Response-Time Analysis for globally scheduled Symmetric Multiprocessor Platforms [*]

Marko Bertogna, Michele Cirinei
Scuola Superiore Sant'Anna, Pisa, Italy
E-mail: marko@sssup.it, cirinei@gandalf.sssup.it

Abstract

In the last years, a progressive migration from single processor chips to multi-core computing devices has taken place in the general-purpose and embedded system market. The development of Multi-Processor Systems is already a core activity for the most important hardware companies. A lot of different solutions have been proposed to overcome the physical limits of single core devices and to address the increasing computational demand of modern multimedia applications. The Real-Time community followed this trend with an increasing number of results adapting the classical scheduling analysis to parallel computing systems.

This paper will contribute to refine the schedulability analysis for Symmetric Multi-Processor (SMP) Real-Time systems composed by a set of periodic and sporadic tasks. We will focus on both fixed and dynamic priority global scheduling algorithms, where tasks can migrate from one processor to another during execution. By increasing the complexity of the analysis, we will show that an improvement is possible over existing schedulability tests, significantly increasing the number of schedulable task sets detected. The added computational effort is comparable to the cost of techniques widely used in the uniprocessor case. We believe this is a reasonable cost to pay, given the intrinsically higher complexity of multi-processor devices.

1. Introduction

In the last decade, an increasing number of multi-core systems has been proposed in the embedded system domain as well as in the high level computing market. The major hardware providers are already developing the second generation of multi-processor chips, and are spending a considerable amount of resources in the research for next-generation parallel architectures.

The integration of multiple processors on a single chip constitutes one of the most important innovations in the design and development of embedded real-time systems. The reasons for moving to Multi-Processor Systems on Chip (MPSoC) are mainly technological. The computing power of a chip can be increased either by raising the operating frequency or by using more computing units that can work in parallel. Current electronic manufacturing process below 65nm impose strong constraints on the operating frequencies of computing devices, due to the more than linear increase of the consumed power and the temperature reached inside a chip. A way to circumvent this problem is to distribute the computing effort to multiple units and dedicated processing elements that are operated in parallel at lower frequencies. This is the solution adopted by, for instance, TI's OMAP [28], NXP's Nexperia [26], STM's Nomadik [27], ARM's MPCore [25], Sony-IBM-Toshiba's Cell [30], and many others.

From the Real-Time community perspective, this kind of platforms represents an interesting workbench upon which validate the scheduling theory for multi-processor systems. Newly proposed mechanisms allow to mitigate the cost of cache misses, task preemptions and inter-processor migrations, allowing to overcome the drawbacks of existing system models. For instance, ARM's MPCore has a mechanism that allows to retrieve a data that is not present in the local CPU's cache, fetching it directly from the local cache of other processing units. Solution of this kind could then reduce the penalties associated to migration-based algorithms, suggesting the use of *global* schedulers instead of statically partitioning the tasks to the available processors.

A *global* scheduler maintains a single scheduling queue from which tasks are extracted to be dynamically executed on the available CPUs. Taking ”centralized” scheduling decision at system level will allow a better view of the whole system, dealing more efficiently with dynamic load variations and temporary overload conditions. Indeed, a well-known result from the queueing theory is that using a single queue scheduler results in lower average response times than having a queue for every single processor [22]. Intuitively, this is motivated by the fact that with a global scheduler no processor is idled when there is backlogged work to do, ie. the scheduler is *work-conserving*. On the other side, when tasks are statically partitioned to the CPUs, it is very likely that an unbalanced load distribution will result in losing the work-conserving behavior of the system. Further motivations on the advantages of the global approach can

[*]This research has been supported

be found in [2], where a convincing argument shows that the average number of preemptions in a partitioned system is typically higher than in a globally scheduled system.

However, the superiority of global approaches cannot be so easily extended to real-time scheduling performances. Even if global schedulers have lower response times on average, the same cannot be said about worst-case performances. Instead, it has been shown that no approach dominates the other [18], ie. there are task sets that are schedulable only with partitioned solutions and others that can be scheduled only using a global scheduler. Due to the complexity and the dimensions of the problem, a complete theory of real-time scheduling for multi-processor systems is still to come. Besides, in literature there are more efficient schedulability tests for the partitioned case. Since they rely on well known uniprocessor techniques, the performances of the existing schedulability tests for partitioned systems are slightly better than those provided by the few results addressing the global case. Even if this field recently obtained an increasing interest, there is still a lot of space left for improvements. This paper will contribute filling this gap.

1.1. Contribution

We will develop a new approach to analyze the timely properties of Real-Time systems scheduled upon identical multiprocessor platforms. We will show the limits of existing results and overcome their major disadvantages enhancing techniques widely used for the uniprocessor schedulability analysis. The Response Time Analysis (RTA) will be adapted to systems composed by more than one CPU, allowing to derive schedulability tests that dramatically improve over all recently proposed algorithms.

This work will be mainly focused on globally scheduled real-time systems composed by a set of periodic and sporadic tasks. The effectiveness of the proposed analysis will be tested using representative migration-based scheduling algorithms, such as Earliest Deadline First (EDF) and Fixed Priority (FP), and proved with convincing arguments and extensive simulations. However, we believe that the techniques hereafter described can be easily adapted to other kinds of global scheduling algorithms.

The rest of the paper is organized as follows. In Section 2 we introduce the terminology and notation. In Section 3, a panoramic view of the existing schedulability tests for globally scheduled multiprocessor systems is presented. In Section 4, we present our new analysis, adapting it to EDF and FP scheduled systems. General considerations will be exposed in Section 5, while the effectiveness of the proposed approach will be validated in Section 6 through a set of experiments. Finally, in Section 7 we present our conclusions.

2. System model

The notation described in this Section and used in the rest of the paper is summarized in Figure 1.

