An Open Computing Resource Management Framework for Real-Time Computing

Vuk Marojevic, Xavier Revés, and Antoni Gelonch

Dept. of Signal Theory and Communications, Universitat Politècnica de Catalunya,
C/ Jordi Girona 1-3, Modul D4, 08034 Barcelona, Spain
{marojevic,xavier.reves,antoni}@tsc.upc.edu

Abstract. This paper introduces an open computing resource management framework for real-time computing systems. The framework is modular and consists of a general computing resource modeling that facilitates a policy-based (open) computing resource management. The computing resource modeling contains two resource model templates, which may be instantiated as often as necessary to capture a platform's computing resources and an application's computing requirements. The computing resource management approach features a parametric algorithm (t_w-mapping with window size w) and a generic and parametric cost function, which implements the computing resource management policy. We present simulations using a simple instance of this cost function to demonstrate the suitability and versatility of the framework. We compute a metric that relates the computing resource management success to its complexity and conclude that adjusting the cost function's parameter is more efficient than augmenting the t_w-mapping's window size.

Keywords: computing resource management, real-time computing, open framework.

1 Introduction

Many applications require huge amounts of computing resources. Multimedia applications or mobile communications systems, for example, need high processing powers for real-time data processing. Moreover, applications are often personalized for a particular user or user group, which demands more and more sophisticated services. This includes communication services but also other types of popular services, such as videostreaming. The solution to these computing demands is *multiprocessing*, where applications are processed in parallel on arrays of processors that offer much higher processing powers than single-processor execution environments. A single application can, generally, be parallelized and may then be executed together with other applications. In software-defined radio (SDR) [1], for example, a single-user mobile terminal would normally execute only a few applications, the radio and user applications in the most basic case, whereas a multi-user base station serves many users at a time and thus executes many applications concurrently.

Parallel or multiprocessing is more complicated than sequential processing because the available resources need to be shared spatially and temporally. A single processor

P. Sadayappan et al. (Eds.): HiPC 2008, LNCS 5374, pp. 169–182, 2008.

executes applications sequentially, where pseudo-parallelism is achieved through assigning processing time slots to different applications or application's parts. Multiprocessor execution environments, on the other hand, allow for distributed computing, enabling true parallelism. The distributed resources though require an appropriate computing resource management. Computing resources include processing powers, inter-processor bandwidths, memory, and energy resources; in general, all those resources that are required for the execution of applications. Their time management is necessary for a real-time execution.

This paper introduces a new approach to real-time computing resource management. It presents a general framework that can efficiently manage the limited computing resources of multiprocessor platforms while providing the necessary amount of resources to real-time applications. The framework is not optimized for a specific objective but rather open to different management policies. We call this an *open* computing resource management framework. It is more general than our earlier proposal [2].

This framework bases itself on previous research results (section 2). It is a modular design that consists of two principal modules: the computing system modeling (section 3) and the computing resource management (section 4). The letter is further divided into the mapping algorithm (section 4.1) and the cost function (section 4.2). This modular design, in particular, the independence between the algorithm and its objective (cost function), facilitates exchanging the computing resource management policy. Numerous simulations demonstrate the versatility and suitability of the entire framework (section 5), leading to interesting conclusions that pave the path for future research (section 6).

2 Related Work

This work focuses on real-time computing systems. It particularly addresses real-time capable execution environments of limited computing resources and applications with real-time processing demands. We assume that the system's constraints—the applications' real-time computing requirements and the platforms' limited computing resources—have just to be met. Additional or other objectives, such as speeding up an application (more than strictly necessary to meet the given timing constraints), are thus irrelevant here. The framework accounts for platforms and applications with heterogeneous computing resources and requirements (*heterogeneous computing*).

Related work considers almost any problem and objective in heterogeneous computing. A vast amount of literature particularly addresses the mapping of real-time applications to multiprocessor platforms and the scheduling of processes and data flows. We consider *mapping* and *scheduling* as two complementary computing resource management methods and try to generalize previous efforts in heterogeneous computing, taking advantage of their results and conclusions. Due to space limitations, the following paragraphs detail only a few related contributions.

References [3] and [4] address the problem of optimally allocating periodic tasks, which are subject to task precedence and timing constraints, to processing nodes of a distributed real-time system. The efficient local scheduling of tasks in a real-time multiprocessor system is the topic of [5]. If a task's deadline cannot be met on a

particular processing node, this task can be sent to another node [6]. The task model, which is identical in both papers, accounts for worst case computation times, deadlines, and resource requirements; no precedence constraints are assumed.

Instead of assuming worst-case application requirements, [7] proposes to adapt the resource allocation to face the runtime changes in the application environment. It describes and evaluates models and mechanisms for adaptive resource allocation in the context of embedded high performance applications with real-time constraints. Based on the same principle, [8] presents a mathematical modeling for an adaptive resource management in dynamic application environments. It precisely models fixed hardware—a network of processors—and dynamic, real-time software at different abstraction layers. It also proposes a framework for allocation algorithms, supporting the three constraints application-host validity, minimum security level, and real-time deadlines, while maximizing the overall utility of the system. Security is a common issue in recent publications, such as [9] which allocates computing resources to real-time and security-constrained parallel jobs.

