Runtime Energy Management under Real-Time Constraints in MPSoCs

André Martins, Marcelo Ruaro, Anderson Santana, Fernando G. Moraes
FACIN - PUCRS – Av. Ipiranga 6681, 90619-900, Porto Alegre, Brazil
{andre.del, marcelo.ruaro, anderson.santana.001}@acad.pucrs.br, fernando.moraes@pucrs.br

***Abstract*—The workload of many-core systems includes real-time (RT) applications. Obtain energy savings while executing RT applications is a challenge due to the RT timing constraints. The techniques used to reduce the consumption, as dynamic voltage and frequency scaling (DVFS), usually delays the applications, leading to constraint violations. Most works in the literature ensure RT constraints based on design-time analysis of the expected workload by defining the voltage and frequency levels. Our main contribution is the proposal of a *Runtime* Energy *Management* for RT applications (RT-REM), using monitored data, and DVFS and task mapping as actuation policies. Supervising the application slack time enables to save energy of RT applications. Evaluations on many-core systems (up to 144 PEs), achieves 18% in energy savings, keeping constraint misses below 2.5%. Our proposal stands out from related works regarding scalability, realistic and accurate energy estimation, and absence of design-time analysis of the application set.**

Keywords—DVFS, energy management, monitoring, real-time applications, MPSoCs

I. Introduction And Related Work

Power management (PM) controls dynamically multiple cores to provide the required performance without wasting energy and power. However, Real-Time (RT) applications have performance constraints to meet. The techniques used to reduce the consumption as DVFS may induce timing violations if the PM is not aware of the RT constraints. Therefore, control the energy consumption, and the performance of RT applications constitute a challenging tradeoff for the PM systems [1].

The evaluation of PM proposals reveals several limitations. High-level abstract models of the systems are used due to the complexity of the PM techniques [2][3][4]. However, assumptions made at the abstract models may lead to inaccuracy when applying the PM model in real systems [5]. On the other hand, works validated in real systems [6][7] present results for small systems (less than ten cores). Another issue is design-time based heuristics. Many-core systems are designed to support a dynamic workload, i.e., new applications may start their execution at any moment, making unfeasible to evaluate all execution scenarios at design-time. Nevertheless, several proposals develop heuristics for PM according to a design-time analysis of the application set [3][4] [6][8].

To save energy in this complex many-core context, related works delay the execution of RT applications by exploiting the slack time inherent in RT applications [3][4]. Moreover, soft RT applications, as multimedia, admit small variations in the throughput rate [9].

The *goal* of this work is to propose an adaptive Runtime Energy Management designed to execute soft RT applications – RT-REM, while exploring the slack time of RT applications to save energy.

To the best of our knowledge, RT-REM is the first work to link *the execution of RT applications and power management without design-time analysis of the application set*. The RT-REM actuates as a high-level manager, setting the DVFS parameters and migrating BE (Best Effort) tasks when it is necessary, and at the PE level, it avoids task violations in the RT constraints.

II. Background

A. Many-core Platform

Fig. 1 illustrates the many-core platform modules. The many-core contains of a set of Processing Elements (PEs) interconnected by a 2D-NoC. The application repository is an external memory that stores the applications' object code. Two similar descriptions model the platform: (*i*) synthesizable VHDL, for characterization purposes; (*ii*) RTL SystemC, with clock-cycle accuracy, enabling the simulation of systems with dozens of PEs.

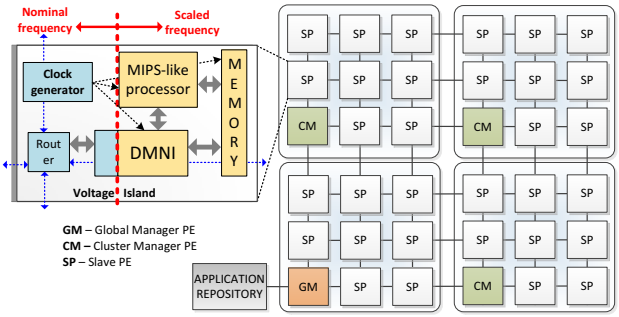

Fig. 1. A 6x6 instance of the reference many-core system, with four 3x3 clusters (adapted from [10]).

