5.1.3.1 Adjacency matrix

Random TPGs are generated such that the maximum out degree and the edges are assigned as per the probability specified before the maximum out degree is reached. A separate module checks for the connected components and independent tasks of the TPG and connects them to a fictitious root node with null execution time and zero communication cost. This module also checks for cycles present in the graph and throws away the TPG if a cycle is found to be present in it. The generated TPG's are in the form of an adjacency matrix and fed to the scheduler through a temporary file.

5.1.3.2 Depth

The maximum depth of each TPG is calculated and written into a separate file for further numerical analysis.

5.1.3.3 Initial Configuration

A mapping of tasks to processors is also generated such that the different search heuristic techniques can start their search at this configuration. Given the common starting point, this allows a chance for heuristics to be fairly compared to each other.

5.2 Offline and Online Scheduler

The scheduler module uses the output of the generator module and applies the offline and online heuristics to extract the offline and online errors for the TPG's under the specified constraint.The scheduler starts by reading the adjacency matrix from a file output by the generator. It generates the communication cost and the sensitivity matrix for the tasks in the TPG. It finds out the connected components of the TPG and uses a fictitious root to connect all identified components of the TPG. The communication cost, sensitivity and execution times of the fictitious root are null.

The next step is to find the depth of the nodes and assign an initial configuration for the heuristic search. It then generates the distribution of the run times of the tasks based on the generator's input. The offline algorithm is then executed on the initial configuration to obtain an acceptable and error minimized configuration for the current TPG. The result of the offline stage is written into a file for temporary analysis. Then the online algorithm is executed based on the generated run times and and the final online error is written into a file. This whole process is repeated for successive TPGs.

5.3 Result Analyzer

The result analyzer reads through the files output by the scheduler and uses the information in them to calculate the average, confidence interval of the results of the current run of the scheduler.

5.4 Usage

The implementation is divided into three separate c files:

1. rtsched_gen.c [Generates the required files]

2. rtsched_sa.c [Scheduler Implementation]

3. read_online.c [Result Analyser]

5.4.1 Compiling

The standarad GNU Scientific library (GSL) is a requirement for the simulator. This is a well known liibrary and can be installed from http://www.gnu.org/software/gsl. It should be linked and used as follows

gcc -std=c99 rtsched_gen.c -o rtsched_gen -lgsl -lgslcblas -lm

gcc -std=c99 rtsched_sa.c -o rtsched_sa -lgsl -lgslcblas -lm

gcc -std=c99 read_online.c -o read_online

5.4.2   Main Parameters

1. #define NUMBER_OF_TASKS (Number of tasks)

2. #define NUMBER_OF_PROCESSORS (Number of Processors)

3. #define NUMBER_OF_GENERATIONS (Total number of TPG's)

4. #define MAX_OUT_DEGREE (Maximum Outdegree)

5. #define EDGE_PROBABILITY (Edge Probability)

6. #define MIN_RUN_TIME (Minimum run time fraction)

7. #define DEVIATION_FACTOR (Deviation factor)

8. #define ALLOC_TIME (Allocation Granularity)

9. #define N_TRIES (Number of tries)

10. #define ITERS_FIXED_T (Iterations at each temperature)

11. #define K (Boltzmann constant)

12. #define T_INITIAL (initial temperature)

13. #define MU_T 1.1 (damping factor for temperature)

14. #define T_MIN 0.01 (Final Temperature)

