Embedded Software in Network Processors - Models and Algorithms

Lothar Thiele, Samarjit Chakraborty, Matthias Gries,
Alexander Maxiaguine, and Jonas Greutert

Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology (ETH) Zürich
CH-8092 Zürich, Switzerland
{thiele, samarjit, gries, maxiagui, greutert}@tik.ee.ethz.ch
http://www.tik.ee.ethz.ch

Abstract. We introduce a task model for embedded systems operating on packet streams, such as network processors. This model along with a calculus meant for reasoning about packet streams allows a unified treatment of several problems arising in the network packet processing domain such as packet scheduling, task scheduling and architecture/algorithm explorations in the design of network processors. The model can take into account quality of service constraints such as data throughput and deadlines associated with packets. To illustrate its potential, we provide two applications: (a) a new task scheduling algorithm for network processors to support a mix of real-time and non-real-time flows, (b) a scheme for design space exploration of network processors.

1 Introduction

The need for intelligent and flexible network packet processing at high data rates, required by many emerging applications, have led to the development of a new class of devices called *network processors* (NPs). NPs are highly programmable dedicated processors optimized to perform packet processing functions, and will become critical components of next-generation networking equipments. Although there is a wide variety of application areas that are addressed by NPs, their task can generally be viewed as manipulation of packets. These include packet processing tasks (such as classification, forwarding, "deep" packet analysis, data stream manipulation, security, TCP termination, etc.) and traffic management tasks (such as bandwidth management, load balancing, admission control, etc.).

The functionality of an NP and hence its architecture and implementation depend to a very large extent on its placement in the Internet hierarchy. Those deployed in the access network are usually required to support a wide and varied range of packet processing functions, but at relatively low data rates (of around 100 Kbps per end-user for standard networks and 1-10 Mbps for networks based on emerging technologies like DSLs), and hence are ideal for a largely software based implementation. NPs placed in core or backbone networks have to handle

much higher data rates (in the range of several Gbps) which therefore restricts their processing capabilities. In this case many core functionalities are implemented in software on a general purpose processor and other operations are delegated to dedicated coprocessors.

Irrespective of the type and functionality of an NP, because of flexibility reasons, software forms an integral part of it. In this direction, very recently several papers proposed different software architectures for flexible and configurable routers which can easily be programmed and extended to support new functionality rather than only routing packets (see [14, 16] and the references therein). However, till date there has been no formal and unified study of this subject. All the previous papers dealt with this topic mainly from a software engineering perspective. There is a large body of literature on packet scheduling, but network processing is much more than that. In this paper we attempt to initiate a formal study of packet processing devices such as NPs. As a first step in this direction, we outline a framework for embedded systems operating on packet streams and based on it provide a unified treatment of some problems arising in this area. Our framework is motivated to some extent by some recent models used in packet scheduling (such as [7, 8, 12]) and primarily consists of

- a task and resource model for network processors and
- a calculus which allows to reason about packet streams and their processing.

As an application of the above, we consider two examples: the first is that of task scheduling in an embedded packet processor, and the second is related to hardware-software interactions where we illustrate the potential of our model and calculus for estimating delays and memory requirements in the context of a design space exploration of NPs. The next two sections introduce our model and the calculus, following which we describe the two examples of scheduling and design space exploration in Sections 4 and 5.

2 Model of Computation for Packet Processing

Our model for describing typical task structures and their mapping to hardware and software resources is not very different from the specification of conventional real-time systems, see e.g. [9]. Nevertheless, there are some notable differences, for example the way in which the capabilities of processing devices and sequences of events or packets are specified. These form the basis of the described unification of models for, for example packet and task scheduling.

It may be noted that here we will define a basic model only. Depending on the particular design task and the envisioned level of abstraction, additional information can easily be added.

Definition 1 (Task Structure). *The task structure of a network processor consists of a set of flows $f \in F$ and a set of tasks $t \in T$. To each flow f there is associated a connected, directed and acyclic task graph $G(f)$. It consists of a set of task nodes $T(f) \subseteq T$ and a set of directed edges $E(f) \subseteq T(f) \times T(f)$. Each task graph has a unique source node $s(f) \in T(f)$ having no incoming edges.*

Figure 1 shows a relatively simple task structure of a sample NP that will be used throughout the paper. There are twenty five tasks (denoted by t) which perform general packet processing functions and operations dedicated to encryption/decryption and voice processing. Depending on the flow to which a packet belongs, different sequences of tasks are executed, see the dotted lines. In the above terminology, we have five flows $f \in F$ and the corresponding task graphs $G(f)$ are simply chains.

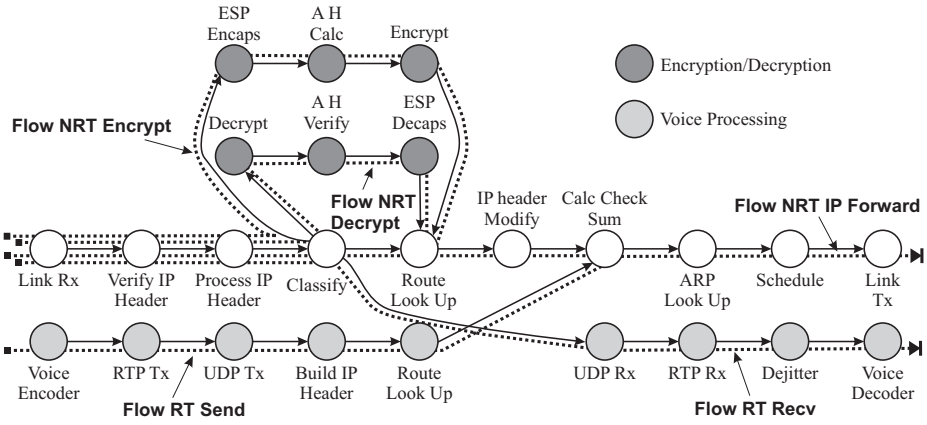

Fig. 1. Example of a task graph corresponding to a simple network processor, see Definition 1.

In addition to the above definition of the basic task structure, it is useful to define a simple resource model consisting of a set of computing resources which are able to execute the tasks defined in the task structure.

Definition 2 (Resource Structure). *Given a set of resources $s \in S$, the function cost : $S \to \mathbb{R}^{\geq 0}$ denotes the (relative) cost of the corresponding resource. The mapping relation $M \subseteq T \times S$ defines possible mappings of tasks $t \in T$ to resources, i.e. if $(t, s) \in M$ then t can be executed on s.*

In order to enable a performance analysis of the above defined resource structure, we now introduce some additional functions.