A list scheduling framework for the run-time stabilization of static and dynamic tasks with hard and soft deadlines, respectively, is described in [10]. It allows for dynamic or static task-to-processor allocations and implements mechanisms that control the degree of resource reclaiming to increase the processor utilization and the response time of the dynamic workload. Reference [11] tackles hard real-time streaming applications in a scenario where jobs enter and leave a particular homogeneous multiprocessor system at any time during operation. It combines global resource allocation (mapping) with local resource provisioning (scheduling).

Other related contributions are [12]-[14]. Although they do not specifically address real-time systems, they deal with particular aspects that this framework adopts. The dynamic level scheduling (DLS) approach [12], for example, accounts for interprocessor communication overheads. It maps precedence-constrained, communicating tasks to heterogeneous processor architectures with limited or irregular interconnection structures. Alhusaini et al. introduce the problem of resource co-allocation, which refers to simultaneous allocations of different types of resources that are shared among applications, and formulate the mapping problem in the presence of co-allocation requirements [13]. Reference [14], finally, introduces a theory for scheduling directed acyclic graphs (DAGs) in internet-based computing. It applies graph theory techniques to model precedence-constrained computing tasks and to derive optimal schedules for different types of DAGs.

3 Computing System Modeling

This section presents a mathematical modeling of computing resources (section 3.1) and requirements (section 3.2). It features resource model templates that can be instantiated as many times as necessary to capture the relevant resources and requirements. We assume the availability of a middleware or hardware abstraction layer, such as [15], that facilitates the information about hardware capacities and software requirements in the units specified below.

3.1 Computing Resources

$\boldsymbol{R^t} \in \mathbb{R}^+ \times \mathbb{R}^+$ represents the template for modeling the computing environment. ($\mathbb{R}^+$ symbolizes non-negative real numbers.) It is an $X(t)$ times $Y(t)$ matrix ($X(t)$, $Y(t) \in \mathbb{N}$),

$$\boldsymbol{R^t} = \begin{pmatrix} R^t_{11} & R^t_{12} & \cdots & R^t_{1,Y(t)} \\ R^t_{21} & R^t_{22} & \cdots & R^t_{2,Y(t)} \\ \vdots & \vdots & \ddots & \vdots \\ R^t_{X(t),1} & R^t_{X(t),2} & \cdots & R^t_{X(t),Y(t)} \end{pmatrix}, \tag{1}$$

where $t \in 1, 2, \ldots, T$ denotes the resource model index and T the number of $\boldsymbol{R^t}$ instances. $\boldsymbol{R^t}$ is apt for characterizing different types of computing architectures and capturing the available computing resources, such as processing powers and inter-processor bandwidths. It is therefore unitless.

$$\boldsymbol{R^1} = \boldsymbol{C} = [C_1, C_2, \ldots, C_N] \text{ MOPS} \tag{2}$$

models the processing powers of processors P_1, P_2, …, P_N in million operations per second (MOPS). It instantiates (1) with $X(1) = 1$, $Y(1) = N$, and $R^1_{1i} = C_i$. Without loss of generality, we label devices in order of decreasing processing capacities, that is, $C_1 \geq C_2 \geq \ldots \geq C_N$.

$$\boldsymbol{R^2} = \boldsymbol{I} = \begin{pmatrix} I_{11} & I_{12} & \cdots & I_{1N} \\ I_{21} & I_{22} & \cdots & I_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ I_{N1} & I_{N2} & \cdots & I_{NN} \end{pmatrix} = \begin{pmatrix} 1 & I_{12} & \cdots & I_{1N} \\ I_{21} & 1 & \cdots & I_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ I_{N1} & I_{N2} & \cdots & 1 \end{pmatrix} \tag{3}$$

represents the logical interconnection model. A logical link corresponds to a directed (unidirectional) communication line between a pair of processor. These logical links map to physical links: A logical link between any two processors maps to a physical link of a certain bandwidth if the two processors are actually connected; otherwise it maps to an imaginary physical link of zero bandwidth. Mathematically, $\boldsymbol{I}$ holds the indexes that point to the physical link bandwidths, which we model as

$$\boldsymbol{R^3} = \boldsymbol{B} = [B_1, B_2, B_3, \ldots, B_{N\cdot[N-1]+1}] = [\infty, B_2, B_3, \ldots, B_{N\cdot[N-1]+1}] \text{ MBPS}. \tag{4}$$

B_x, where $x = I_{32}$ for instance, is the maximum bandwidth in mega-bits per second (MBPS) that is available for the directed data transfer from the local data memory of processor P_3 to the local data memory of processor P_2. B_1 models the processor-internal bandwidth capacities, assuming direct memory access (DMA) or pointer transfers for processor-internal data flows.

Equations (3) and (4) facilitate modeling shared or bidirectional buses, mapping the corresponding logical links to a single entry in $\boldsymbol{B}$. Unnecessary positions in (4) are then filled with zeros. $\boldsymbol{I}$ can be organized in such a way that $B_2 \geq B_3 \geq \ldots \geq B_{N\cdot[N-1]+1}$.