CHAPTER 6

NUMERICAL RESULTS

6.1 Experimental Setup

6.1.1 Task Graph modeling

Our numerical results are based on simulating 1000 random directed acyclic directed TPGs, each of which was run 500 times with different random on-line runtimes. Each TPG was generated based on an Edge Probability P, $P \in (0,1)$, which specifies the probability of an edge between two nodes in the TPG, and a Maximum Out Degree D specifying the maximum number of children a node can have. Low values of P and D will generate leaner TPGs with less dependencies, and vice versa. The worst case mandatory and optional parts of each task were selected at random out of {5,10,15}. The deadline for each TPG was selected as no lower than the sum of the worst case mandatory parts of the longest directional path in the graph. During allocation of time or energy to tasks, a critical path violation (a path in the TPG which violates the time deadline) is identified using standard algorithms mentioned in [10]. When applying the offline Simulated Annealing algorithm, we performed 90 iterations of the algorithm before starting the online phase (unless stated otherwise). This value was chosen as there was not much improvement in quality after this for the TPGs considered. Any other special settings for the experimental results are mentioned in the subsequent sections.

6.1.2 Error Functions and Sensitivity values

We assume that the error generated by an incomplete task is a convex function of the fraction of the uncompleted optional part out of the total optional part. We

33

used as error function the function x^8 unless stated otherwise. In addition, each task has sensitivity values, which denote the sensitivity of its output error to its input errors. We selected these sensitivities at random for each task out of the interval $(0, 2.0]$. A high sensitivity value will lead to high increases in output error for small input errors and vice versa. A linear error propagation model is assumed for all the experiments conducted; the output error is convex with respect to the fraction of unexecuted optional part whereas it is linear with respect to the input errors.

6.1.3 Run Time Requirements modeling

The run time characteristics of the tasks are modeled as follows. The actual run time follows the Normal distribution, conditioned on falling between specified minimum and maximum values, with the mean midway between them. The minimum value is given by a fraction (mf) of the worst case requirement whereas the maximum is the worst case itself: $t_{man}^i \in \{[\text{mf}, 1.0] * t_{wm}^i\}$ and $t_{opt}^i \in \{[\text{mf}, 1.0] * t_{wo}^i\}$. The on-line phase is sampled 500 times and the average of 1000 successfully scheduled TPGs is used for analysis.

Based on the above described setup, we performed several experiments in order to determine the effect of some key parameters on the output quality of the task graphs

Figure 6.1. Effect of Outdegree

6.2   Effect of OutDegree

Figure 6.1 shows the average online error against edge probability P which is the probability with which a task is connected to another task in the system before the max out degree is reached. Each point on this graph is the average online error for 1000 successfully scheduled TPGs, each TPG sampled 500 times for online error with random run times for tasks. This graph shows that as the maximum out degree allowed on the TPGs get higher the dependencies in the TPGs increase and the task's input error increases eventually driving the final error higher.

Figure 6.2. Effect of Minimum Run Time

6.3 Effect of Minimum Run times

Figure 6.2 shows the average online error as a function of r_{min} the ratio of the minimum run time to the worst case run time of tasks. The different curves pertain to different values of edge probability P. As the value of P increases, the dependency between the tasks increases. From the error model it can be noted that both precedence and quality dependency increases as P increases. This results in the sharp rise of curves pertaining to higher P value. Very high values of P result in too many TPGs with cycles and cannot produce meaningful results.

Figure 6.3. Effect of Standard Deviation of Run Times

6.4 Effect of Standard Deviation

Figure 6.3 shows the effect of varying the standard deviation of the run time distribution with different time deadlines. The standard deviation decreases from left to right along the x-axis. As the coefficient of variation goes up, the offline algorithm has less information about the actual execution times of the tasks. This leads to an increase in error, as can be seen in the figure.

Figure 6.4. Effect of Online Reclamation

6.5 Effect of Online Reclamation

Figure 6.4 shows the effect of online reclamation. The plot shows the ratio of the average online error with reclamation to the average online error without reclamation. We first observe that for a very tight time deadline, not much reclamation is done even for increasing energy constraints. This can be owed to the fact that cycles released by tasks are not effectively used by other tasks, because doing so would violate the TPG's time deadline. Second, for medium time deadlines, reclamation is more effective for relatively tighter energy constraints than for loose energy constraint. This is due to the fact that for stricter energy constraints even a small amount of energy released can be distributed much more effectively than for looser energy constraints. Thirdly, for very relaxed time deadlines, there were not many opportunities to reclaim as tasks were allocated with ample resources.