Definition 3 (Timing Properties). *To each flow $f \in F$ there is associated an end-to-end deadline $d : F \to \mathbb{R}^{\geq 0}$. If a task f can be executed on a resource s, then it creates a "request", i.e. for all $(t, s) \in M$ there exist a request $w(t, s) \in \mathbb{R}^{\geq 0}$.*

The above definitions can be interpreted as follows: To each flow f there is associated a set of tasks T. If a packet belonging to this flow arrives at the processing device, the tasks corresponding to the flow are executed respecting the

partial order defined by the edges. To each of these packets there is associated a (possibly infinite) end-to-end deadline $d(f)$. The execution of the complete task graph corresponding to the packet must be finished within at most $d(f)$ time units after the packet arrival. The resource nodes s denote the available computing resources. If a task is executed on a computing resource, it creates a request w, for example measured in number of instructions.

As an example, Figure 2 shows a part of the resource structure of our simple network processor. Only five of the eight resource nodes $s \in S$ and four of the twenty five tasks $t \in T$ are shown along with their associated costs *cost*, mapping relations $(t, s) \in M$ and requests $w(t, s)$.

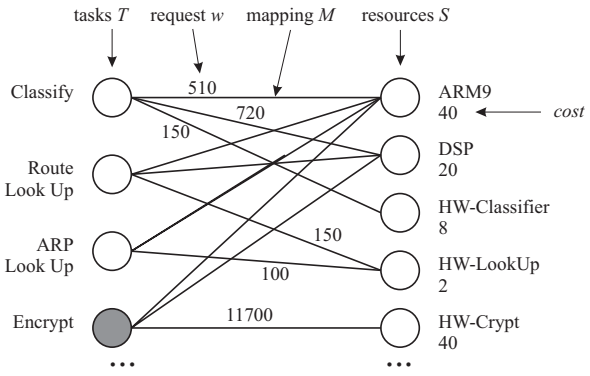

Fig. 2. Example of a resource structure as defined in Definition 2.

Associated to each flow are also some arrival curves modelling the incoming event/packet sequences, and to each resource some service curves modelling its processing capabilities; details of these will be described in the next section.

The above model abstracts important properties of a task structure or resource structure such as communication nodes, interconnection structure, memory or data nodes, power consumption, cost functions and bandwidth requirements. It should be obvious that the model can easily be extended. Finally, it may be noted that the above model closely resembles the one that was used for a design space exploration of hardware/software architectures in [6].

3 Modeling Discrete Event Streams and Systems

3.1 Basic Models

Traditionally event streams are modeled statistically. Methods that have been applied are well known in the area of queuing theory. However, If we are interested in hard bounds instead of failure probabilities, it is more suitable to use a more appropriate characterization of discrete event models and systems.

In order to unify the methods commonly used in the domain of communication networks with those used in operating systems and real-time scheduling, we adopt the concept of arrival curves and service curves, see [12]. Recently, this approach has been formalized [7] and put into a network theory context based on a linear algebra for discrete event systems, see e.g. [3]. This approach has also been used to derive efficient packet scheduling algorithms, see e.g. [1].

Definition 4 (Arrival and Service Function). *An event stream can be described by an arrival function R where $R(t)$ denotes the number of events that have arrived in the interval $[0,t)$. A computing resource can be described by a service function C where $C(t)$ denotes the number of events that could have been served in the interval $[0,t)$.*

Depending on the context in which these functions are used, the number of events may model the number of packets, the number of bytes or the number of instructions to be performed. Obviously, the functions $C(t)$ and $R(t)$ are non-decreasing and can be used to accurately describe the incoming events and the processing capabilities. The abstraction used in this paper is based on deterministic bounds on the corresponding behavior.

Definition 5 (Arrival and Service Curves). *The upper and lower arrival curves $\alpha^u(\Delta), \alpha^l(\Delta) \in \mathbb{R}^{\geq 0}$ of an arrival function $R(t)$ satisfy*

$$\alpha^l(t-s) \leq R(t) - R(s) \leq \alpha^u(t-s) \qquad \forall s,t : 0 \leq s \leq t$$

The upper and lower service curves $\beta^u(\Delta), \beta^l(\Delta) \in \mathbb{R}^{\geq 0}$ of a service function $C(t)$ satisfy

$$\beta^l(t-s) \leq C(t) - C(s) \leq \beta^u(t-s) \qquad \forall s,t : 0 \leq s \leq t$$

Similar functions have been used in network calculus [7] and its applications in hard real-time systems, see [23, 24] and [29]. Therefore, we will not repeat those results here. However, it may be useful to note a simple interpretation of the above definitions. The values $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ can be interpreted as the minimum and maximum number of events arriving within any time interval of length Δ, respectively. Therefore, given R, the corresponding curves can be computed using $\alpha^l(\Delta) = \min_{u \geq 0}\{R(\Delta+u) - R(u)\}$ and $\alpha^u(\Delta) = \max_{u \geq 0}\{R(\Delta+u) - R(u)\}$. In a similar way, the service curves $\beta^l(\Delta)$ and $\beta^u(\Delta)$ can be interpreted as the minimum and maximum available computing service within any time interval of length Δ, respectively. Therefore, given the service function C, we have $\beta^l(\Delta) = \min_{u \geq 0}\{C(\Delta + u) - C(u)\}$ and $\beta^u(\Delta) = \max_{u \geq 0}\{C(\Delta + u) - C(u)\}$.

Figure 3 shows examples of upper and lower arrival curves and service curves. Referring to the task and resource structures defined in Section 2, we can now extend the characterization of flows and resource nodes by the above defined curves.

Definition 6 (Curves and Flows). *To each flow f there are associated upper and lower arrival curves $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$, respectively. To each resource s there are associated upper and lower service curves $\beta^u(\Delta)$ and $\beta^l(\Delta)$, respectively.*

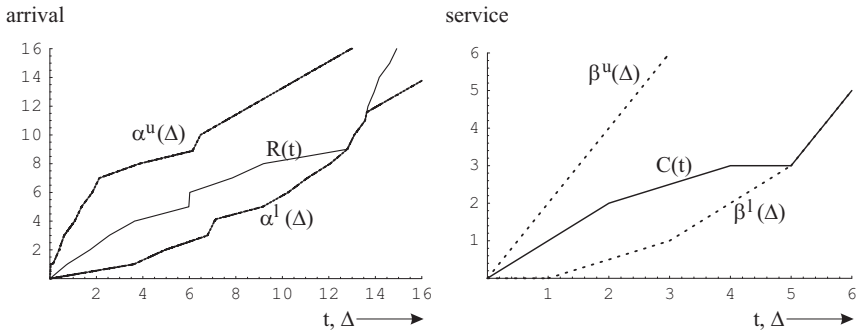

Fig. 3. The left hand side shows an arrival function that has been generated from a finite size exponentially distributed event sequence with mean 1. The right hand side may model a service curve from a resource node that processes 1 event per time unit in the interval $[0, 2]$, only $1/2$ a unit in $[2, 4]$ (e.g. because of a processor share with another task), has no computing capability in $[4, 5]$ (e.g. because of an interrupt handler), and can process 2 events per time unit starting from time 5.