This section has presented three instances of (1); one captures the communication architecture ($\boldsymbol{I}$) and two actual computing resources ($\boldsymbol{C}$ and $\boldsymbol{B}$). We introduce $t' \in 1, 2, \ldots, T'$, which indexes the instances of (1) that are actual computing resources. Then,

$t' = 1$ and $t' = 2$ index $\boldsymbol{C}$ and $\boldsymbol{B}$. Modeling the actual computing resources per time unit facilitates handling real-time applications with limited resources (section 4).

3.2 Computing Requirements

Matrix $\boldsymbol{r^t} \in \mathbb{R}^+ \times \mathbb{R}^+$ is the applications' general computing model. It is, equivalently to $\boldsymbol{R^t}$, an $x(t)$ times $y(t)$ matrix ($x(t)$, $y(t) \in \mathbb{N}$):

$$\boldsymbol{r^t} = \begin{pmatrix} r^t_{11} & r^t_{12} & \cdots & r^t_{1,y(t)} \\ r^t_{21} & r^t_{22} & \cdots & r^t_{2,y(t)} \\ \vdots & \vdots & \ddots & \vdots \\ r^t_{x(t),1} & r^t_{x(t),2} & \cdots & r^t_{x(t),y(t)} \end{pmatrix}. \tag{5}$$

We model an application as M processes that process and propagate data. Then we can introduce instances of (5) that correspond to instances (2)–(4) of (1).

$$\boldsymbol{r^1} = \boldsymbol{c} = [c_1, c_2, \ldots, c_M] \text{ MOPS}, \tag{6}$$

particularly, resumes the processing requirements of processes p_1 to p_M. We assume that applications' processing chains represent directed acyclic graphs (DAGs) [12]-[14]; cyclic dependencies are then process-internal. DAGs can be logically numbered: If p_x sends data to p_y, then $y > x$ [16]. This leads to

$$\boldsymbol{r^2} = \boldsymbol{i} = \begin{pmatrix} i_{11} & i_{12} & \cdots & i_{1M} \\ i_{21} & i_{22} & \cdots & i_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ i_{M1} & i_{M2} & \cdots & i_{MM} \end{pmatrix} = \begin{pmatrix} 1 & i_{12} & \cdots & i_{1M} \\ 1 & 1 & \cdots & i_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix} \tag{7}$$

and

$$\boldsymbol{r^3} = \boldsymbol{b} = [b_1, b_2, b_3, \ldots, b_{M\cdot[M-1]/2+1}] = [0, b_2, b_3, \ldots, b_{M\cdot[M-1]/2+1}] \text{ MBPS}. \tag{8}$$

Equation (7) indicates an application's precedence constraints and together with (8) represents the dataflow requirements: b_x, where $x = i_{12}$, for instance, is the minimum bandwidth that is necessary to transmit data from process p_1 to process p_2 in real time. Unreferenced elements in $\boldsymbol{b}$ are filled with zeros. Organizing the upper diagonal of $\boldsymbol{i}$ so that $b_2 \geq b_3 \geq \ldots \geq b_{M\cdot[M-1]/2+1}$ facilitates implementing certain mapping techniques.

Separating the precedence constraints from the bandwidth demands facilitates distinguishing between dependent and independent data flows. For example, if p_1 sends the same data to p_3 and p_4, i_{13} and i_{14} may point to the same entry in $\boldsymbol{b}$ ($i_{13} = i_{14}$ – dependent data flows), whereas if p_1 sends two different data chunks, one to p_3 and one to p_4, i_{13} and i_{14} should point to the different entries in $\boldsymbol{b}$ ($i_{13} \neq i_{14}$ – independent data flows). This paper considers independent data flows.

The processing or bandwidth requirements can be obtained from multiplying the number of operations or bits that need to be processed or propagated by the available time for doing so. This correctly models applications' real-time requirements.

4 Computing Resource Management

4.1 The t_w-Mapping

The t_w-mapping was introduced in [2]. Here we summarize its principal characteristics. It is a windowed dynamic programming algorithm, where w indicates the window size. It is organized through the t_w-mapping diagram (Fig. 1), which contains a trellis of N x M (row x column) t-nodes. A t-node is identified as $\{P_{k(l)}, p_s\}$ and absorbs the mapping of process p_s to processor $P_{k(l)}$. Any t-node at *step* s (column s in the t_w-mapping diagram) connects to all t-nodes at *step* s+1. The sequence of processors $[P_{k(0)}\ P_{k(1)}\ ...\ P_{k(w)}]_s$ identifies the w-path, a path of length w, that is associated with $\{P_{k(1)}, p_s\}$, where $P_{k(0)}$ is the w-path's origin processor at *step* s–1 and $P_{k(w)}$ the destination processor at *step* s+w–1. Table 1 contains the most important variables and expressions that appear in the rest of the paper.

The main feature of the t_w-mapping is that it is cost function independent. That is, any cost function can, in principle, be applied. The cost function guides the mapping process. It is responsible for managing a platform's available computing resources and an application's real-time processing requirements (section 4.2).

The algorithm sequentially pre-assigns, or pre-maps, processes to processors, starting with process p_1 and finishing with process p_M (parts I and II of the t_w-mapping). This is followed by a post processing that determines the final mapping (part III).