Symbol	Description
m	Number of processors in the platform
n	Number of tasks in the task set
τ	Task set
τ_k	k-th task $\in \tau$
J_k^j	j-th job of task τ_k
C_k	Worst-case computation time of τ_k
D_k	Relative deadline of τ_k
T_k	Period or minimum interarrival time of τ_k
r_k^j	Release time of job J_k^j
f_k^j	Finishing time of job J_k^j
d_k^j	Absolute deadline of job J_k^j
s_k	$\min_{J_k^j \in \tau}(d_k^j - f_k^j)$, ie. minimum slack of τ_k
R_k	$\max_{J_k^j \in \tau}(f_k^j - r_k^j)$, ie. response time of τ_k
U_k	C_k/T_k, utilization of τ_k
U_{max}	$\max_{\tau_i \in \tau}(U_i)$
U_{tot}	$\sum_{\tau_i \in \tau}(U_i)$, ie. total utilization of task set τ
λ_k	C_k/D_k, density of τ_k
λ_{max}	$\max_{\tau_i \in \tau}(\lambda_i)$
λ_{tot}	$\sum_{\tau_i \in \tau}(\lambda_i)$, ie. total density of task set τ
$I_k(a,b)$	Interference on τ_k in interval $[a,b)$
$I_k^i(a,b)$	Interference of τ_i on τ_k in interval $[a,b)$
$\varepsilon_k(a,b)$	Carry-in of τ_k in interval $[a,b)$
$z_k(a,b)$	Carry-out of τ_k in interval $[a,b)$
$W_k(a,b)$	Worst-case workload of τ_k in $[a,b)$

Figure 1. Notation used throughout the paper.

We consider a set τ of n periodic and sporadic tasks [14] to be scheduled on m identical processors using a global algorithm. Each task $\tau_k = (C_k, D_k, T_k) \in \tau$ is characterized by a worst-case computation time C_k, a period or minimum interarrival time T_k, and a relative deadline D_k. Unless otherwise stated, we will assume every task having *constrained* deadlines, ie. every deadline is less than or equal to the corresponding period. Results for *implicit* deadline systems – having deadline equal to period – can be easily derived as special cases of the constrained deadline model.

The *utilization* of a task is defined as $U_k = \frac{C_k}{T_k}$, while the *density* as $\lambda_k = \frac{C_k}{D_k}$, representing the "worst-case" request of a task in a generic time interval. Let U_{max} (resp. λ_{max}) be the largest utilization (resp. the largest density) among all tasks: $U_{\text{max}} = \max_{\tau_i \in \tau}\{U_i\}$ (resp. $\lambda_{\text{max}} = \max_{\tau_i \in \tau}\{\lambda_i\}$).

A task τ_k is a sequence of jobs J_k^j, where each job is characterized by an arrival time r_k^j and a finishing time f_k^j. We say that a job is *ready* at time t, if $t \in [r_k^j, f_k^j)$. Moreover, each job has an absolute deadline $d_k^j = r_k^j + D_k$, which represents the latest time instant at which we can accept the job to complete execution. The *minimum slack* s_k of a task τ_k is defined as the minimum distance between the absolute deadline and the finishing time of jobs of τ_k, ie. $s_k = \min_{J_k^j \in \tau_k}(d_k^j - f_k^j)$. Finally, the *response time*

R_k of task τ_k is the worst-case finishing time among all its jobs, ie. $R_k = \max_{J_k^j \in \tau_k}(f_k^j - r_k^j)$. Note that when a task set is schedulable, each task has a non-negative slack and a response time lower than or equal to the deadline.

The global schedulers analyzed in this paper maintain a system-wide queue in which ready tasks are inserted and ordered following some kind of policy (EDF, fixed priority, etc.). When a processor is idle, a dispatcher will extract from top of this queue the highest priority task, and will schedule it on the available CPU until it completes execution, or is preempted by another task. Whenever a task with priority higher than one of the executing tasks is released, the task having lowest priority among the executing ones is preempted and re-inserted in the ready queue. This mechanism guarantees that the m highest priority ready tasks are always the ones executing on the multi-processor platform.

Depending on the policy used to sort the ready queue, global schedulers will be divided into static-priority, fixed-job-priority and dynamic-job-priority schedulers. Static priority systems are often shortly called *fixed priority*, omitting to explicitly refer to task priorities. Similarly, schedulers that don't change the priority of a job during its execution are also called *priority-driven*.

2.1. Interference and workload

To analyze the interactions among tasks concurrently executing upon a multiprocessor platforms, we hereafter define two parameters that will be useful in the schedulability analysis of the considered system: the *interference* and the *workload*.

Interference The interference $I_k(a, b)$ on a task τ_k over an interval $[a, b]$ is the cumulative length of all intervals in which τ_k is backlogged but cannot be scheduled on any processor due to the contemporary execution of m higher priority tasks.

We also define the interference $I_k^i(a, b)$ of a task τ_i on a task τ_k over an interval $[a, b]$ as the cumulative length of all intervals in which τ_k is backlogged but cannot be scheduled on any processor, while τ_i is executing. Notice that by definition:

$$I_k^i(a, b) \le I_k(a, b), \quad \forall i, k, a, b. \tag{1}$$

We underline here that for fixed priority systems the interference caused by lower priority tasks is always null, ie. it is: $I_k^i(a, b) = 0, \forall \tau_i$ with priority lower than τ_k.

Workload The *workload* $W_k(a, b)$ of a task τ_k in an interval $[a, b)$ represents the amount of computation that the task requires in $[a, b)$ in a given situation. As in [7], we define $W_k(a, b)$ taking into account three different contributions (see Figure 2):

- *body*: the contribution of all jobs with both release time and deadline in the interval; each job contributes to the workload in that interval with a complete execution time C_k;
- *carry-in*: the contribution of at most one job (called *carried-in job*) with release time before a and deadline in $[a, b)$; this job contributes with the *carry-in* ε_k, i.e. the fraction of its execution time actually executed in the interval.
- *carry-out*: the contribution of at most one job (called *carried-out job*) with release time before $[a, b)$ and deadline after b; this job contributes with the *carry-out* z_k, i.e. the fraction of its execution time actually executed in the interval.

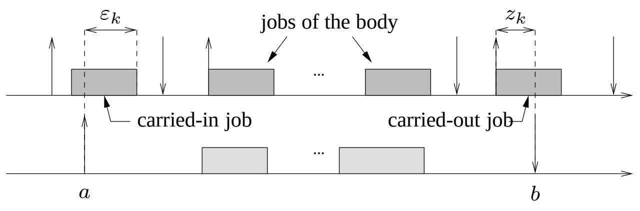


Figure 2. Body, carried-in and carried-out jobs of task τ_k in interval $[a, b]$.