In order to understand the processing of event streams using resource nodes, we start with a simple unit receiving only one event stream, see also [23], [24] and [29]. In this case, the number of events is supposed to denote the computation request of the incoming stream.

Definition 7 (Function Processing). *Given a resource node s with its corresponding service function $C(t)$ and an event stream described by the arrival function $R(t)$ being processed by s, we then have*

$$C'(t) = C(t) - R'(t)$$

$$R'(t) = \min_{0 \le u \le t} \{R(u) + C(t) - C(u)\}$$

where $C'(t)$, $R'(t)$ denote the remaining service function of the resource node and the amount of computation delivered to the processed event stream, respectively.

The first equation just states, that the remaining amount of computation $C'(t)$ (for example in terms of the number of instructions) available until time t is the initial amount reduced by the amount spent for the processed events. In order to understand the meaning of the second equation, note that $R'(t)$ is the maximum value that satisfies $R'(t) - R(u) \le C(t) - C(u)$ for any $u \le t$. The left hand side denotes the number of events that arrived after time u and are processed before time t. This is clearly smaller than the available computing resource in the interval $[u, t]$.

Figure 4 shows the processing network view of a simple resource node. As we are interested in the processing of whole classes of input streams, we will now describe the processing of event streams in terms of the upper and lower curves.

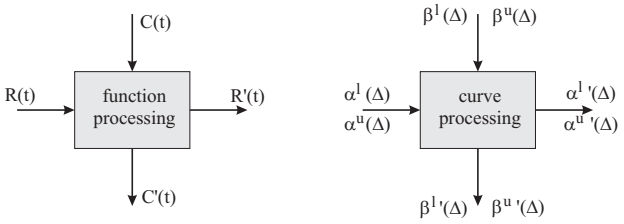

Fig. 4. Diagrams which show the processing of event streams by resource nodes.

These results substantially generalize and sharpen the bounds obtained in [29, 8], as *both* lower *and* upper bounds are involved.

Proposition 1 (Curve Processing). *Given an event stream described by the arrival curves $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ and a resource node described by the service curves $\beta^l(\Delta)$ and $\beta^u(\Delta)$, then the following expressions bound the remaining service function of the resource node and the arrival function of the processed event stream:*

$$\alpha^{l\prime}(\Delta) = \min_{0 \le u \le \Delta} \left\{ \alpha^l(u) + \beta^l(\Delta - u), \beta^l(\Delta) \right\}$$

$$\alpha^{u\prime}(\Delta) = \min_{0 \le u \le \Delta} \left\{ \max_{v \ge 0} \left\{ \alpha^u(u+v) - \beta^l(v) \right\} + \beta^u(\Delta - u), \beta^u(\Delta) \right\}$$

$$\beta^{l\prime}(\Delta) = \max_{0 \le u \le \Delta} \left\{ \beta^l(u) - \alpha^u(u) \right\}$$

$$\beta^{u\prime}(\Delta) = \max_{0 \le u \le \Delta} \left\{ \beta^u(u) - \alpha^l(u) \right\}$$

Figure 5 shows the application of Proposition 1 to the lower and upper arrival and service curves given in Figure 3.

Using well known results from the area of communication networks, see e.g. [8], the bounds derived in Proposition 1 can be used to determine the maximal delay of events and the necessary memory required to store waiting events. The number of events still waiting to be processed at time t is $R(t) - R'(t)$. The delay that an event entering the resource node at time t will experience can be given by $d(t) = \min\{\tau \ge 0 : R(t) \le R'(t+\tau)\}$. The following two equations give bounds on both quantities:

$$d(t) \le \max_{u \ge 0} \left\{ \min\{\tau \ge 0 : \alpha^u(u) \le \beta^l(u+\tau)\} \right\}$$

$$R(t) - R'(t) \le \max_{u \ge 0} \{\alpha^u(u) - \beta^l(u)\}$$

In other words, the delay can be bounded by the maximal horizontal distance between curves $\alpha^u(\Delta)$ and $\beta^l(\Delta)$ whereas the backlog is bounded by the maximal vertical distance between them.

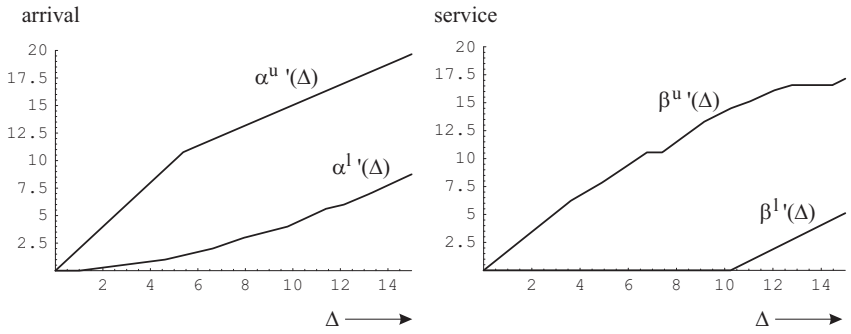

Fig. 5. Processing of the curves given in Figure 3 in accordance with Proposition 1.

In [23, 24] it has been shown that the above event model generalizes some recent attempts to describe bursty tasks in real-time systems, e.g. [21, 4]. In addition, the above bounds lead to schedulability tests which are equivalent to those known in case of fixed priority and earliest deadline first scheduling, e.g. [18, 17, 2].

Based on these results, we can now formulate different problems in the network packet processing domain on a unified basis. As a simple illustration of the above results we consider a fixed priority scheme in the next subsection.

3.2 Simple Processing Network

Following our main approach, to investigate the effect of a scheduling algorithm, we represent the "scheduling block" in the form of a network consisting of nodes which operate on event streams. In addition, there are (virtual) resource streams which model the available resources, see Figure 4. This way, it is possible to describe and analyze packet scheduling, task scheduling and hierarchical approaches. As a simple example, we consider a fixed priority scheme.