Table 1. Ranges and descriptions of variables and expressions

Variable or expression	Range (Argument range)	Description
N	1, 2, …	number of processors
M	1, 2, …	number of processes
w	1, 2, …, M–1	window size
$k(l)$	1, 2, …, N; ($l \in$ 0, 1, …, w)	processor index $k(l)$ with its relative position l in the w-path
$P_{k(l)}$	$P_1, P_2, \ldots, P_N$	processor
s	1, 2, …, M	step index (process index)
p_s	$p_1, p_2, \ldots, p_M$	process
$\{P_{k(l)}, p_s\}$		t-node indicating the mapping of p_s to $P_{k(l)}$
$[P_{k(0)}\ P_{k(1)}\ ...\ P_{k(w)}]_s$	($s \in$ 2, 3, …, M–w+1)	w-path associated with $\{P_{k(1)}, p_s\}$
$h = s + (l - 1)$	($l \neq 0$; $s \in$ 2, 3, …, M–w+1)	step index h substitutes $s + (l - 1)$
$[P_{k(l-1)}\ P_{k(l)}]_h$	($l \neq 0$; $s \in$ 2, 3, …, M–w+1)	edge between $\{P_{k(l-1)}, p_{h-1}\}$ and $\{P_{k(l)}, p_h\}$
$\mathrm{WT}[P_{k(l-1)}\ P_{k(l)}]_h$	($l \neq 0$; $s \in$ 2, 3, …, M–w+1)	edge weight
$\boldsymbol{R}^{t'\,@\{k(l),\,h\}}$		remaining computing resources of type t' at $\{P_{k(l)}, p_h\}$
$\boldsymbol{r}^{t'\,@\{k(l),\,h\}}$		required computing resources of type t' at $\{P_{k(l)}, p_h\}$

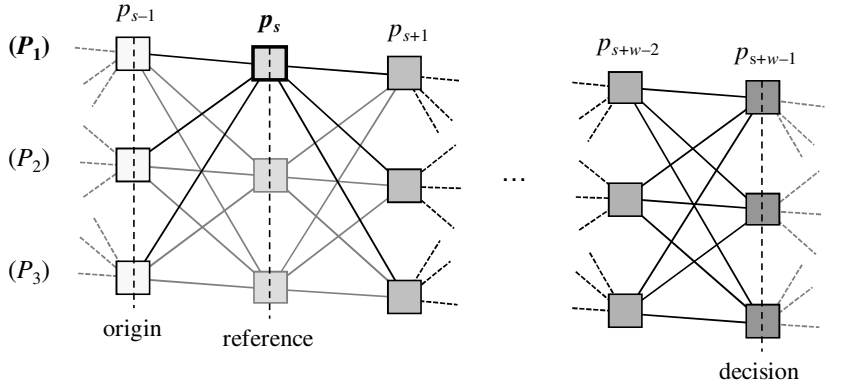


Fig. 1. Extract of the t_w-mapping diagram for three processors ($N = 3$). The black edges indicate those w-paths that are examined at t-node $\{P_1, p_s\}$, assuming $w \geq 3$.

Part I consists of pre-mapping process p_1 to all N processors and storing the pre-mapping costs at t-nodes $\{P_1, p_1\}$ through $\{P_N, p_1\}$. Costs are computed due to some cost function.

At step s of part II ($2 \leq s \leq M–w+1$) the t_w-mapping examines all N^w w-paths that are associated with $\{P_{k(1)}, p_s\}$. These w-paths originate at a t-node at step $s–1$, pass through $\{P_{k(1)}, p_s\}$, and terminate at a t-node at step $s+w–1$. Fig. 1 illustrates this for $P_{k(1)} = P_1$.

In case that $s < M–w+1$, the algorithm highlights the edge between a t-node at step $s–1$ and t-node $\{P_{k(1)}, p_s\}$ that corresponds to the minimum-cost w-path. The minimum-cost w-path is the path that is associated with the minimum accumulated cost due to the corresponding pre-mappings of p_1, p_2, …, and $p_{s+w–1}$, where the w-path's origin t-node provides the pre-mapping information of p_1 to $p_{s–1}$. The algorithm then stores the cost and the remaining resources up to t-node $\{P_{k(1)}, p_s\}$ at $\{P_{k(1)}, p_s\}$. It (simultaneously) processes all t-nodes at step s before considering those at step $s+1$.

If $s = M–w+1$, however, the entire minimum-cost w-path is highlighted. The total cost and finally remaining resources are then stored at $\{P_{k(1)}, p_{M–w+1}\}$. After having processed all N t-nodes at step $M–w+1$, part III of the algorithm follows.

Part III tracks the t_w-mapping diagram back- and forward along the highlighted edges, starting at the t-node at step $M–w+1$ that holds the minimum cost. This process finds the complete mapping solution for the given problem and cost function.