The PEs have different roles in the system: manager processors, global manager (GM) and cluster managers (CM); and slave processors (SP). The system is organized hierarchically, in regions named *clusters,* with one manager PE (GM or CM) and a set SPs. The cluster may increase its size, borrowing slave PEs from neighbor clusters, in a process named *reclustering*. The hardware of the PEs is the same, being the differentiation made by software. The CM works as a resource manager at the cluster level, assigning the applications' tasks to the SPs of the cluster. The GM accumulates the role of a CM and receives application execution requests via the external interface (application repository).

Definition 1: OS_{SP} – operating system executing at SPs.

Definition 2: OS_{MG} – operating system executing at CMs and GM.

B. DVFS Design

Fig. 1 shows that the frequency scaling actuates only on the processor, memory, and DMNI (a network interface with DMA capabilities). The main goal is to enable processors to work at different frequencies while the NoC transmits packets using the nominal frequency. The reason for transmitting packets at the

nominal frequency is to avoid PEs running at higher frequencies stall due to PEs running at lower frequencies. The DMNI synchronizes the hardware modules working at different frequencies through bisynchronous FIFOs added in the DMNI.

The set of voltage levels is a function of the available Liberty files. In the current work, standard cells are characterized for 1.1V, 1.0V, and 0.9V. The timing analysis of the netlist produced by the logic synthesis at the highest voltage (1.1V) defines the nominal frequency, which is 250 MHz (T=4 ns). Next, timing analysis is applied in the same netlist, for 1.0V and 0.9V. The minimum period to obtain a zero or positive time slack is 4.479 ns and 5.229 ns, for 1.0 V and 0.9 V, respectively. The router maintains a positive time lack for all supply voltages. Thus, the DVFS protocol uses *vf-pairs*: (1.1V, 4ns), (1.0V, 4.5ns), (0.9V, 5.5ns).

Due to the low latency of fine-grain voltage regulators [11] and the feature of frequency scaling at the PE level, we model a fine-grain (PE level) voltage scaling considering that the latency of a voltage scaling (up or down) is 100ns, and the energy overhead from on-chip voltage regulators increases the PE energy in 10% [12].

C. Real-time Modeling

RT applications contain a set of tasks, where the correct execution includes not only the expected results but also when these results are produced. A directed acyclic task graph *G(T, E)* models an *m*-task application $A=\{t_1, t_2, ..., t_m\}$, where each vertex $t_i \in T$ is a task and the directed edge (e_i, e_j), denoted as $e_{ij} \in E$, is the communication between tasks t_i and t_j. Fig. 2(a) shows an example of a task graph for an RT application.

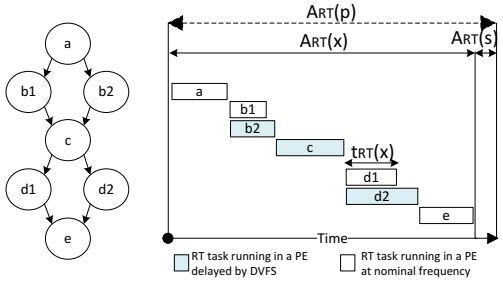

Fig. 2. Task graph of an RT application (a) and its scheduling (b).

Definition 3: $A_{RT}(p)$ - application hyper-period. The hyper-period is the smallest interval of time after which the periodic patterns of all the tasks are repeated.

Definition 4: $A_{RT}(x)$ - application execution time, corresponds to the time to execute all tasks during one $A_{RT}(p)$.

Definition 5: $A_{RT}(s)$ - application slack time, corresponds to the difference between $A_{RT}(p)$ and $A_{RT}(x)$ as follows:

$$A_{RT}(s) = A_{RT}(p) - A_{RT}(x) \quad (1)$$

The model of an RT task t_{RT} is a tuple $t_{RT} = (x, u)$ where x is the task execution time including operating system events and communication time, and u is the task utilization. Fig. 2(b) presents the $A_{RT}(p)$, $A_{RT}(x)$, $A_{RT}(s)$, and $t_{RT}(x)$ for task *d1*. The task utilization, $t_{RT}(u)$, is defined as follows:

$$t_{RT}(u) = \frac{t_{RT}(x) * 100}{A_{RT}(p)} \quad (2)$$

The $A_{RT}(s)$ is the *key* opportunity for energy saving because the system manager may set the *vf-pairs* of each RT task without violating the $A_{RT}(p)$. Applying DVFS imposes delay to tasks. In Fig. 2(b), for instance, some of the RT tasks have their execution time increased due to the adoption of the DVFS. Delayed tasks do not violate the $A_{RT}(p)$ *iff* $A_{RT}(s) \geq 0$, i.e., $A_{RT}(x) \leq A_{RT}(p)$.