Let us start from the models defined in Definitions 1 to 5. In case of a fixed priority scheme, let us suppose that there is a set of flows $f_1, ..., f_n$ with associated event streams $\overline{R}_1(t), ...\overline{R}_n(t)$ ordered according to decreasing priority, i.e. $\overline{R}_1(t)$ has highest priority. In addition, for each event of flow f_i, a task t_i must be executed on one resource s with associated request $w(t_i, s)$ or w_i in short. The event stream associated to a flow f_i is described by arrival curves $\overline{\alpha}_i^u(\Delta)$ and $\overline{\alpha}_i^l(\Delta)$. The resource node s is characterized by the service curves $\beta_i^u(\Delta)$ and $\beta_i^l(\Delta)$.

Because of the fixed priority scheme, the resource stream serves the flows in the order of decreasing priority by the use of Definition 7 and Proposition 1. In order to have compatible units, we first have to multiply the arrival functions and arrival curves with the request for each event, namely w_i. Correspondingly, the request stream leaving the resource must be divided by w_i. If a unit using these events or packets can start only when the whole task has finished on the

preceding unit, we need to apply the floor-function to the outgoing streams, i.e. $\overline{R}_i'(t) = \lfloor R_i'(t)/w_i \rfloor$. In a similar way, the outgoing arrival curves are transformed according to $\overline{\alpha}_i^{u\prime}(\Delta) = \lceil \alpha_i^{u\prime}(\Delta)/w_i \rceil$ and $\overline{\alpha}_i^{l\prime}(\Delta) = \lfloor \alpha_i^{l\prime}(\Delta)/w_i \rfloor$. These curves correctly bound $\overline{R}_i(t)$ as one can show that $\lfloor a \rfloor - \lfloor b \rfloor \leq \lfloor a - b \rfloor$ and $\lceil a \rceil - \lceil b \rceil \geq \lceil a - b \rceil$. In addition to the relations shown in Proposition 1 (which hold for all flows $1 \leq i \leq n$), we have the following equations which describe the processing of event streams using fixed priority scheduling with preemption:

$$\alpha_i^u(\Delta) = w_i \cdot \overline{\alpha}_i^u(\Delta) \quad , \quad \alpha_i^l(\Delta) = w_i \cdot \overline{\alpha}_i^l(\Delta)$$

$$\overline{\alpha}_i^{u\prime}(\Delta) = \lceil \alpha_i^u(\Delta)/w_i \rceil \quad , \quad \overline{\alpha}_i^{l\prime}(\Delta) = \lfloor \alpha_i^l(\Delta)/w_i \rfloor$$

$$\beta_1^u(\Delta) = \beta^u(\Delta) \quad , \quad \beta_i^u(\Delta) = \beta_{i-1}^{u\prime}(\Delta) \quad \forall 1 < i \leq n \quad , \quad \beta^{u\prime}(\Delta) = \beta_n^{u\prime}(\Delta)$$

$$\beta_1^l(\Delta) = \beta^l(\Delta) \quad , \quad \beta_i^l(\Delta) = \beta_{i-1}^{l\prime}(\Delta) \quad \forall 1 < i \leq n \quad , \quad \beta^{l\prime}(\Delta) = \beta_n^{l\prime}(\Delta)$$

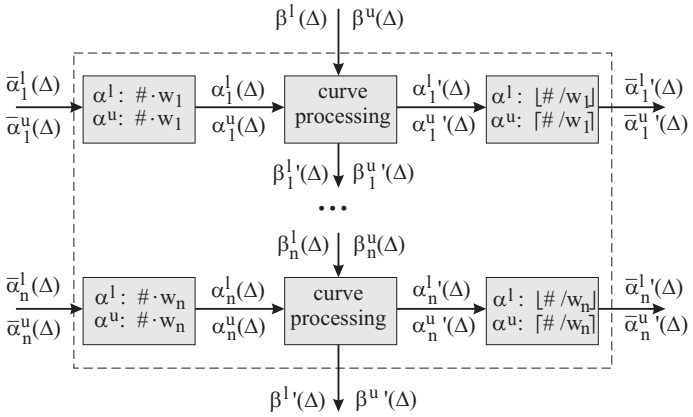

Fig. 6. Diagram showing a processing network for the processing of a set of event streams by a resource node using fixed priority preemptive scheduling.

Finally note, that the remaining resource service $\beta'(\Delta)$ can be used to service other flows with a different scheduling scheme in a hierarchical manner. The event streams $\overline{R}_i'(t)$ can enter further resource nodes which are responsible for executing other tasks $t \in T$ or for performing link scheduling.

In a similar way, one could also describe other scheduling schemes such as Generalized Processor Sharing, Weighted Fair Queueing, First Come First Served and Earliest Deadline First. As it has also been pointed out in the context of communication networks [8], the determination of accurate bounds in these cases is still an active area of research.

Using the results described in the previous section, we can also bound the delay caused by the processing and the necessary memory in terms of the number

packets waiting to be processed. These quantities can be used to determine the necessary memory inside a network processor and the worst-case delay which packets might experience. An application of this technique will be described in Section 5.

4 Task Scheduling in Network Processors

As a first application of the models described in the last two sections, we consider the problem of task scheduling in an NP where different packet processing functions are implemented as programs running on a general-purpose processor. A detailed formal description of the problem is given below; essentially the problem is to schedule the CPU cycles of the processor to process a mix of real-time and non-real-time packets such that all real-time packets meet their deadlines and the non-real-time packets experience the minimum processing delay. There has not been much work on issues related to task scheduling in software-based routers/NPs (see [27]), and most of the previous work on scheduling a mix of real-time and non-real-time tasks is based on CPU reservations (see [5] and the references therein). Our algorithm in this section is on the contrary based mainly on Earliest Deadline First scheduling, and is motivated by algorithms for scheduling a mix of periodic and aperiodic tasks (see [10] and the references therein) and approaches used for packet scheduling (such as [12]).

4.1 Scheduling a Mix of Real-Time and Non-Real-Time Flows

Given a set of flows F, we consider it to be composed of two disjoint subsets F_{RT} and F_{NRT}. All flows $f_i \in F_{RT}$ are real-time flows having finite end-to-end deadlines $d(f_i)$ (see Definition 3). For each packet of f_i the execution of the corresponding task graph $G(f_i)$ must complete within $d(f_i)$ time units after the arrival of the packet. Real-time flows might represent traffic such as voice or video streams. Flows belonging to F_{NRT} have no time constraints (i.e. they have infinite deadlines) and are used to model packet streams corresponding to bulk data transfers such as FTP; we refer to these flows as non-real-time flows. We assume each real-time flow f_i to be constrained by an upper arrival curve α_i^u. The processing cost of each packet of a flow f_k (both real-time and non-real-time) on a single resource s (the CPU) is denoted by $w(f_k)$, where $w(f_k) = \sum_{t \in T(f_k)} w(t, s)$ (see Definitions 1 and 3).