The algorithm's complexity depends on the cost function. Assuming that the complexity of the cost function (*ccf*) is constant throughout the mapping process, we can write

$$\text{complexity}(t_w\text{-mapping}) \approx (M - w) \cdot N^2 \cdot \frac{N^w - 1}{N - 1} \cdot ccf\,. \tag{9}$$

Considering that $M >> w$ and $ccf = 1$, the order of magnitude becomes

$$\text{complexity-order}(t_w\text{-mapping}) = O(M \cdot N^{w+1}). \tag{10}$$

This indicates that the algorithm is not computing efficient for large N. We therefore suggest to cluster (huge) arrays of processors, which will eventually execute many applications, and to apply the t_w-mapping on each cluster.

4.2 Cost Function

This section proposes a generic cost function that manages the available computing resources based on our modeling concept. We define it through the edge weight:

$$\mathrm{WT}[P_{k(l-1)}\ P_{k(l)}]_h = \sum_{t'=1}^{T'} q_{t'} \cdot cost_{t'}^{@\{k(l),h\}}, \tag{11a}$$

where

$$cost_{t'}^{@\{k(l),h\}} = \begin{cases} \sum_{i,j} \dfrac{r_{ij}^{t'@\{k(l),h\}}}{R_{i'j'}^{t'@\{k(l),h\}}}, & \text{if } \dfrac{r_{ij}^{t'@\{k(l),h\}}}{R_{i'j'}^{t'@\{k(l),h\}}} \le 1 \ \ \forall i,j \\ \infty, & \text{otherwise} \end{cases}. \tag{11b}$$

$[P_{k(l-1)}\ P_{k(l)}]_h$ represents any edge in the t_w-mapping diagram; it is for $w > 1$ part of a w-path. Each summand in (11a) stands for the weighted cost for the allocations of resource type t' at t-node $\{P_{k(l)}, p_h\}$. Equation (11b) defines this cost as the sum of ratios between the required resources ($r_{ij}^{t'}$) and the corresponding remaining ones ($R_{i'j'}^{t'}$) at $\{P_{k(l)}, p_h\}$. This implies a dynamic resource update.

Each ratio between a resource requirement and its availability is either less, equal, or greater than 1. In the latter case, it is mapped to infinity (11b). Hence, assuming $q_{t'} \neq 0 \,\forall\, t'$, the weighted sum in (11a) returns either a finite or infinite value. A finite edge weight indicates a feasible, an infinite an infeasible pre-mapping. This permits identifying and discarding infeasible solutions, which cannot meet the system's real-time computing constraints.

Cost function (11) defines the computing resource management policy through parameter $\boldsymbol{q}$, where $\boldsymbol{q} = [q_1, q_2, \ldots, q_{T'}]$ weights the cost terms. The higher $q_{t'}$ the higher the relative importance of resource type t'. The sum of all weights can be normalized to 1. Then, $q_1 = q_2 = \ldots = q_{T'} = 1/T'$ would mean equally weighted cost terms.

4.3 Scheduling

On the basis of a feasible mapping—a mapping of finite cost—N processor-local schedulers need to schedule processes, data transfers, and possibly other resource allocations to guarantee that real-time constraints will finally be met. Finding such schedules is possible if we assume that processing chains can be pipelined, that data processing and data transfers can overlap, and that partial results can be immediately forwarded to the next process [2].

Access to any shared resource requires its temporal management or scheduling. Each shared link, for example, requires a scheduler. Assuming the availability of data buffers, these schedulers can use a simple policy to ensure timely data transfers: Transfer data immediately to output data buffers. This data is sent to the corresponding input buffer as soon as the bus becomes available, gaining access to the different processors that share the bus in a round-robin fashion.

5 Simulations

5.1 Simulation Setup

The following simulations serve for demonstrating the suitability and possibilities of the framework. Due to space limitations we consider a single computing platform and two computing resources ($T' = 2$)—processing powers ($t' = 1$) and inter-processor bandwidths ($t' = 2$)—based on the system model instances of section 3. Fig. 2 shows the computing platform model. This platform may represent a stand-alone computing cluster of three heterogeneous devices or be connected to an array of processors.

We randomly generate 100,000 DAGs, which model different applications, where

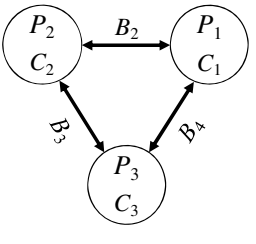
- the number of processes is $M = 25$,
- process p_i is connected to p_j with a probability of 0.2 if $j > i$ (we allow disconnected subgraphs, modeling parallel chains, but connect any isolated node to its next neighbor),
- the processing demands are uniformly distributed in [1, 2, ..., 600] MOPS, and
- the bandwidth demands are uniformly distributed in [1, 2, ..., 500] MBPS.

These parameters have been derived from a real SDR application [2]. Nevertheless, this random DAG generation results in many different application topologies (precedence constraints) and computing requirements. The mean processing requirement is 7429.8 MOPS, which is slightly less than $25 \cdot (600+1)/2 = 7512.5$ MOPS, because we discard applications which need more than the 9000 available MOPS (Fig. 2). 7429.8 MOPS correspond to 82.6 % of the platform's total processing resources. An application's total bandwidth requirement is 15,069.2 MBPS in the mean, being 167.4 % of the available inter-processor communication bandwidths.