If it would be possible to derive correct values for all interferences or workloads with a limited computational effort, then a schedulability test would easily follow. However, as we will see in Section 4, this is not so easy in the multiprocessor case.

2.2. Time division

Despite the fact that for mathematical convenience, time-instants and interval lengths are often modeled using real numbers, in an actual system time is not infinitely divisible. The times of event occurrences and durations between them cannot be determined more precisely than one tick of the system's most precise clock. Therefore, any time value t involved in scheduling is assumed to be a non-negative integer value and is viewed as representing the entire interval $[t, t + 1)$.

This convention allows the use of mathematical induction on clock ticks for proofs, avoids potential confusion around end-points, and prevent impractical schedulability results that rely on being able to slice time at arbitrary points.

3. Related Work

Response Time Analysis (RTA) is an effective technique that has been widely used to derive schedulability tests and properties for various different models of task systems scheduled on a single processor. Even if RTA has initially been applied to fixed priority uniprocessor systems [6, 17], later works addressed as well the EDF case [31, 29]. Basically, uniprocessor RTA relies on the concepts of *critical instant* and *busy period*. A *critical instant* of a task is an arrival time of an instance such that it suffers the worst possible interference. This allows to find the worst possible response time for the considered task. For fixed priority scheduling, the simultaneous activation of all tasks represents a critical instant [23]. For EDF, things slightly change, ie. the worst-case is not necessary given by an instance arriving when all other tasks are contemporarily released. However the notion of critical instant may still be useful, since it can be proved [14] that the worst-case response time for a task can be found inside a continuously backlogged interval starting with the synchronous arrival of all tasks. This interval is often called *busy period*, or *problem window*.

A schedulability test for periodic and sporadic task sets is then easily derived checking the response times of all tasks in an interval starting with a critical instant and in which jobs are released as soon as possible, and comparing it to the corresponding deadlines.

However, when trying to adapt these technique to multiprocessor systems, there are various anomalies to consider. There are situations [24] in which the synchronous case is not the worst one, and others [3] in which enlarging the interarrival time of the jobs of a task can render the system unfeasible. This is a big problem when analyzing multiprocessor platforms, since it is not so easy to find a "representative" interval where to check if deadlines are met in the worst case situation.

If a multiprocessor system is the target platform, only sufficient results can then be derived in a reasonable amount of time. The few existing results applying Response Time Analysis to globally scheduled multiprocessor systems were only a first attempt to generalize the uniprocessor techniques to the more complex case under consideration. Sufficient RTA-based schedulability tests are shown in [24, 4, 17] for multiprocessor system scheduled with fixed priority. We hereafter recall the main result.

Theorem 1 (from [24, 4, 17]) *Given a task set τ scheduled with fixed priority, a bound on the maximum response time R_k^{max} of a task $\tau_k \in \tau$ is given by the fixed point reached, iteratively repeating the following operation with initial value $R_k^{max} = C_k$:*

$$R_k^{max} \leftarrow C_k + \frac{1}{m} \sum_{\tau_j \in hp(k)} \left(\left\lceil \frac{R_k^{max}}{T_j} \right\rceil C_j + C_j \right) \qquad (2)$$

where $hp(k)$ is the set of tasks with priority higher then τ_k's.

A sufficient schedulability condition is then obtained checking if $R_k^{\max} \leq D_k$ for every $\tau_k \in \tau$. Basically, with Equation 2, a bound on the maximum response time is derived considering the maximum possible contributions, ie. equal to the task WCET, for both the carried-in and carried-out jobs. This is an overly pessimistic assumption, leading to a significant number of rejected task sets.

A better approach to refine carry-in and carry-out estimations has been proposed by Baker for both fixed priority [10] and EDF [9] global scheduling, and later improved in [33]. Goossens *et al.* [21] addressed the schedulability problem by a different point of view, deriving simple and effective utilization bounds to be used with implicit deadline systems scheduled with global EDF. Their analysis has been later adapted to constrained deadline systems in [15], where as well other methods to bound the worst-case interference have been proposed for the EDF case. Density and utilization bounds for fixed priority global schedulers are derived in [16].

When considering scheduling algorithms that may change the priority of an executing job, Pfair [13, 1] algorithms are optimal for implicit deadline systems, allowing a schedulable utilization equal to the available system capacity. However, such systems can have a number of context changes significantly higher than priority-driven schedulers. Instead, if the priority of an executing job is fixed, the number of preemptions in an interval is bounded by the number of jobs arrivals in the same interval. Recently, dynamic-job-priority algorithms have been proposed to achieve high schedulability performances, at a lower preemption cost [5]. A dynamic-job-priority algorithm that has the same worst-case number of preemptions of EDF, but much better scheduling performances for multiprocessor systems is EDZL [19, 20].

We believe the above list of results well represents the state-of-the-art of global scheduling analysis for the task model described in Section 2. We omitted other results that have been obsoleted by the cited papers, or that assumed different system models.

Since, due to space reasons, it is not possible to apply our analysis to every major global scheduling algorithm, we decided to show two representative cases: EDF and FP. Results for other kinds of algorithms could then be derived in similar ways.

4. Schedulability analysis

As explained in Section 3, to find the worst case response times of tasks scheduled with EDF or fixed priority on a single CPU, it is possible to consider only a particular *critical* situation. However, when analyzing multiprocessor platforms, finding a worst-case situation in which the response time of a task is maximized is not as easy. To the best of our knowledge, it is not possible to find the worst-case behavior of a task without simulating the system. For the sporadic case this would require to check every possible legal arrival of jobs for every task in the system, which is computationally intractable for non-trivial task sets.

Since we don't have a critical instant where to start the analysis, an alternative can be to consider an upper bound on the interference a task might be subject to.

Bounding the Interference An upper bound on the interference is represented by the workload, as the next theorem states.

Theorem 2 *The interference $I_k^i(a,b)$ of a task τ_i on a task τ_k in an interval $[a,b)$ cannot be higher than the workload $W_i(a,b)$ of τ_i in $[a,b)$.*

Proof: Obviously, a task can interfere only when it is executing. The theorem follows from the definition of interference and workload in Section 2.1. ■

We can further restrict the interference on a task τ_k, by noting that no interfering task can contribute to the response time of τ_k for more than $R_k - C_k$. To formally state this result we first need the following lemma, proved in [15].