The objective of our scheduling algorithm is multi-fold: (i) to guarantee that all real-time packets meet their associated deadlines, (ii) that the non-real-time packets experience the minimal possible delay, and (iii) we associate with each non-real-time flow f_j a weight ϕ_j and require that the remaining CPU power, given by the lower service curve $\beta^{l'}$, after processing all real-time flows is divided among the non-real-time flows in proportion to their corresponding weights i.e. the number of CPU cycles on the average allocated to flow f_j is $\frac{\phi_j}{\sum_{f_k \in F_{NRT}} \phi_k} \beta^{l'}$. This offers the provision for allocating different non-real-time flows a relative importance.

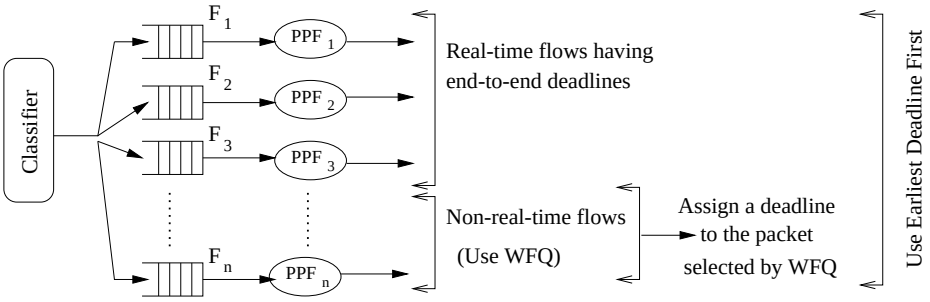

Fig. 7. A new task scheduler based on a hierarchy of WFQ and EDF.

An overview of our scheduling algorithm meeting the above mentioned goals is given in Figure 7. It is composed of a Weighted Fair Queuing (WFQ) [25] scheduling algorithm organized in a hierarchical fashion with an Earliest Deadline First (EDF) scheduler on the top. A WFQ scheduler is based on the Generalized Processor Sharing [15] scheduling algorithm and guarantees that the relative share of the CPU bandwidth among the non-real-time flows follows the proportions dictated by the respective weights associated with them. $F_1, \ldots, F_n$ indicate packet queues (which are filled by the packet classifier) corresponding to the different flows, some of which are real-time and the rest being non-real-time. $PPF_1, \ldots, PPF_n$ denote chains of packet processing functions corresponding to the different flows (as shown in Figure 1).

Given the upper arrival curve α_i^u of each real-time flow f_i, we can compute the processor demand resulting from the set of all real-time flows F_{RT}. Towards this, for each α_i^u we compute a curve $\overline{\alpha}_i^u$ given by

$$\overline{\alpha}_i^u(\Delta) = \begin{cases} 0 & \text{if } \Delta \leq d(f_i) \\ w(f_i)\alpha_i^u(\Delta - d(f_i)) & \text{otherwise} \end{cases}$$

The processor demand by the set of all real-time flows F_{RT} within any time interval of length Δ if all packets have to meet their associated deadlines is then given by a curve α_{RT} where $\alpha_{RT}(\Delta) = \sum_{f_i \in F_{RT}} \overline{\alpha}_i^u(\Delta)$. It is possible to prove (using arguments similar to those given in [11]) that the set of real-time flows F_{RT} is preemptively schedulable on a single processor (such that all packets meet their respective deadlines) if and only if $\beta^l(\Delta) \geq \alpha_{RT}(\Delta)$ for all $\Delta \geq 0$, where the lower service curve β^l is as described in Definition 5. This schedulability test can form the basis for admission control for real-time flows. Now we are in a position to state our algorithm.

Given the set of real-time flows and a lower service curve β^l, we first compute the curve α_{RT} as described above and a curve α_{NRT} which is defined as $\alpha_{NRT}(\Delta) = \min_{t \geq \Delta}\{\beta^l(t) - \alpha_{RT}(t)\}$. The WFQ scheduler shown in Figure 7 computes an ordering of the non-real-time packets according to which they should be processed to respect their relative CPU reservations.

For each packet selected by the WFQ scheduler for processing, if the packet belongs to flow f_i and has a processing requirement of $w(f_i)$ then it is assigned a deadline $d(f_i) = \min\{\Delta : \alpha_{NRT}(\Delta) \geq w(f_i)\}$. Obviously, at any instant of time there is exactly one non-real-time packet that is processed and is assigned a deadline according to the scheme just described. The top level EDF scheduler preemptively schedules this packet along with all the real-time packets (which already have associated deadlines) according to the earliest deadline first scheduling strategy.

Proposition 2 (Schedulability). *If the set of real-time flows is preemptively schedulable (i.e. there exists some scheduling algorithm using which all real-time packets can be processed within their respective deadlines) then our algorithm also schedules the real-time flows such that all deadlines are met.*

Proof. If $d(f_i)$ is the deadline associated with the non-real-time packet having a processing requirement of $w(f_i)$, then for this packet along with all the real-time flows to be schedulable, the following must hold:

$$\beta^l(\Delta) \geq \alpha_{RT}(\Delta) + w(f_i), \quad \forall \Delta \geq d(f_i)$$

which is equivalent to requiring that

$$w(f_i) \leq \beta^l(\Delta) - \alpha_{RT}(\Delta), \quad \forall \Delta \geq d(f_i)$$

The above condition is obviously satisfied if

$$w(f_i) \leq \min_{\Delta \geq d(f_i)} \{\beta^l(\Delta) - \alpha_{RT}(\Delta)\}$$

Hence the proposition follows.

The above scheduling algorithm is preemptive, meaning that the processing of each packet can be preempted at any stage of its task graph and then resumed later. In general such arbitrary preemptions might be costly for any practical implementation, and the only allowable preemption points might be at the end of each node of the task graph (i.e. a packet processing can not be preempted in the middle of executing any node of the corresponding task graph). However, assuming that the execution time of each node is small compared to the total execution time of the whole task graph, the above analysis gives a good approximation of an algorithm where preemption is allowed only at the end of each node.