III. RT Application Monitoring

The RT-REM does not receive at design time the workload information related to the applications to execute in the system. Consequently, this work adopts code annotation to transmit to the operating system the RT constraints. Fig. 3 presents a code snippet for task *e* of Fig. 2(a). The OS handles two system calls (*syscalls*):

- *RealTime*: sends the $A_{RT}(p)$ and the $t_{RT}(x)$ to the OS_{SP}, and to the OS_{MG} (line 2 of Fig. 3). The task execution time, $t_{RT}(x)$, corresponds to the number of clock cycles the scheduler execute one task iteration (nominal frequency). If the PE frequency reduces, the task takes longer to execute, as shown in Fig. 2.
- *PeriodMonitoring*: sends the current time (at the nominal frequency) to the OS_{MG}. The first task of the application includes this *syscall* at the beginning of an iteration. The last task includes this *syscall* at the end of the application iteration (line 7 of Fig. 3). This *syscall* allows to the OS_{MG} to measure of $A_{RT}(x)$, $A_{RT}(s)$ and $A_{RT}(p)$.

RT task e: code

```
1 int hyper_period=60000, exec_time=8000;
2 RealTime(hyper_period, exec_time);
3 for k iterations do
4   Receive(msg from d1);
5   Receive(msg from d2);
6   DoSomething();
7   PeriodMonitoring(get_nominal_tick_counter());
8 end for
```

Fig. 3. Code snippet for the last RT task of an application.

The OS_{SP} uses the *RealTime* data to control the execution time of the RT task. The OS_{SP} schedules the RT tasks using the *Least Slack Time* (LST) scheduler [13], and a round-robin scheduler for BE tasks.

The OS_{MG} uses the *RealTime* data to compute $t_{RT}(u)$ enabling the DVFS actuation at the PE level (Section IV.C). Note that, the *PeriodMonitoring* captures delays produced by application tasks, traffic in the NoC, and operating systems events (as interruptions). Therefore, the computed $A_{RT}(x)$ and $A_{RT}(s)$ are accurate, enabling the RT-REM heuristic to act in the application.

IV. Real-Time Runtime Energy Management

When an incoming application request execution in the system, the GM selects the cluster to receive the application, executing an *application admission* heuristic (Section IV.A). Next, the selected CM employs heuristics for task mapping at the cluster level (IV.B). A given SP may execute one RT task; a set of BE tasks; or one RT tasks and a set of BE tasks. Section IV.C details the process to apply DVFS to SPs executing an RT task.

A. Application Admission

The GM receives requests to execute new applications. The GM verifies if there are enough resources in the system to run all tasks of the application. In the absence of resources to execute the application, the GM delays the incoming application admission up to the release of enough resources.

Definition 6: *resource* - available page in the memory of a PE able to execute a task, assuming a paged memory system, where each processor may execute a set of tasks simultaneously.

When the GM admits an application, it executes the Algorithm 1. Lines 4-9 creates the set *set_cl*, with the clusters that may execute the application. If the *set_cl* is not empty (lines 10-11), the selected cluster is the one with the lowest consumed energy, based on the monitoring data. A hierarchical monitoring method, as the one presented in [10], estimates the energy of the application set

executing in the system. The goal is to distribute the load evenly in the system, avoiding hotspots. Otherwise (line 13), the selected cluster is the one with the maximum number of available resources. In such a case, the reclustering protocol is applied, and the select cluster borrows resources from neighbor clusters.

```
Algorithm 1: Application admission (executed at the GM)
1  Input: graph G(T,E)
2  Output: cl_k to execute the application
3  begin
4     set_cl ← ø
5     foreach cl_k of CL do     // CL is the set of all clusters of the system
6        if available_resources(cl_k) ≥ |T| then
7           set_cl ← set_cl ∪ cl_k
8        end if
9     end for
10    if set_cl ≠ ø then
11       cl_k ← min_energy(set_cl)
12    else
13       cl_k ← max_available_resources(cluster_set)
14    end if
15 end
```

B. Task Mapping

The OS_{MG} employ a set of cost functions to guide the task mapping and migration: (*i*) *available_SPs()* returns the set of SPs with available *resources* to execute tasks; (*ii*) *SP_with_RT_task()* returns the set of SPs running RT tasks; (*iii*) *min_energy()* returns the SP with the minimum *current energy* (obtained by the hierarchical monitoring process) nearest to the SPs executing tasks of the application.