Lemma 1 (from [15]) *For any global scheduling algorithm it is:*

$$I_k(a,b) \geq x \iff \sum_{i \neq k} \min\left(I_k^i(a,b), x\right) \geq mx$$

The following part of the analysis will consider the particular instance of task τ_k that is subjected to the maximum possible interference. Even if we don't know the location of this instance, nor the conditions at which it is maximally interfered, we can anyway denote it with J_k^*. Since $f_k^* = R_k$, computing an upper bound on the finishing time of J_k^* will lead to a valid upper bound on the response time of τ_k.

With these notations and with the above lemma, a result that will be useful to bound the worst-case interference imposed by each task is hereafter derived.

Theorem 3 *A task τ_k has a response time upper bounded by R_k^{ub} if*

$$\sum_{i \neq k} \min\left(I_k^i(r_k^*, r_k^* + R_k^{ub}), R_k^{ub} - C_k + 1\right) < m(R_k^{ub} - C_k + 1)$$

Proof: If the inequality holds for τ_k, from Lemma 1 we have

$$I_k(r_k^*, r_k^* + R_k^{ub}) < (R_k^{ub} - C_k + 1)$$

therefore J_k^* will be interfered for at most $R_k^{ub} - C_k$ time units. From the definition of interference, it follows that J_k^* (and therefore every other job of τ_k) will complete at most at time R_k^{ub}. ■

To effectively use Theorems 2 and 3 in our response time analysis, we need to derive an estimation of the workload in a window $[r_k^*, r_k^* + R_k^{ub})$.

Bounding the Workload Also evaluating the worst-case workload is a complex task. Again, we will use an upper bound to avoid the need to simulate the system.

Theorem 4 *When no deadline is missed, a bound on the workload of a task τ_i in a generic interval $[a,b)$ can be computed considering a situation in which the carried-in job J_i^ε starts executing at the beginning of the interval, with $a = d_i^\varepsilon - C_i$, and every other instance of τ_i is executed as soon as possible.*

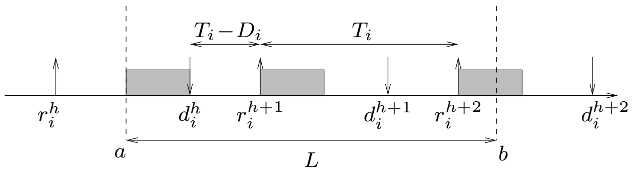


Figure 3. Scenario described in the proof of Theorem 4.

Proof: The situation is represented in Figure 3. Since a job J_i^j can be ready only in $[r_i^j, d_i^j)$ and for at most C_i time units, it is immediate to see that the depicted situation provides the highest amount of execution possible in interval $[a,b)$. Moving backwards the interval, the carry-in cannot increase, while the carry-out can only decrease. Instead, advancing the interval, the carry-in will decrease, while the carry-out can increase by at most the same amount. The situation is periodic. ■

We now compute the workload of task τ_i in an interval $[a.b)$ of length L, in the situation considered in Theorem 4 and represented in Figure 3. Note that the first job of τ_i after the carry-in, is released at time $a + C_i + T_i - D_i$. The next jobs are then released periodically every T_i time units. Therefore the number $N_i(L)$ of jobs of τ_i that contribute with an entire WCET to the workload in an interval of length L is at most $(\lfloor \frac{L-(C_i+T_i-D_i)}{T_i} \rfloor + 1)$. So,

$$N_i(L) = \left\lfloor \frac{L + D_i - C_i}{T_i} \right\rfloor \tag{3}$$

The contribution of the carried-out job can then be bounded by $\min(C_i, L + D_i - C_i - N_i(L)T_i))$. A bound on the workload of a task τ_i in a generic interval of length L is then:

$$\mathfrak{W}_i(L) = N_i(L)C_i + \min(C_i, L + D_i - C_i - N_i(L)T_i) \tag{4}$$

Note that no assumption on the scheduling algorithm used has been made in the proof of the above result. Therefore, the bound of Equation 4 is valid for any scheduling algorithm. Nevertheless, when the algorithm in use is known, other bounds can be derived. Next paragraphs will consider the EDF and FP cases.

4.1. Systems scheduled with EDF

When tasks are scheduled with EDF, the workload in interval $[r_k^*, r_k^* + D_k)$ can be analyzed in a particular situation, as stated by the next theorem.

Theorem 5 *For EDF-scheduled systems, when no deadline is missed, the interference of a task τ_i on a task τ_k in an interval of length D_k is at most*

$$\mathfrak{I}_k^i(D_k) = \text{DBF}_k^i + \min(C_i, \max(0, D_k - \text{DBF}_k^i \frac{T_i}{C_i})), \tag{5}$$

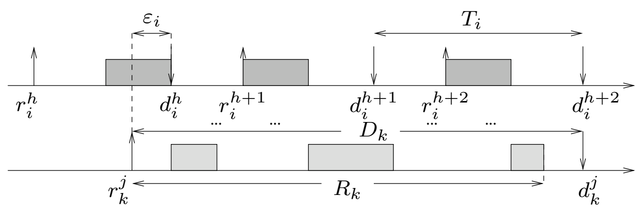


Figure 4. Scenario described in the proof of Theorem 5.

where $\text{DBF}_k^i \doteq \left(\left\lfloor \frac{D_k - D_i}{T_i} \right\rfloor + 1\right) C_i$.

Proof: Follows from Lemma 5 in [15]. Basically, the worst-case situation is described in Figure 4, where the carried-out job J_i^z has its deadline at the end of the interval, ie. coincident with a deadline of τ_k, and every other instance of τ_i is executed as late as possible. The bound on the interference can then be easily derived analyzing the above situation, and is composed by the DBF, representing the body, and the min, representing the carry-in of τ_i in the considered interval. ■

Note that the bound of Theorem 5, differently from the previously derived bounds, is valid only if the length of the considered interval is D_k, ie. the relative deadline of the interfered task.

We are now ready to state a first result for the EDF case.