Figure 6.5. Greedy Vs BFS Allocation

6.6 Our Allocation Vs BFA Allocation

Figure 6.5 compares the performance of our algorithm to a Breadth First Allocation (BFA) algorithm. The BFA offline and online algorithms have no knowledge of the error functions associated with the tasks. This algorithm prioritizes the tasks based on their appearance in a Breadth First Search and allocates time to them as per their priorities in a round robin fashion while taking care not to violate the time deadline. The plot shows the ratio of the final average error of our algorithm to BFA as a function of the time deadline, for three values of the number of iterations of the offline algorithm, which depends on the time allotted to the offline scheduler. Our algorithm performs much better than BFA when the scheduler has less time and a very tight deadline for the TPG. As expected, our algorithm beats BFA by larger margins as the deadline gets loose. This is mainly because the optional part for the tasks is allocated more wisely based on the benefit in final error.

Figure 6.6. Effect of Step Error Function

6.7 Effect of Different Error Functions

Figure 6.6 compares the average error for different error functions. We used a convex error function $f(x) = x^2$ and a series of step functions $f(x, steps) = (\lfloor (x * steps) \rfloor)/steps$ (where x is the fraction of the unexecuted/unallocated optional part and $steps$ is a power of 10). As the number of steps increases the step error function behaves much like the convex error function but is bounded below by it. The plot shows that for larger step sizes (indicating a more abrupt but less frequent change in error value), the output error increases.

Figure 6.7. Anytime Robot [21]

6.8   A Real World Application

This experiment was conducted on a real world model of a robot implementing anytime sensing, planning and action shown in Figure 6.7 [21]. Our analysis concentrates on determining the resource allocation for each of the tasks by using our scheduling algorithm for minimizing the error in the final output. Figure 6.8 shows the effect of simultaneous deadline and energy constraints on the system. The error functions we used were obtained by curve fitting to the performance profiles found in [21] as shown in Figure 6.9. The curves in Figure 6.9 also show the effect of input error on the performance profiles. The tasks that don't have an optional part, don't have an effect on the final output quality. The figure shows that the output quality of the system improves with loose time deadline and energy constraints. It can be noted that rate of fall in error decreases with higher deadlines.

Figure 6.8. Error Variation with Time and Energy Constraints

Figure 6.9. Error Functions from Performance Profiles

42

CHAPTER 7

CONCLUSION

This work has concentrated on developing a model for minimizing the final error for systems with dependent IRIS tasks. Greedy algorithms are developed for minimizing error in both offline and online stages of the systems. The advantages of using this model are justified by comparing against a base algorithm and by applying the model to a real world system. Task graphs with different characteristics are studied with a run-time model depicting actual workloads. The developed algorithms can be made use along with real world system simulations for studying the effect of resource allocation on final output quality. Future work includes the use of hierarchical scheduling methods to allow IRIS tasks to coexist with traditional 0-1 task sets. The instrumenting of IRIS code which would allow one to determine its execution progress and thereby provide additional run-time information to the system without imposing too great an overhead is another promising area

BIBLIOGRAPHY

[1] Chung, Jen-Yao, Liu, Jane W. S., and Lin, Kwei-Jay. Scheduling periodic jobs that allow imprecise results. *IEEE Trans. Comput. 39* (September 1990), 1156–1174.

[2] Dey, J. K., Kurose, J., and Towsley, D. On-line scheduling policies for a class of iris (increasing reward with increasing service) real-time tasks. In *IEEE Transactions on Computers* (July 1996), pp. 802–813.

[3] Dey, J. K., Kurose, J. F., Towsley, D., Krishna, C. M., and Girkar, M. Efficient on-line scheduling for a class of iris real-time tasks. In *ACM SIGMETRICS Performance Evaluation Review* (June 1993), pp. 217–228.