The OS_{MG} maps a task t_i according to the Algorithm 2. The first action is the creation of a set with the SPs able to execute t_i (line 5) - set_SP. For an RT task, the mapping algorithm excludes from set_SP the SPs running RT tasks (line 7), and then chooses the SP with the minimum *current energy* (line 8). For BE tasks only the minimum *current energy* is considered (line 10). If an SP may receive t_i, line 13 executes the mapping, i.e., request to the GM to transfer the object code of t_i to the selected SP. The mapping of an RT task may not occur, even if the cluster has resources to execute the task. This scenario happens when SPs that are not executing RT tasks have all resources used by BE tasks. In this case, it is necessary to migrate a BE task. Since the GM maps application only if there is place in the system, the migration algorithm always finds an SP available to remap the BE task.

```
Algorithm 2: RT-REM Mapping (executed at the manager PEs)
1  Input: cluster_SPs[x_size][y_size], t_i
2  Output: map a task in an SP of the cluster
3  begin
4     selected_SP ← ø
5     set_SP ← available_SPs(cluster_SPs)
6     if t_i.type = RT_TASK then
7        set_SP ← set_SP – SP_with_RT_task(set_SP)
8        selected_SP ← min_energy(set_SP)
9     else
10       selected_SP ← min_energy(set_SP)
11    end if
12    if selected_SP ≠ ø then
13       map_task(t_i, selected_SP)
14    end if
15 end
```

C. DVFS for an SP running an RT task

The RT-REM adopts the following definitions to set the *vf-pairs* of each task of an RT application:

Definition 7: *HIGH_UTILIZATION* – $t_{RT}(u)$ above a predefined threshold related to the $A_{RT}(p)$. Set to 70% in the current work

Definition 8: *LOW_UTILIZATION* – $t_{RT}(u)$ below a predefined threshold related to the $A_{RT}(p)$. Set to 30% in the current work.

The current work adopts fixed thresholds (Definitions 7 and 8) because the adaptation at runtime of these thresholds without evaluating the applications set at design-time would require a large warming up period and learning-based algorithms [14].

As detailed in Section II.C, the $A_{RT}(p)$ is the constraint to meet in RT applications. As the voltage scaling introduces delays on the $t_{RT}(x)$, RT-REM heuristic selects *vf-pairs* based on $t_{RT}(u)$ assuming a low utilization task is less likely to generate $A_{RT}(p)$ violations than a high utilization one.

Algorithm 3 runs at manager processors. This algorithm defines the *vf-pair* of an SP executing an RT task (t_{RT}) when the manager processor receives a packet created by the execution of a *syscall RealTime* in an SP.

```
Algorithm 3: DVFS for t_RT (executed at the manager PEs)
1  Input: t_RT, x_pos, y_pos
2  Output: send to SP(x_pos,y_pos) new settings of VF
3  Begin
4     t_RT(u) ← (t_RT(x)/ * 100) / A_RT(p)    // equation 2
5     if t_RT(u) > HIGH_UTILIZATION then
6        sendDVFS_CHANGE(x_pos,y_pos,VF_PAIR(1.1V, 4ns))
7     elsif t_RT(u) < LOW_UTILIZATION then
8        sendDVFS_CHANGE(x_pos,y_pos,VF_PAIR(0.9V, 5.5ns))
9     else
10       sendDVFS_CHANGE(x_pos,y_pos,VF_PAIR(1.0V, 4.5ns))
11    end if
12 end
```

PEs running at 1.0V and 0.9V delays a task execution by 12.5% and 37.5%, respectively, due to the frequency reduction. Thus, an SP running a t_{RT} with high utilization operates at the nominal voltage (lines 5-6) while an SP running a t_{RT} with low utilization operates at *vf-pair* (0.9V, 5.5ns), delaying the task by 37.5% (lines 7-8). Between the thresholds, the SP executes at *vf-pair* (1.0V, 4.5ns) (line 10).