Theorem 6 (RTA for EDF) *An upper bound on the response time of a task τ_k in an EDF-scheduled multiprocessor system can be derived by the fixed point iteration on the value R_k^{ub} of the following expression, starting with $R_k^{ub} = C_k$:*

$$R_k^{ub} \leftarrow C_k + \left\lfloor \frac{1}{m} \sum_{i \neq k} \hat{I}_k^i(R_k^{ub}) \right\rfloor \tag{6}$$

with $\hat{I}_k^i(R_k^{ub}) = \min(\mathfrak{W}_i(R_k^{ub}), \mathfrak{I}_k^i(D_k), R_k^{ub} - C_k + 1)$.

Proof: The proof is by contradiction. Suppose the iteration ends with a value $R_k^{ub} \leq D_k$, but the response time of τ_k is higher than R_k^{ub}. Since the iteration ends, it is

$$R_k^{ub} = C_k + \left\lfloor \frac{1}{m} \sum_{i \neq k} \min(\mathfrak{W}_i(R_k^{ub}), \mathfrak{I}_k^i(D_k), R_k^{ub} - C_k + 1) \right\rfloor$$

For Theorems 2 and 4, $\mathfrak{W}_i(R_k^{ub}) \geq I_k^i(r_k^*, r_k^* + R_k^{ub})$. Let $I_k^{i*} \doteq I_k^i(r_k^*, r_k^* + R_k^{ub})$. For Theorem 5, $\mathfrak{I}_k^i(D_k) \geq I_k^i(D_k) \geq I_k^{i*}$ as long as $R_k^{ub} \leq D_k$. Therefore,

$$R_k^{ub} \geq C_k + \left\lfloor \frac{1}{m} \sum_{i \neq k} \min(I_k^{i*}, R_k^{ub} - C_k + 1) \right\rfloor$$

Since, by hypothesis, the response time of τ_k is higher than R_k^{ub}, the inverse of Theorem 3 gives

$$R_k^{ub} \geq C_k + \left\lfloor \frac{1}{m} m(R_k^{ub} - C_k + 1) \right\rfloor = R_k^{ub} + 1$$

reaching a contradiction.

It remains to show that the iteration converges in a finite amount of time. This is assured by the integer time convention assumed in Section 2.2. ■

A schedulability test can than be performed by repeating the iteration described above for every task $\tau_k \in \tau$. If every iteration ends before the corresponding deadline value, than the task set is schedulable with EDF.

4.2. Fixed Priority systems

For fixed priority systems the bound on the interference given by Theorem 5 isn't applicable. However, another property allows nevertheless to increase the effectiveness of the response time analysis. Assume tasks are ordered by decreasing priority, ie. $i \leq j$ iff τ_i has more priority than τ_j. From the definition of interference, it is clear that no task can contribute to the interference on a higher priority task, ie. $I_k^i = 0, \forall i \leq k$. The next theorem immediately follows from this consideration and the proof of Theorem 6.

Theorem 7 (RTA for FP) *An upper bound on the response time of a task τ_k in a multiprocessor system scheduled with fixed priority can be derived by the fixed point iteration on the value R_k^{ub} of the following expression, starting with $R_k^{ub} = C_k$:*

$$R_k^{ub} \leftarrow C_k + \left\lfloor \frac{1}{m} \sum_{i < k} \hat{I}_k^i(R_k^{ub}) \right\rfloor \tag{7}$$

with $\hat{I}_k^i(R_k^{ub}) = \min(\mathfrak{W}_i(R_k^{ub}), R_k^{ub} - C_k + 1)$.

Even if $\mathfrak{I}_k^i(D_k)$ cannot be inserted inside the minimum defining $\hat{I}_k^i(R_k^{ub})$, we will see that the limitation of the sum to the first $k-1$ terms compensates by far this loss.

4.3. Exploiting slack values

The performance of our response time analysis can be significantly improved with a simple consideration. Since the algorithms of Theorems 6 and 7 allow to find a response time for a task τ_i, when applying the same algorithm to another task τ_k it is possible to consider the previously derived upper bound on the response time of τ_i. This can decrease the possible interference of τ_i on τ_k. Theorems 6 and 7 don't need to be modified. It is enough to change the upper bounds given by Equation 4 and 5, including the slacks previously computed for the interfering tasks. A lower bound on the slack time of a task τ_i is, trivially, $s_i^{lb} = D_i - R_i^{ub}$. Figure 5 represents the worst-case situations with the additional information on the slack of task τ_i. The upper bound

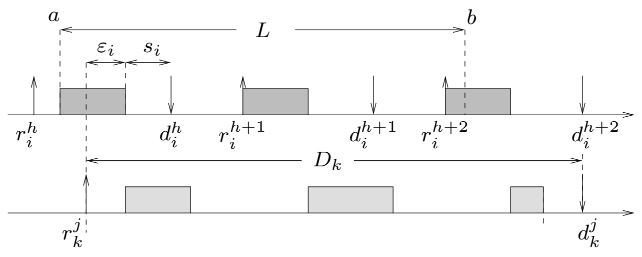


Figure 5. Worst-case situations with slack.

on the workload is updated by changing term $N_i(L)$ in the expression of $\mathfrak{W}_i(L)$ in the following way:

$$\mathfrak{W}_i(L) = N_i(L)C_i + \min(C_i, L + D_i - C_i - s_i - N_i(L)T_i) \tag{8}$$

with

$$N_i(L) = \left\lfloor \frac{L + D_i - C_i - s_i}{T_i} \right\rfloor .$$

Instead, the expression of the worst-case interference $\mathfrak{I}_k^i(D_k)$ can account for a lower carry-in, and can be given by

$$\mathfrak{I}_k^i(D_k) = \text{DBF}_k^i + \min(C_i, \max(0, D_k - \text{DBF}_k^i \frac{T_i}{C_i} - s_i)). \tag{9}$$

Everything else remains unchanged.

Theorems 6 and 7 can then be applied to every task in the system, using each time the most recently computed values for the slack of the interfering tasks. The analysis can then be repeated again starting with the slack values from the previous iteration. The first task, that at the previous iteration didn't consider any slack for the interfering tasks, can this time take advantage of the positive slacks previously computed for the other tasks, leading to a lower worst-case response time.