5.2 Results I: Ordering

The t_w-mapping with cost function (11) maps computing requirements to computing resources. It does so sequentially, starting with process p_1 and finishing with process p_M (section 4). Related work demonstrated the importance of the mapping or scheduling order. Reference [12], for instance, assigns dynamic levels to determine the next process to be scheduled.

The modeling of section 3.2 facilitates the reordering or relabeling of processes through basic matrix operations. In particular, to change process p_i for p_j, exchange c_i and c_j in (6) and switch rows and columns i and j in (7). Switching rows i and j in (7)



$\boldsymbol{C} = (4000, 3000, 2000)$ MOPS

$$\boldsymbol{I} = \begin{pmatrix} 1 & 2 & 4 \\ 2 & 1 & 3 \\ 4 & 3 & 1 \end{pmatrix}$$

$\boldsymbol{B} = (\infty, 3000, 3000, 3000, 0, 0, 0)$ MBPS

Fig. 2. Computing platform model

changes the successors of p_i for those of p_j and vice versa, whereas switching columns i and j changes the predecessors of p_i for those of p_j and vice versa. This maintains the same DAG with another labeling of processes.

Here we study static reordering techniques. Dynamic reordering of the remaining processes to be mapped will be examined in future work. We consider 3 approaches:

- no reordering (ORD-0),
- reordering by decreasing processing requirements (ORD-C), and
- reordering by decreasing bandwidth demands (ORD-B).

ORD-0 assumes the initial order based on a logical numbering, which is generally not unique. ORD-C leads to $c_1 \geq c_2 \geq \ldots \geq c_M$, whereas in case of ORD-B, the pair of processes with the heaviest data flow demand become p_1 and p_2, the next highest bandwidth requirement specifies p_3 (and p_4), and so forth. The flexibility of mapping lower computing requirements to the remaining computing resources is the reason for choosing ORD-C or ORD-B. Simulations will show which approach is more suitable for the considered scenario.

Table 2 shows the t_w-mapping results for three values of q_1. We observe that ORD-B and ORD-C outperform ORD-0. This can be explained as follows: Bandwidth resources, as opposed to processing resource, can be saved if (heavily) communicating processes are mapped to the same processor. (This is why we can map applications that have a higher total bandwidth requirement than the platform's total inter-processor bandwidth capacity.) Saving bandwidths is only possible if there is a processor with sufficient processing capacities for executing two communicating processes. If heavily communicating processes are considered first (ORD-B), it is most probable that heavy links can be solved processor-internally. Correspondingly, the flexibility of mapping lower processing requirements can potentially merge heavily communicating processes and explains the good behavior of ORD-C.

Additional simulations have shown that ORD-C is more suitable than ORD-B if the processing resources are the bottleneck, whereas ORD-B performs better than ORD-C if the bandwidth requirements are dominating. Here, the high processing and bandwidth loads (section 5.1) explain the similar performances of ORD-C and ORD-B. Since the best results are obtained for $q_1 = 0.7$ and ORD-B (Table 2), the following simulations apply the ORD-B algorithm before executing the t_w-mapping.

Table 2. Results I

	$w = 1$			$w = 3$		
q_1	ORD-0	ORD-C	ORD-B	ORD-0	ORD-C	ORD-B
0.3	33.73	15.39	16.12	19.06	7.39	8.95
0.5	25.07	11.48	11.31	13.61	6.05	6.05
0.7	21.97	13.09	9.92	11.71	6.58	5.13

5.3 Results II: Cost Function Parameter *q* vs. Window Size *w*

Methods. First we consider a fixed q vector for all 100,000 DAGs. We examine $q_1 = 0.1, 0.2, \ldots, 0.9$ to obtain the optimal q in the mean (Method A). Then we propose to choose q_1 dynamically, trying different values until either a feasible mapping is found

or all values have been examined. In particular, q_1 is iteratively updated in the following order: 0.5, 0.4, 0.6, 0.3, 0.7, 0.2, 0.8, 0.1, 0.9 (Method B). This is a simple method that considers q_1 at a granularity of 0.1, starting with equally weighted cost terms and discarding single-term cost functions. The number of iterations, or mapping intents per application, is then between 1 (for a successful mapping with $q_1 = 0.5$) and 9 (if $q_1 = 0.9$ is finally examined). The mean number of iterations (*m-iter*) is the number of iterations averaged over the considered DAGs.

Metric. In order to compare the two methods and to formalize the significance of the cost function parameter q versus the window size w, we introduce

$$\textit{metric-I} = \textit{quality} / \textit{complexity}. \tag{12}$$

This metric relates the quality of the computing resource management approach to its computing complexity. It may be considered an efficiency indicator, because efficiency indicates good results at little effort.

Here we define *quality* as follows: If the algorithm fails in mapping x % of the applications, its *quality* is $1/x$. We measure the complexity in two ways, theoretically and practically. In both cases we count the number of multiply-accumulate operations (MACs), where 1 MAC stands for one multiplication or division followed by a summation.