If the OS_{MG} receives a predefined number of *hyper-period slack time* violations (from the *syscall PeriodMonitoring*), the RT-REM reset the *vf-pair* of all application tasks to the nominal voltage.

V. RESULTS

The experiments are executed using the clock cycle accurate RTL SystemC model of the reference many-core system. Applications and OS are described in C language, compiled from C code and executed over cycle-accurate models of the processing cores. The RT benchmarks are *DTW* (6 tasks), *Dijkstra* (7 tasks), and *MPEG* (5 tasks), while the BE applications are *synthetic* (6 tasks) and *prod_cons* (2 tasks).

The baseline many-core has no DVFS support but keeps the RT scheduler for processing RT tasks, the mapping and the energy monitoring. The addition of the DVFS support in the platform resulted in an execution time overhead of 6.55%. Therefore, the baseline platform has neither 10% of energy overhead nor spends 6.55% more execution time.

A. Evaluation of RT-REM for an RT application

Fig. 4(a) presents the hyper-period of an RT application, the baseline execution time (black curve) and the execution time using RT-REM (red curve). The distance from the execution time curves to the hyper-period corresponds to the slack-time.

The variability in the execution time observed in both scenarios (baseline and REM) are induced by branches of the application, OS events, interruptions, and network congestion. As the RT tasks of

the benchmarks do not use WCET, a low frequency of timing violations is acceptable [15].

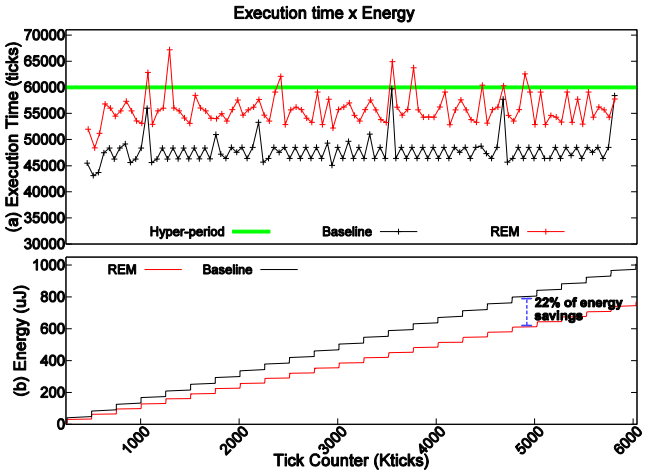

Fig. 4. Execution time and energy of an RT application with and without REM for 100 iterations.

The hyper-period defined by *syscall RealTime* for each iteration is 60 Kticks (green horizontal line). The execution of the baseline system does not present hyper-period violations, but an important slack-time is observed. Using REM, after the definition of the *vf-pairs*, the slack-time reduces, with few violations in the hyper-period (smaller than 10% of the hyper-period value). Fig. 4(b) plots the energy consumed by the system for the baseline and the system with RT-REM. As the simulation advances the energy savings increases, reaching 22% after executing 100 iterations of the applications.

B. *Evaluation of the RT-REM proposal*

Table 1 presents the evaluation of hyper-period violations and energy savings, for systems having up to 144 PEs. Five many-core systems (first column), divided into different cluster sizes (second column), run the applications set. The number of RT and BE tasks to process is the same of the number of SPs of the many-core (third column). For instance, the 6x6-many-core has one GM, three CMs, and 32 SPs such as the example of Fig. 1. The fourth column presents the number of executed task migrations.

Table 1 - Violations of hyper-periods and energy savings of RT-REM compared to the baseline system.

Many-core size	Cluster size	Number of tasks for each type	Number of migrations	Violations of hyper-period	Energy savings
6x6	3x3	32	3	1.47%	15%
8x8	4x4	60	6	2.09%	18%
9x9	3x3	72	8	1.84%	20%
12x12	3x3	128	8	1.87%	18%
12x12	4x4	135	14	1.97%	18%

The fifth column of Table 1 shows the percentage of hyper-period violations (the execution of the applications in the baseline system does no present violations, as in Fig. 4(a)). The RT-REM produces a small number of hyper-period violations (< 2.1 %), with an amplitude inferior to 10% of the defined constraint. This result shows that the proposed RT-REM is suitable for soft-RT applications. For example, few hyper-period violations decoding a video frame do not affect the performance of the applications [9].