If the target is to verify the schedulability of the system, the whole procedure can successfully stop when all tasks are verified to have an upper bound on the response time lower than their deadline. If a task still didn't converge when $R_k^{ub} > D_k$, it will be temporarily set aside, waiting for a slack update (ie. increase) of potentially interfering tasks; in this case, if no update takes place during a whole run for all tasks in the system, than there is no possibility for further improvements and the test fails.

On the other hand, if the target is to derive the closest possible value for every response time, the procedure can go on until there is no more change in any response time. Note that every slack function is monotonically non-decreasing since, at each step, the considered interference from other tasks can only be lower than or equal to the interference considered in the precedent step. This allows to bound the overall complexity of the whole slack-based analysis.

Since introducing the slack updates to our analysis will significantly improve performances at a reasonable cost, we suggest the use of this extended RTA version every time there are no tighter requirements on the affordable run-time complexity.

5. Considerations

Computational Complexity The complexity of a single run of the procedure of Theorems 6 and 7 is comparable to the complexity of similar uniprocessor techniques. Since the response time R_k^{ub} of a task τ_k is updated with integer values, a single iteration for a task τ_k will converge, or fail, in at most $(D_k - C_k)$ steps.

However, it is possible to further improve the average behavior of the algorithm, noting that a potential weakness is given by the contribution $(R_k^{ub} - C_k + 1)$ in the minimum of the interference $\hat{I}_k^i(R_k^{ub})$. This value can cause a slow progression of the iteration towards the final value, due to the low rate at which the response time is increased at each step. If the final response time is very late in time, the iteration will potentially converge after a lot of iterations. Even if the observed overall speed of the procedure seems sufficiently high (allowing to positively check millions of tasks in a few seconds), slight modifications on the algorithm may be desirable for faster run-time admission control tests. An alternative that provably increases the speed of the procedure without compromising the performances can be to split the procedure in two stages. In the first stage, the value $(R_k^{ub} - C_k + 1)$ is replaced by $(D_k^{ub} - C_k + 1)$. If the task nevertheless converges to a value $R_k^{ub} \leq D_k$, then it is possible to refine the derived bound on the response time in a second stage, using again the minimum on the interference with the original term $(R_k^{ub} - C_k + 1)$, updated with the value derived in the precedent stage. This allows to proceed by greater steps towards the final bound, eventually retreating if the step was too big. The simulations we ran with this alternative strategy didn't show significant losses in the number of schedulable task sets detected in comparison with the original algorithm.

Another factor that could affect the overall average complexity is the order in which the minimum slacks are updated. In other words, it is possible to apply the RTA Theorems sequentially to every tasks, or alternatively re-start from the first task every time one of the potentially interfering tasks updates its slack value, or, again, follow some particular order to maximize the slack updates at each step. We believe that the first sequential approach represents on average a good compromise. The worst-case complexity of this approach can be derived noting that to trigger a further round of analysis on all tasks, at least one task should have updated its response time. Since every task can increase its slack lower bound at most $(D_k - C_k)$ times, the worst-case number of rounds to be performed can then be bounded by $\sum_k(D_k - C_k)$. Each one of this rounds will take at most $\sum_k(D_k - C_k)$ steps to sequentially update the slacks of all tasks. Therefore a bound on the overall number of steps of the whole process of finding the best possible estimations on the response times for every task in the task set is given by $(\sum_k(D_k - C_k))^2$. This bound can then be lowered noting that not every round requires $\sum_k (D_k - C_k)$ steps, but later rounds will converge sooner. It can be proved that a tighter bound on the overall number of steps is given by $\frac{(\sum_k (D_k - C_k))^2}{2}$, which is is O($n^2 D_{\max}^2$). Since every step is just a sum of at most n contributions, the overall complexity of the RTA is O($n^3 D_{\max}^2$).

When fixed priority schedulers are used, the complexity of our RTA is much lower: since the interference from lower priority tasks is always null, there won't be any advantage in performing more than a single round of slack updates. The overall complexity is therefore O($n^2 D_{\max}$).

The pseudo-polynomial bound on the worst-case number of steps of the most complete version of our RTA considers a very pessimistic situation. We found in our experiments that average performances are much better than that. To give an idea, we have been able to perform the full response time analysis for millions of task sets in a few minutes, for various different system configurations. This rates suggest that our test can be a good candidate also for on-line admission control.

Applicability to other global schedulers The general approach followed allows to extend the main ideas behind the proposed analysis to global scheduler different from plain EDF or fixed priority. As an example, the RTA of Theorem 7 can be applied to *any work-conserving global scheduler*, by extending the sum to every task in the system. This somewhat surprising result can be used to analyze systems for which no schedulability test exists in literature, providing as well useful timely characteristics like slack and response time estimations.

Another important class of schedulers to which our response time analysis is applicable is given by the so-called *hybrid* global scheduling algorithms. These algorithms exploit the advantages of both static and dynamic priority schedulers, scheduling some task with fixed priority and some other with EDF. Examples are given by EDF-US [32], fpEDF [12], EDF^k [21, 8], etc. These solutions allow to overcome the major drawbacks of plain EDF or FP. However, existing schedulability tests for these algorithms seem still very far from necessary conditions, losing a great share of system capacity to guarantee hard real-time performances, relying in most of the cases on utilization and density bounds close to half of the system capacity. The analysis developed in this paper seems instead very promising in this sense, needing only minor changes to be adapted to such systems. Due to the variety of existing hybrid schedulers, a deeper analysis of these algorithms, as well as of other interesting solutions like EDF-ZL [19, 20], is left to future works.

Robustness and sensitivity Note that the minimum slack values s_k, that are computed as by-products of our schedulability analysis, are not only useful to check the schedulability of a task set, but can also be used to measure the *sensitivity* of the system to variation of timely parameters. If every task has a reasonable minimum slack value, the whole system will be sufficiently robust to deal with isolated anomalies and overload conditions.

Alternatively, it is possible to use the available slack to decrease the frequency of the system clock feeding the CPUs in a synchronous Symmetric Multiprocessor Platform without affecting the overall schedulability. The obvious outcome of such a solution would be to allow a corresponding more than linear decrease in the power consumed. Explicit relations between robustness properties and slack parameters depend on the global algorithm used, but, due to space limits, will not be treated in this paper.