The theoretical complexity for the given two-term cost function can be approximated as

$$\textit{theoretical-complexity} \approx (M - w) \cdot N \cdot \left\{ \sum_{k=1}^{w} N^k \cdot \frac{M - w + 2k + 1}{2} + 2N^w \right\}. \tag{13}$$

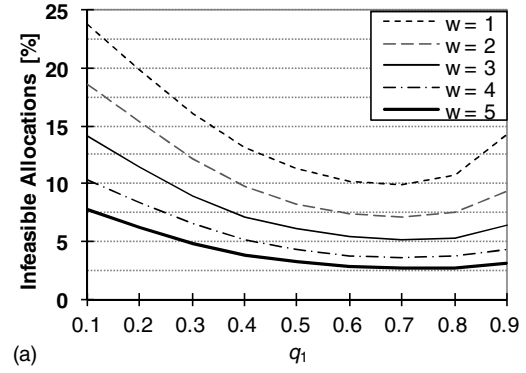
The different terms in (13) are:

- $(M - w)$ is the number of steps that pertain to part II of the t_w-mapping,
- N is the number of t-nodes per step, and
- $\{\cdot\}$ represents the complexity of computing the cost due to processing and bandwidth requirements at a t-node of part II, where $2N^w$ are the additional 2 MACs for multiplying each w-path's cost term with q_1 and q_2.

Equation (13) approximates the theoretical maximum complexity. It accounts for fully connected DAGs, dividing the bandwidth requirement of each possible link between processes by the corresponding finite or infinite bandwidth capacity.

The theoretical complexity for method B is computed as *m-iter* times (13). The practical complexity model accounts for code optimizations: The practical complexities are obtained from C-code implementations, counting each MAC that is actually realized.

Results. Fig. 3a shows the percentage of unfeasibly mapped DAGs due to method A as a function of q_1 and w. It clearly indicates that the mapping success is a function of the window size. We further observe that the lowest number of infeasible mappings is achieved with $q_1 = 0.7$ for any w. The number of infeasible allocations is two to three times lower with $q_1 = 0.7$ than it is with the least favorable q_1, which is $q_1 = 0.1$. This justifies Method B's order of examining the different q_1 values. (Another order would



w	Infeasible allocations [%]	*m-iter*
1	3.79	1.53
2	2.64	1.38
3	1.94	1.28
4	1.42	1.21
5	0.98	1.15

(b)

Fig. 3. Results II with method A (a) and method B (b)

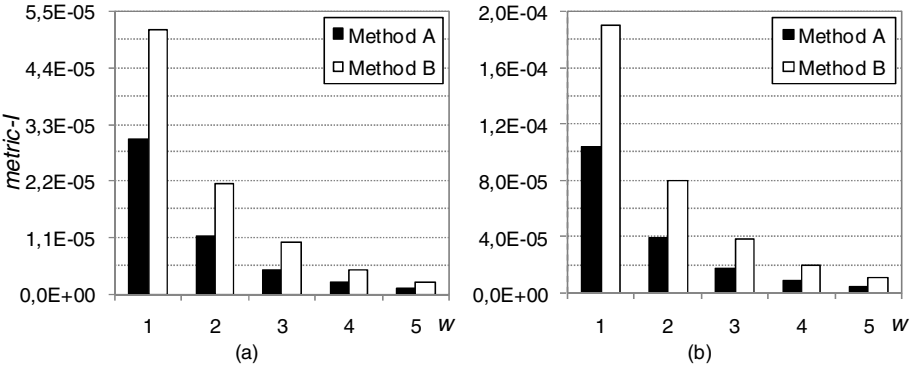


Fig. 4. *metric-I* for methods A and B based on theoretical (a) and practical (b) complexities

affect *m-iter* but not the percentage of infeasible allocations.) Fig. 3b shows the corresponding results. We again observe that the higher w the fewer the number of infeasible mappings. The mean number of iterations decreases correspondingly. It is generally low because more than 87.5 % of the DAGs are successfully mapped for $q_1 = 0.5$ (Fig. 3a), that is, after 1 iteration.

Fig. 3 shows that choosing $\boldsymbol{q}$ dynamically significantly reduces the number of infeasible results. This leads to the conclusion that $\boldsymbol{q}$ should be chosen on application basis (Method B) but also reinforces its importance within cost function (11).

We compute *metric-I* to formally compare the two methods and to discuss the dynamic selection of $\boldsymbol{q}$ versus the increase of w. Fig. 4 illustrates the results as a function of w for both, the theoretical and practical complexities. The practical complexities and the qualities for method A are based on $\boldsymbol{q} = [0.7, 0.3]$.

We observe that *metric-I* based on theoretical complexity numbers (Fig. 4a) is qualitatively equivalent to *metric-I* for practical complexities (Fig. 4b). We interpret this as a validation of the theoretical and practical complexity models.

Fig. 4 shows that searching for a suitable $\boldsymbol{q}$ on application basis—even with the basic $\boldsymbol{q}$-selection algorithm of method B—approximately doubles the proposed metric for any w. This metric considerably decreases with w; adjusting $\boldsymbol{q}$ is thus more interesting

than increasing w. In particular, if the quality of the t_w-mapping results for some w and $\boldsymbol{q}$ is insufficient, we can improve it trying other $\boldsymbol{q}$ vectors without sacrificing efficiency. For example, a mapping success of 95 % is achieved with $w = 1$ in case of method B (Fig. 3b), whereas method A needs at least a window size of 3 (Fig. 3a). Relating the corresponding values of *metric-I*, we obtain a difference of one order of magnitude in favor of method B (Fig. 4). On the other hand, we argue for a complementary adjustment of both parameters. For instance, to achieve less than 3 % infeasible allocations, apply method B with $w = 2$ (instead of method A with $w = 5$).