The last column of Table 1 presents the energy savings, for a simulation time of 50 ms. The average energy reduction observed is in average 18%. As shown in Fig. 4(b) the energy saving follows the execution time. Therefore, for applications execution for long periods, important energy savings are expected.

VI. Conclusions and Future Work

The RT-REM proposed in this work reduces the consumed energy in many-core systems while executing soft RT applications meeting the applications' constraints. The proposed closed-loop control responsible for managing the system executes: monitoring (hyper-period slack time), decision (RT-REM heuristic), and actuation (DVFS and task mapping/migration).

As the results section shown, the proposal is the *scalable*, with similar energy savings for different system sizes (from 36 up to 144 PEs), producing a small number of hyper-period violations (<2%).

Summarizing, the main *original contribution* of this work is the integration of a comprehensive set of techniques for a runtime energy management at the PE level and the execution of RT applications respecting the RT constraints.

Future works include: (*i*) define consumption limits to cope with dark silicon issues; (*ii*) include other actuation techniques, as power gating; (*iii*) evaluate the approach for SOI technologies; (*iv*) include additional cost functions to the RT-REM heuristic to enable more than one RT task running at the same SP.

Acknowledgement

The Author Fernando Moraes is supported by CNPq and FAPERGS funding agencies.

References

[1] H. J. Siegel, et al. "Energy-Aware Resource Management for Computing Systems". In: *IC3*, 2014, pp. 7–12.

[2] S. Maiti, N. Kapadia, and S. Pasricha, "Process variation aware dynamic power management in multicore systems with extended range voltage/frequency scaling". In: *MWSCAS*, 2015, pp. 2–5.

[3] X. Li, Z. Jia, and L. Ju, "Slack-time-aware energy efficient scheduling for multiprocessor SoCs". In *HPCC-EUC 2013*, 2014, pp. 278–285.

[4] A. K. Singh, A. Das, and A. Kumar, "Energy optimization by exploiting execution slacks in streaming applications on multiprocessor systems". In: *DAC*, 2013, p. 1-7.

[5] S. L. Xi, et al. "Quantifying sources of error in McPAT and potential impacts on architectural studies". In: *HPCA*, 2015, pp. 577–589.

[6] A. Das, et al. "Adaptive and Hierarchical Runtime Manager for Energy-Aware Thermal Management of Embedded Systems". *ACM Trans. Embed. Comput. Syst.*, vol. 15, no. 2, pp. 24:1–24:25, 2016.

[7] H. Yu, R. Syed, and Y. Ha, "Thermal-aware frequency scaling for adaptive workloads on heterogeneous MPSoCs". In: *DATE*, 2014, 6p.

[8] H. Jung, et al. "Dynamic Behavior Specification and Dynamic Mapping for Real-Time Embedded Systems". *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 4s, pp. 1–26, Apr. 2014.

[9] R. Feghali, et al. "Video quality metric for bit rate control via joint adjustment of quantization and frame rate". *IEEE Trans. Broadcast.*, vol. 53, no. 1, pp. 441–446, 2007.

[10] A. Martins, M. Ruaro, F. Moraes. "Hierarchical energy monitoring for many-core systems". In: *ICECS*, 2015, pp. 657–660.

[11] W. Kim, et al. "System Level Analysis of Fast, Per-Core {DVFS} using On-Chip Swithching Regulators". In: *HPCA*, 2008, pp. 123–134.

[12] Y. Choi, N. Chang, and T. Kim. "DC-DC converter-aware power management for low-power embedded systems". *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 26, no. 8, pp. 1367–1381, 2007.

[13] H. Myunggwon; C. Dongjin; K. Pankoo. "Least Slack Time Rate First: New Scheduling Algorithm for Multi-Processor Environment". In: CISIS, 2010, pp.806-811.

[14] S. Yang, et al. "Adaptive energy minimization of embedded heterogeneous systems using regression-based learning". In: *PATMOS*, 2015, pp. 103-110.

[15] S. Manolache, P. Eles, Z. Peng, "Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints"" ACM TECS, vol. 7 no. 2, pp. 1-35, 2008.