4.2 Experimental Evaluation

We have implemented and evaluated our algorithm using the Moses tool-suite [22] which is used for modelling and simulation of discrete event systems. Our experimental setup consists of six flows, of which three are real-time and three others are non-real-time flows. Each flow is specified by a TSpec [28] with all its

parameters specified in terms of packets rather than bytes. A TSpec is described by a conjunction of two token buckets and an incoming packet complies with the specified profile only if there are enough tokens in both the buckets (we refer to them as the *avg. bucket* and the *peak bucket* in Table 1). The arrival curves of the different flows were chosen to reflect a typical access network scenario. In particular, the high-volume flows such as video and FTP are bounded by an average rate of around 10 MBits/s. Table 1 gives the specifications of the different flows.

Table 1. Specifications of the real-time (1-3) and non-real-time flows (4-6).

	deadline (Flows 1-3) *WFQ weight* (Flows 4-6) $[ms]$ for Flows 1-3	*avg. bucket* (burstiness, rate) $[pkts, pkts/s]$	*peak bucket* (burstiness, rate) $[pkts, pkts/s]$	*CPU demand* $[cycles]$
Flow 1	2	$(150, 300)$	$(1, 1000)$	40000
Flow 2	10	$(40, 840)$	$(1, 4200)$	600
Flow 3	1	$(3, 300)$	$(1, 1000)$	20000
Flow 4	0.5	$(400, 1000)$	$(1, 5000)$	600
Flow 5	0.2	$(50, 150)$	$(1, 700)$	4000
Flow 6	0.1	$(8, 30)$	$(1, 700)$	40000

Flows 1-3 are real-time flows and Flows 4-6 are non-real-time flows. Flow 1 represents some transactions with encryption, Flow 2 represents a video traffic and Flow 3 some voice encoding. Among the non-real-time flows, Flow 4 represents an FTP download, Flow 5 represents HTTP page downloads, and Flow 6 represents email traffic with encryption.

For the above specified flows, we have compared our algorithm with a scheduling algorithm consisting of a combination of plain EDF for real-time packets and WFQ for non-real-time packets. The ordering of the non-real-time packets is due to the WFQ scheduler and they are processed only when there are no backlogged real-time packets. The real-time packets are scheduled according to EDF scheduling. To avoid any possible confusion (because of the similar terminology and setup), we remind the reader that here we are describing a task scheduling and not a link scheduling algorithm, so the ordering of a packet stream is not used for putting them on the output link, but to execute the packet processing functions in that order.

Figure 8 shows an excerpt of a simulation, comparing the above algorithm with ours. The horizontal-axis shows the simulation time and the vertical-axis represents the delay experienced by a packet in getting processed. Any point on the horizontal-axis represents the completion time of a packet processing task and the corresponding point on the vertical-axis plots the delay experienced (completion time minus the arrival time) by this packet. Note that in the plain EDF + WFQ scheduler, packets from the real-time Flow 2 are processed much before their deadline. In our improved algorithm, the non-real-time Flows 4

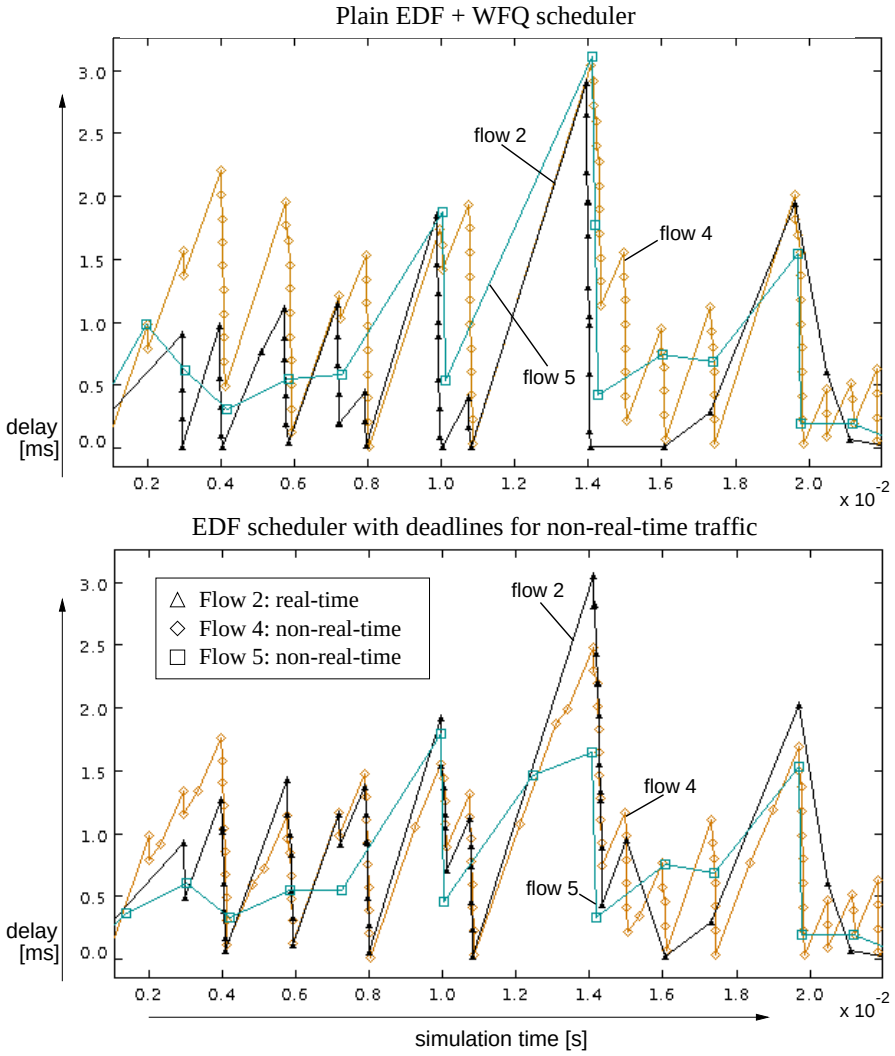

Fig. 8. Comparison of delays experienced by three selected flows.

and 5 experience shorter response times at the expense of slightly higher delays for Flow 2 (which nevertheless meets its deadlines). On the average, the response time of Flows 4 and 5 improve by around 20%, at the expense of the response time of Flows 1, 2, and 3 increasing by around 2%, 32%, and 3% respectively. Due to its high CPU demand, the response time of Flow 6 (not sketched in the figure) does not improve much since the deadlines assigned by our algorithm to packets of this flow fall behind those of the real-time packets, and therefore they are executed after the real-time packets as in the case of the plain EDF + WFQ scheduler.