6. Experimental Results

In order to validate the proposed test and compare its behavior with the best existing tests, cited in Section 3, we ran a long series of simulations, using different combinations of task parameters. We analyzed as well the behavior of our RTA varying the number of processors, the number of tasks and the total system utilization. Due to space limits, we report here only some of the experiments, which are anyway representative of the general behavior.

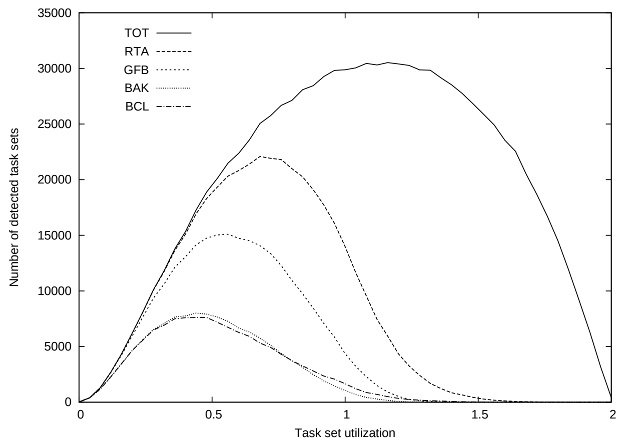
The experiments reported in the figures were generated based on the following characteristics of the tasks: utilization extracted according to exponential distribution with mean 0.25, re-extracting tasks with utilization $U_i > 1$; period (and, implicitly, the execution time) extracted uniformly in $[0, 2000]$; deadline uniformly extracted between C_i and P_i. Histograms in the figures represent $1.000.000$ task sets, each one passing the necessary test for feasibility in [11]. In other words, we excluded from our simulations the task sets that are infeasible according to the test in [11].

Each line represents the number of task sets proved schedulable by one specific test. The curve is drawn connecting a series of points, each one representing the collection of task sets that have total utilization in a range of 4% near the point. For EDF, we considered a test proposed by Baker (BAK, in [9]), the density bound test by Goossens *et al.* (GFB, in [21]) generalized to constrained deadlines, the test by Bertogna *et al.* (BCL, in [15]) and our RTA test. For FP, we implemented the test proposed by Baker (BC, in [33]), the schedulability test and density bound test proposed by Bertogna *et al.* (BCL and DB, respectively, both in [16]), and our RTA test.

Further simulations and different task generations will be shown in an extended version of the paper, currently under preparation. However, we anticipate that the examples we chose for this section well represents the general behavior.

6.1. Evaluation of experiments

In the upper part of Figure 6, we show the case with $m = 2$ for EDF. The RTA-based test clearly outperforms all existing schedulability tests at every utilization. Compared to them, RTA is constantly superior and can detect many schedulable task sets also with $U_{\text{tot}} \geq 1$. As a side remark, note that since we are using the constrained deadline model,



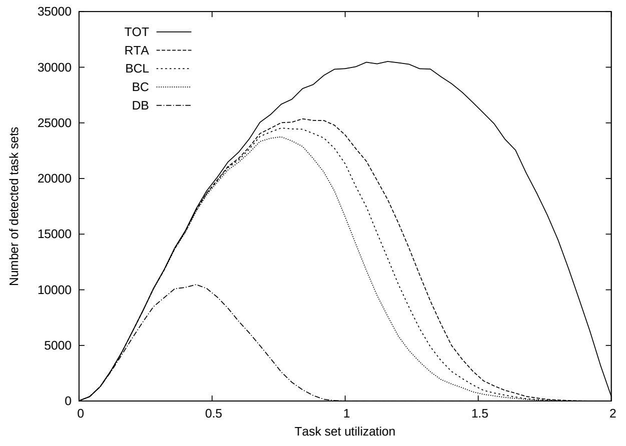


Figure 6. Experiments with EDF (above) and DM (below) on $m = 2$ processors.

no scheduling algorithm can reach a schedulable utilization in the number of processors. To give an *upper bound* on the number of feasible task sets, we included the continuous curve labeled with TOT. This curve *doesn't* represent the number of EDF-schedulable task set, neither it indicates how many task sets are feasible. We included it just to give an indication on how many generated task sets aren't for sure infeasible, using techniques from [11], at the considered utilizations. If an exact feasibility test would exist, its curve would be below the TOT curve. Moreover, considering that EDF isn't optimal for multiprocessors, a hypothetical necessary and sufficient schedulability test for EDF would have an even lower curve.

Similar considerations are valid as well for the FP case, depicted in the lower part of Figure 6. The priority assignment used is Deadline Monotonic (DM). This figure is very meaningful, since it shows that the RTA-based test improves even over the EDF case, getting closer to the upper bound on the general feasibility condition represented by the continuous curve.

Increasing the number of processors the results are similar. In Figure 7 we show the case with $m = 4$ processors both for EDF and FP, plotting only the curves for GFB with EDF, BC with DM and both our RTA-based tests. We omitted the other curves because they are below the considered ones. The RTA test for fixed priority outperforms the correspondent test for EDF. This is due to the fact that for fixed priority systems the interference from lower priority tasks can be neglected, which cannot be made with plain EDF. The higher distance from the TOT curve is motivated by the worse performances of EDF and DM when the number of processor increases, and doesn't seem a weak point of our RTA.

The above considerations suggest that for multiprocessor platforms it may be convenient to use fixed priority scheduling instead of EDF. Even if a common opinion is that the absolute performances of EDF are arguably better than the performances of fixed priority scheduling, we showed that the superiority of FP relatively to the best existing schedulability test largely compensate this disadvantage. Since real-time systems are interested in finding a provable schedulability, and considering the easier implementation of fixed priority systems, a fixed priority scheduler can be preferable in many cases. Moreover, since there is no particular reason in using DM as priority assignment in the multiprocessor case, an interesting task could be to explore which priority assignment could further magnify the performances of the RTA-based schedulability test. We intend to analyze this issue in future works, together with the analysis of more general scheduling algorithm, like hybrid or dynamic-job-priority schedulers, that are expected to have a lower gap from the necessary condition upper bounded by the TOT curve.