[4] Dey, Jayanta K., Kurose, James F., Towsley, Donald F., Krishna, C. M., and Girkar, Mahesh. Efficient on-line processor scheduling for a class of iris (increasing reward with increasing service.) real-time tasks. In *SIGMETRICS* (1993), pp. 217–228.

[5] Feng, W., and Liu, J. W. S. Algorithms for scheduling real-time tasks with input error and end-to-end deadlines. In *IEEE Transactions on Software Engineering* (Feb. 1997), pp. 93–106.

[6] Feng, W., and Liu, J.W.S. An extended imprecise computation model for time-constrained speech processing and generation. In *Real-Time Applications, 1993., Proceedings of the IEEE Workshop on* (may 1993), pp. 76 –80.

[7] Gupta, Vijay. On an anytime algorithm for control. *Proceedings of the 48h IEEE Conference on Decision and Control CDC held jointly with 2009 28th Chinese Control Conference*, 0 (2009), 6218–6223.

[8] Humphrey, M., and Stankovic, J. A. Predictable threads for dynamic, hard real-time environments. In *IEEE Transactions on Parallel and Distributed Systems* (Mar. 1999), pp. 261–296.

[9] Koopmans, T. C., and Beckmann, M. J. Assignment problems and the location of economic activities. In *IEEE 21st Symposium on Operations Research* (Dec. 1973), pp. 498–516.

[10] Kwok, Yu-Kwong, and Ahmad, Ishfaq. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys 31*, 4 (1999), 406–471.

[11] Lin, Natarajan, and Liu. Imprecise results: Utilizing partial computations in real-time systems. In *IEEE 8th Real-Time Systems Symposium* (Dec. 1987), pp. 210–217.

[12] Lin, K., Natarajan, S., and Liu, J. Concord: A distributed system for making use of imprecise results. In *COMPSAC* (Oct. 1987).

[13] Liu, C. L., and Layland, James W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM 20* (January 1973), 46–61.

[14] Menger, Karl. Travelling salesman problem.

[15] Park, Jaehyun, Shin, Donghwa, Chang, Naehyuck, and Pedram, Massoud. Accurate modeling and calculation of delay and energy overheads of dynamic voltage scaling in modern high-performance microprocessors. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design* (New York, NY, USA, 2010), ISLPED '10, ACM, pp. 419–424.

[16] Pillai, Padmanabhan, and Shin, Kang G. Real-time dynamic voltage scaling for low-power embedded operating systems. pp. 89–102.

[17] Shih, W., and Liu, J. On-line scheduling of imprecise computations to minimize error. In *IEEE RealTime Systems Symposium* (Dec. 1992).

[18] Source, Open. Xvid tool.

[19] Tchamgoue, Guy Martin, Kim, Kyong Hoon, Jun, Yong-Kee, and Lee, Wan Yeon. Hierarchical real-time scheduling framework for imprecise computations. In *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing* (2010), pp. 273–280.

[20] T.Dean, and M.Boddy. An analysis of time-dependent planning. In *Proceedings of 7th National Conference on Artificial Intelligence* (Aug. 1988), pp. 49–54.

[21] Zilberstein, Shlomo, and Russell, Stuart J. Anytime sensing, planning and action: A practical model for robot control. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (Chambery, France, 1993), pp. 1402–1407.

[22] Zilberstein, Shlomo, Washington, Richard, Bernstein, Daniel S., and Mouaddib, Abdel-Illah. Decision-theoretic control of planetary rovers. In *Revised Papers from the International Seminar on Advances in Plan-Based Control of Robotic Agents* (London, UK, 2002), Springer-Verlag, pp. 270–289.

[23] a Alvarez, P. Mej, Melhem, R., and Moss, D. An incremental approach to scheduling during overloads in real-time systems. In *IEEE RealTime Systems Symposium* (Dec. 2000), pp. 283–293.