5 Design Space Exploration

The final section of this paper is devoted to the design space exploration of network processors, where we show how the basic models and methods described in the previous sections can be used to perform an exploration of hardware and software architectures. It is expected that the next generation of network processors will consist of general purpose processing units and dedicated modules for executing run-time extensive functions. Therefore, the purpose of a high level exploration is to select appropriate functional units such that the performance of the processor is maximized under various constraints such as cost, packet delay and power consumption. Here we will concentrate on the following questions:

- How can we estimate the performance of a network processor?
- How can we estimate delay and memory consumption of a hardware/software architecture?

The approach taken here is influenced by our previous results in design space exploration of hardware/software architectures [30]. In particular, we will adopt the model based approach in combination with concepts of multi-objective optimization (see [19] and the references therein).

As described in the last section, we consider the set of flows F to be composed of two disjoint sets F_{RT} and F_{NRT}. The arrival and service curves associated to these flows are interpreted differently. Whereas in case of F_{RT} they describe the rates of the incoming packet streams, they are interpreted as relative rates in case of F_{NRT}. In other words, the rates of the packet streams can be described by the lower and upper arrival curves $\psi \cdot \alpha^l(f)$ and $\psi \cdot \alpha^u(f)$ for $f \in F_{NRT}$ where ψ is a positive scaling variable. The problem of design space exploration can now be formulated as the following multi-objective optimization problem:

> Allocate resource nodes $s \in S$ and bind the tasks $t \in T$ of the flows $f \in F$ to the allocated resource nodes such that ψ is maximized, other criteria such as cost, memory and power consumption are minimized and the deadlines $d(f)$ associated to the flows are satisfied.

The rationale for this performance criterion is based on the fact, that real-time flows such as video and voice streams often have a fixed rate which must be handled by the network processor. On the other hand, the throughput of other flows such as FTP, web traffic or email should be maximized. The relative arrival curves determine the relative weights for these kinds of flows.

For the network processor architecture we assume a heterogeneous set of components consisting of RISC processors, digital signal processors, micro-controllers and dedicated units for compute intensive tasks such as header parsing, table look-up and encryption/decryption. The purpose of the allocation is to select the "right" subset of this modules. In Definition 2, a function *cost* was defined which models the cost of allocating the corresponding unit. It may be noted, that much more complex measures could be added to tasks and resource nodes such as power consumption, and program and data memory. The following definition extends Definition 2 by formally defining the terms allocation and binding.

Definition 8 (Allocation and Binding). *The set $A \subseteq S$ denotes the set of allocated resource nodes $s \in A$. The binding of a task $t \in T$ to a resource $s \in S$ is a relation $B \subseteq T \times S$ where $B \subseteq M$ (see Definition 2), i.e. $(t, s) \in B$ if task t is executed on resource s.*

Based on our simple model for design space exploration of network processors, we can now determine the cost of an implementation as $Cost = \sum_{s \in A} cost(s)$. The problem of minimizing $Cost$ might involve a trade-off between several conflicting criteria such as memory and power requirements, and performance issues. Among the possible candidates for solving such a multi-objective optimization problem, where we are interested in obtaining all the trade-off or the so called Pareto points, are branch-and-bound methods and evolutionary algorithms [13].

For the purpose of illustrating the potential of the formal model introduced in this paper, we solved the design space exploration problem for the task structure shown in Figure 1 using a branch-and-bound search algorithm. It is based on a specification of the whole problem in the form of integer linear equations (see [20]). Only a portion of the resource structure used is shown in Figure 2. There are end-to-end deadlines d for the two real-time flows *RT Send* and *RT Receive*. The network processor architecture was designed for a 10-times higher rate of packets from the flow *NRT-IP-Forward* compared to each of the two other non-real-time flows. We measured *performance* by the scaling variable ψ which is maximized in the integer linear program. The total cost is the sum of costs of the allocated resource nodes and was introduced as a constraint into the ILP. An overview of the optimization setup is shown in Figure 9.

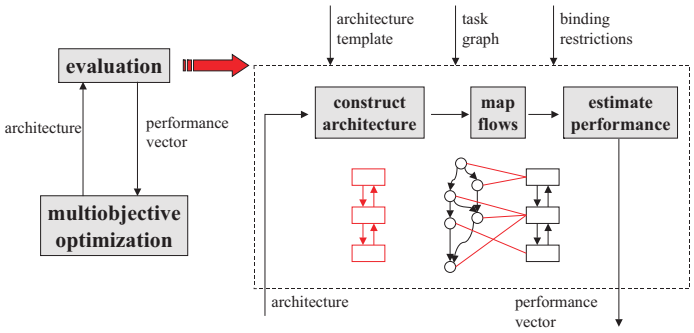

Fig. 9. Strategy of the design space exploration based on a multi-objective search algorithm and an estimation phase.

All the equations were generated using well known techniques [20, 26] and the nonlinear function for taking task scheduling into account was modelled using piecewise linear approximations. Figure 10 shows the result of the design space exploration where the right hand side shows the different pareto points

corresponding to the different architectures and the left hand side gives the corresponding resource usages. Note that the hardware units are not loaded to 100% because of the end-to-end delay constraints.

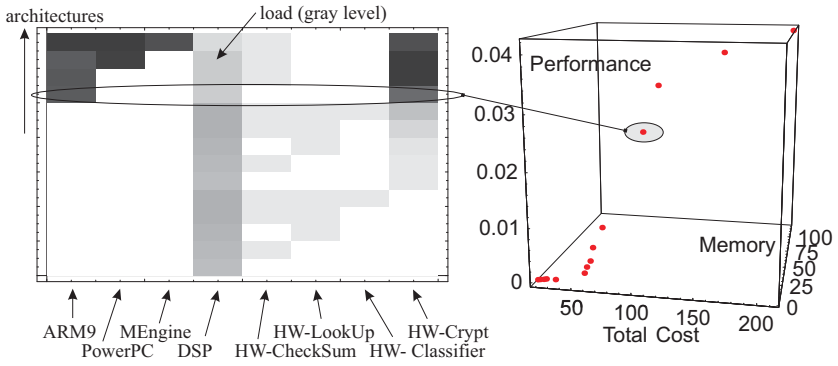

Fig. 10. Results from a design space exploration run. The left hand side shows in a diagram the hardware resources allocated for the Pareto points that have been found. The columns are related to the resource nodes and the rows correspond to the Pareto points, ordered with respect to the performance ψ. The gray level corresponds to the load of the respective hardware unit. The right hand side diagram represents the Pareto points in the performance-memory-cost coordinate system.

The above example took 521 variables, used 544 inequalities and required about 4 seconds per Pareto point on a 500 MHz Pentium III. We used the commercial branch and bound ILP solver CPLEX and Mathematica for generating the input and processing the output.