The above conclusions demonstrate the suitability of the two parameters w and $\boldsymbol{q}$ but also validate *metric-I*. These conclusions are valid for the above simulations. Other problems may behave differently and so may require similar simulations to derive corresponding conclusions and appropriate parameter adjustments.

6 Conclusions

This paper has introduced a computing resource management framework for real-time systems. It consists of a modular design, which features systematically extensible computing system models and an open computing resource management approach. This approach comprises the t_w-mapping—a cost function independent mapping algorithm—and a generic cost function, which manages the available computing resources of any type to satisfy the applications' real-time execution demands.

The simulations have demonstrated the suitability of the entire framework as well as the significance of the two independent parameters w and $\boldsymbol{q}$; the proper adjustment of these parameters can significantly enhance the efficiency of the computing resource management. There is still room for improvement: A low-complex algorithm that dynamically reorders the remaining processes to be pre-mapped or dynamic adjustments of parameters w and $\boldsymbol{q}$ throughout the t_w-mapping process may further increase *metric-I*. We will investigate these issues as well as simulate scenarios with additional computing resource types, such as memory and energy.

This work is focused on real-time computing systems, where the objective is to meet real-time execution demands with limited computing resources. We are currently examining how to adapt the framework to other types of systems and objectives.

Acknowledgments. This work was supported by the Spanish National Science Council CYCIT under Grant TEC2006-09109, which is partially financed from the European Community through the FEDER program.

References

1. Mitola, J.: The software radio architecture. IEEE Commun. Mag. 33(5), 26–38 (1995)
2. Marojevic, V., Revés, X., Gelonch, A.: A computing resource management framework for software-defined radios. In: IEEE Trans. Comput. 57(10), 1399–1412 (2008)
3. Peng, D.-T., Shin, K.G., Abdelzaher, T.F.: Assignment and scheduling communicating periodic tasks in distributed real-time systems. IEEE Trans. Software Eng. 23(12), 745–758 (1997)

4. Hou, C.-J., Shin, K.G.: Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. IEEE Trans. Comput. 46(12), 1338–1356 (1997)
5. Ramamritham, K., Stankovic, J.A., Shiah, P.-F.: Efficient scheduling algorithms for real-time multiprocessor systems. IEEE Trans. Parallel Distrib. Syst. 1(2), 184–194 (1990)
6. Ramamritham, K., Stankovic, J.A., Zhao, W.: Distributed scheduling of tasks with deadlines and resource requirements. IEEE Trans. Comput. 38(8), 1110–1123 (1989)
7. Rosu, D., Schwan, K., Yalamanchili, S., Jha, R.: On adaptive resource allocation for complex real-time applications. In: Proc. 18th IEEE Int'l. Real-Time Systems Symp., pp. 320–329 (1997)
8. Ecker, K., et al.: An optimization framework for dynamic, distributed real-time systems. In: Proc. 17th IEEE Int. Parallel and Distributed Processing Symp. (IPDPS 2003) (2003)
9. Xie, T., Qin, X.: Security-aware resource allocation for real-time parallel jobs on homogeneous and heterogeneous clusters. IEEE Trans. Parallel Distrib. Syst. 19(5), 682–697 (2008)
10. Krings, A.W., Azadmanesh, M.H.: Resource reclaiming in hard real-time systems with static and dynamic workloads. In: Proc. 30th IEEE Hawaii Int'l. Conf. System Sciences (HICSS), pp. 116–625 (1997)
11. Moreira, O., Mol, J.-D., Beckooij, M., van Meerbergen, J.: Multiprocessor resource allocation for hard-real-time streaming with a dynamic job-mix. In: Proc. 11th IEEE Real Time Embedded Technology and Applications Symp. (RTAS 2005), pp. 332–341 (2005)
12. Sih, G.C., Lee, E.A.: A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Trans. Parallel Distrib. Syst. 4(2), 175–187 (1993)
13. Alhusaini, A.H., Prasanna, V.K., Raghavendra, C.S.: A framework for mapping with resource co-allocation in heterogeneous computing systems. In: Proc. 9th Heterogeneous Computing Workshop (HCW 2000), pp. 273–286. IEEE CS Press, Los Alamitos (2000)
14. Malewicz, G., Rosneberg, A.L., Yurkewych, M.: Toward a theory for scheduling DAGs in internet-based computing. IEEE Trans. Comput. 55(6), 757–768 (2006)
15. Revés, X., Gelonch, A., Marojevic, V., Ferrus, R.: Software radios: unifying the reconfiguration process over heterogeneous platforms. EURASIP J. Applied Signal Processing 2005(16), 2626–2640 (2005)
16. Robinson, D.F., Foulds, L.R.: Digraphs: Theory and Techniques. Gordon and Breach Science Publisher Inc. (1980)