7. Conclusions

We developed a new approach for the analysis of real-time task systems globally scheduled on a Symmetric Multiprocessor Platform. Response times and slack values are efficiently computed in pseudo-polynomial time, allowing to derive efficient schedulability tests that can easily be adapted to many different scheduling algorithms. We showed that the proposed approach dramatically improves over existing solutions, significantly increasing the number of schedulable task sets detected. The effectiveness of the analysis has been extensively proved through exhaustive simulations.

References

[1] J. Anderson and A. Srinivasan. Mixed Pfair/ERfair scheduling of asynchronous periodic tasks. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, Delft, The Netherlands, June 2001. IEEE Computer Society Press.

[2] B. Andersson and J. Jonsson. Fixed-priority preemptive multiprocessor scheduling: To partition or not to partition. In *Proceedings of the International Conference on Real-Time Computing Systems and Applications*, pages 337–346, Cheju Island, South Korea, December 2000. IEEE Computer Society Press.

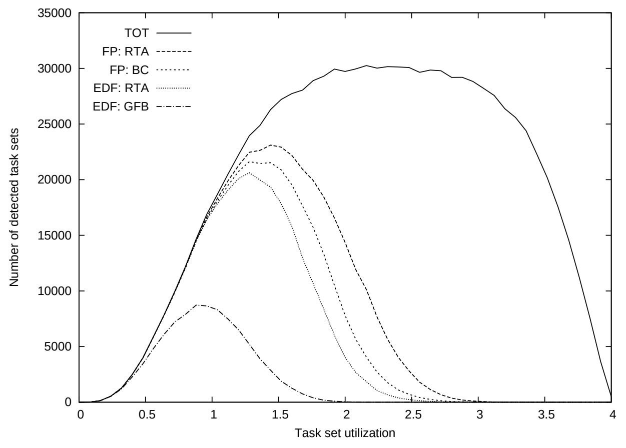


Figure 7. Comparison between EDF and FP schedulability tests on 4 processors.

[3] B. Andersson and J. Jonsson. Some insights on fixed-priority preemptive non-partitioned multiprocessor scheduling. In *Proceedings of the Real-Time Systems Symposium – Work-In-Progress Session*, Orlando, FL, November 2000.

[4] B. Andersson and J. Jonsson. Some insights on fixed-priority preemptive non-partitioned multiprocessor scheduling. Technical Report 01-2, Department of Computer Engineering, Chalmers University of Technology, Sweden, 2001. March, 16.

[5] B. Andersson and E. Tovar. Multiprocessor scheduling with few preemptions. In *RTCSA*, pages 322–334, 2006.

[6] N. Audsley, A. Burns, M. Richardson, and A. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–293, 1993.

[7] T. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 120–129. IEEE Computer Society Press, December 2003.

[8] T. Baker. A comparison of global and partitioned edf schedulability tests for multiprocessors. Technical Report TR-051101, FSU Computer Science, November 2005.

[9] T. P. Baker. An analysis of EDF schedulability on a multiprocessor. *IEEE Transactions on Parallel and Distributed Systems*, 16(8):760–768, 2005.

[10] T. P. Baker. An analysis of fixed-priority schedulability on a multiprocessor. *Real-Time Systems: The International Journal of Time-Critical Computing*, 32(1–2):49–71, 2006.

[11] T. P. Baker and M. Cirinei. A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks. *rtss*, 00:178–190, 2006.

[12] S. Baruah. Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. *IEEE Transactions on Computers*, 53(6), 2004.

[13] S. Baruah, N. Cohen, G. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, June 1996.

[14] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190, Orlando, Florida, 1990. IEEE Computer Society Press.

[15] M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of EDF on multiprocessor platforms. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 209–218, Palma de Mallorca, Balearic Islands, Spain, July 2005. IEEE Computer Society Press.

[16] M. Bertogna, M. Cirinei, and G. Lipari. New schedulability tests for real-time tasks sets scheduled by deadline monotonic on multiprocessors. In *Proceedings of the 9th International Conference on Principles of Distributed Systems*, Pisa, Italy, December 2005. IEEE Computer Society Press.

[17] A. Burns and A. Wellings. *Real-Time Systems and Programming Languages*. Addison-Wesley, 3rd edition, 2001.

[18] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. In J. Y.-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press LLC, 2003.

[19] S. Cho, S. K. Lee, and K.-J. Lin. On-line algorithms for real-time task scheduling on multiprocessor systems. In *IASTED International Conference on Internet and Multimedia Systems and Applications*, pages 395–400, Hawaii, August 2001.

[20] M. Cirinei and T. P. Baker. Edzl scheduling analysis. In *ECRTS*, Pisa, Italy, July 2007.

[21] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real Time Systems*, 25(2–3):187–205, 2003.

[22] D. Gross and C. M. Harris. *Fundamentals of Queueing Theory*. Wiley Series in Probability and Statistics, 1998.

[23] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[24] L. Lundberg. Multiprocessor scheduling of age constraint processes. In *RTCSA*, pages 42–, 1998.

[25] A. A. MPCore. http://www.arm.com/products/cpus/arm11mpcoremultiprocessor.html. Web page, July 2007.

[26] P.-N. Nexperia. http://www.nxp.com. Web page, July 2007.

[27] S. Nomadik. www.st.com/nomadik. Web page, July 2007.

[28] T. I. OMAP. www.omap.com. Web page, February 2006.

[29] J. C. Palencia and M. G. Harbour. Response time analysis of edf distributed real-time systems. *Journal of Embedded Computing*, 1(2):225–237, 2005.

[30] T. C. P. Sony, IBM. http://cell.scei.co.jp/. Web page, July 2007.

[31] M. Spuri. Analysis of deadline scheduled real-time systems. Technical Report 2772, Institut National de Recherche en Informatique et en Automatique, 1996.

[32] A. Srinivasan and S. Baruah. Deadline-based scheduling of periodic task systems on multiprocessors. *Information Processing Letters*, 84(2):93–98, 2002.

[33] T.P.Baker and M.Cirinei. A unified analysis of global edf and fixed-task-priority schedulability of sporadic task systems on multiprocessors. *Journal of Embedded Computing*, 2007. To appear. TR available at http://www.cs.fsu.edu/research/reports/TR-060401.pdf.