Scheduling was done in the simplest possible way, in particular using FCFS queuing for each resource, which obviously gives rough estimates only. Here it is possible to use the results obtained in Section 3.2. Let us suppose that the tasks $t \in T(f_i)$ of some flow f_i are assigned to allocated resource nodes $s \in A$, i.e. $(t, s) \in B$. In addition, let us suppose that the task graph $G(f_i)$ is a simple chain. Then we can model the flow of events and resources in form of a computation network (see Figure 11). In particular, we have two different kinds of streams through the network, namely event streams caused by the packet streams entering the system and resource streams determined by the service curves of the resource nodes. Each distinct input flow is mapped onto a directed path through the communication network. To each edge in the communication network, either lower and upper arrival curves or lower and upper service curves are assigned. Using the relations derived in Section 3.2, these quantities can be determined, unless there are no directed cycles for any flow f_i in the network. Finally, interesting performance measures such as delay and memory can be determined using the estimations on backlog and delay as given in Section 3.1.

More work needs to be done in this direction to exploit the the full potential of the approach described in Section 3.2.

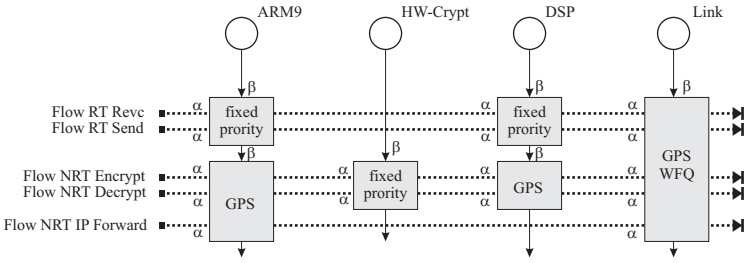

Fig. 11. Example of a simple processing network that is used to operate on packet streams such as those specified in the task graph in Figure 1. A final last resource node may model the link scheduler for the outgoing packet streams.

6 Concluding Remarks

We have shown that the design of embedded packet processing devices such as network processors pose interesting research issues related to models of computation, processing and communication networks, task and packet scheduling and design space exploration. Our approach was based on a unified modelling of these aspects.

It must be noted however, that there are still many open research issues related to the above approach. In particular, the full potential of the chosen task, resource and stream processing specification must still be explored.

References

1. R. Agrawal, R. L. Cruz, C. Okino, and R. Rajan. Performance bounds for flow control protocols. *IEEE/ACM Transactions on Networking*, 7(3):310–323, 1999.
2. N. Audsley, A. Burns, M. Richardson, and A. Wellings. Fixed priority preemptive scheduling: A historical perspective. *Real-Time Systems*, 8:173–198, 1995.
3. F. Baccelli, G. Cohen, G.J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity*. John Wiley, Sons, New York, 1992.
4. S.K. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks, 2001. To appear in Real-Time Systems.
5. A. Bavier and L. Peterson. BERT: A scheduler for best effort and real-time tasks. Technical Report TR-602-99, Department of Computer Science, Princeton University, 2001. Revised in January 2001.
6. T. Blickle, J. Teich, and L. Thiele. System-level synthesis using evolutionary algorithms. *Design Automation for Embedded Systems*, 3(1):23–58, 1998.
7. J.Y. Le Boudec. Application of network calculus to guaranteed service networks. *IEEE Trans on Information theory*, 44(3), May 1998.
8. J.Y. Le Boudec and P. Thiran. *Network Calculus - A Theory of Deterministic Queuing Systems for the Internet*. LNCS 2050, Springer Verlag, 2001.

9. G.C. Buttazzo. *Hard Real-Time Computing Systems - Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 1997.
10. G.C. Buttazzo and F. Sensini. Optimal deadline assignment for scheduling soft aperiodic tasks in hard real-time environments. *IEEE Transactions on Computers*, 48(10):1035–1052, 1999.
11. S. Chakraborty, T. Erlebach, and L. Thiele. On the complexity of scheduling conditional real-time code. In *Proc. 7th International Workshop on Algorithms and Data Structures (WADS)*, LNCS 2125, 2001.
12. R.L. Cruz. A calculus for network delay. *IEEE Trans. Information Theory*, 37(1):114–141, 1991.
13. K. Deb. *Multi-objective optimization using evolutionary algorithms*. John Wiley, Chichester, 2001.
14. D. Decasper, Z. Dittia, G.M. Parulkar, and B. Plattner. Router plugins: A software architecture for next-generation routers. *IEEE/ACM Transactions on Networking*, 8(1):2–15, 2000.
15. A. Demers, S. Keshav, and S. Shenkar. Analysis and simulation of a fair queueing algorithm. *Journal of Internetworking Research and Experience*, 1(1):3–26, 1990.
16. E. Kohler, R. Morris, B. Chen, J. Jannotti, and M.F. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.
17. J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm. In *Proc. IEEE Real-Time Systems Symposium*, pages 166–171, 1989.
18. C. Liu and J. Layland. Scheduling algorithms for multiprogramming in hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.
19. L. Thiele M. Eisenring, E. Zitzler. Handling conflicting criteria in embedded system design. *IEEE Design & Test of Computers*, 17(2):51–59, 2000.
20. G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill International Editions, New York, 1994.
21. A.K. Mok and D. Chen. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10):635–645, 1997.
22. The Moses project homepage. http://www.tik.ee.ethz.ch/~moses/.
23. M. Naedele, L. Thiele, and M. Eisenring. General task and resource models for processor task scheduling, 1998. TIK Report 45, ETH Zürich.
24. M. Naedele, L. Thiele, and M. Eisenring. Characterizing variable task releases and processor capacities. In *Proceedings of the 14th IFAC World Congress*, 1999.
25. A.K. Parekh and R.G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, 1993.
26. S. Prakash and A.C. Parker. Synthesis of application-specific multiprocessor systems including memory components. In *Proc. IEEE Application Specific Array Processors*, 1992.
27. X. Qie, A. Bavier, L. Peterson, and S. Karlin. Scheduling computations on a software-based router. In *Proc. SIGMETRICS*, 2001.
28. S. Shenker and J. Wroclawski. General characterization parameters for integrated service network elements. RFC 2215, IETF, September 1997.
29. L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systeme. In *Proc. IEEE Internation Conference on Circuits and Systems*, 2000.
30. E. Zitzler, J. Teich, and S.S. Bhattacharyya. Evolutionary algorithms for the synthesis of embedded software. *IEEE Transactions on VLSI Systems*, 8(4):452 – 456, August 2000.