Issue of Heterogeneous Distributed Real-Time Embedded Systems for Voltage Scheduling

Navin Soni[1], Dr. Rahul Mishra[2]

[1]M. Tech Scholar in Dr. A.P.J Abdul Kalam University, Indore, Madhya Pradesh, India
[2]Associate Professor in Dr. A.P.J Abdul Kalam University, Indore, Madhya Pradesh, India

Abstract

This paper addresses the problem of static and dynamic variable voltage scheduling of multi-rate periodic task graphs (i.e., tasks with precedence relationships) and aperiodic tasks in heterogeneous distributed real-time embedded systems. Such an embedded system may contain general-purpose processors; field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs). Variable voltage scheduling is performed only on general-purpose processors. The static scheduling algorithm constructs a variable voltage schedule via heuristics based on critical path analysis and task execution order refinement. The algorithm redistributes the slack in the initial schedule and refines task execution order in an efficient manner.

The variable voltage schedule guarantees all the hard deadlines and precedence relationships of periodic tasks. The dynamic scheduling algorithm is also based on an initially valid static schedule. The objective of the on-line scheduling algorithm is to provide best-effort service to soft aperiodic tasks, as well as to reduce the system power consumption by determining clock frequencies (and correspondingly supply voltages) for different tasks at run-time, while still guaranteeing the deadlines and precedence relationships of hard real-time periodic tasks.

Keywords: *Voltage-Scalable, Heterogeneous, Distributed Systems, Real time Embedded System.*

1. INTRODUCTION

Integrating heterogeneous devices allows to raise the processing capacity without, necessarily, having a centralized control on a single device. To improve performance and increase cost-effectiveness, the processing tasks can be, normally, distributed. However, the integration of diverse devices demands a reliable communication, which is not an easy task, needing a mechanism that manages and synchronizes the members' messages. Building an environment to manage the exchange of data is even more difficult, because problems may arise from the integration of different devices. The integration of computing systems (software and hardware) allows to create a System of Systems (SOS).

The evolution and growth of the heterogeneous systems used for high-performance computing have been surprising for many years. Generally, heterogeneous systems [1] for parallel and distributed computing, consist of the system software, multiple processors, memories, and a communication network. Nowadays, more than 60 percent of electronic control unit parts in both automotive electronics and avionics systems employ the heterogeneous distributed embedded system (HDES) [2] because of the benefits in the resource utilization, weight, scale, and power consumption and high-performance. In the HDES architecture, the specialized processing elements are used to accelerate the complex tasks, also coprocessors for general purpose computing cooperated with single or multicore processors are exploited for tasks require flexibilities. There are many types of coprocessors such as application-specific integrated circuit chips, FPGAs (field-programmable gate arrays), GPUs (graphics processing units) architecture. Similar to GPGPUs (general

purpose GPUs) produced by Intel Corporation, MIC (many integrated core) is another kind of coprocessor, which is targeted for highly parallel workloads in a large set of scientific data. Data-dominated applications in a variety of fields, such as computational, engineering, and scientific, are described by a DAG (directed acyclic graph). Nodes represent tasks and edges represent communication messages between tasks in the DAG. The scheduling tasks on HDES processors [3] to minimize the schedule length of a DAG-based parallel scientific application is a well-known nondeterministic polynomial-hard (NP-hard) optimization problem, and numerous meta-heuristic list scheduling algorithms have been proposed to generate near-optimal solutions of the problem.

2. HETEROGENEOUS DISTRIBUTED REAL-TIME EMBEDDED SYSTEMS

An interesting comparison of scheduling approaches from a more industrial— in particular, automotive—perspective can be found in Lonn and Axelsson [1999]. ¨ Their conclusion is that we have to choose the right scheduling approach depending on the particularities of the scheduled processes. This means not only that there is no single "best" approach to be used, but also that inside a certain application several scheduling approaches can be used together. Efficient implementation of new, highly sophisticated automotive applications entails the use of time-triggered process sets together with event-triggered ones implemented on top of complex distributed architectures.

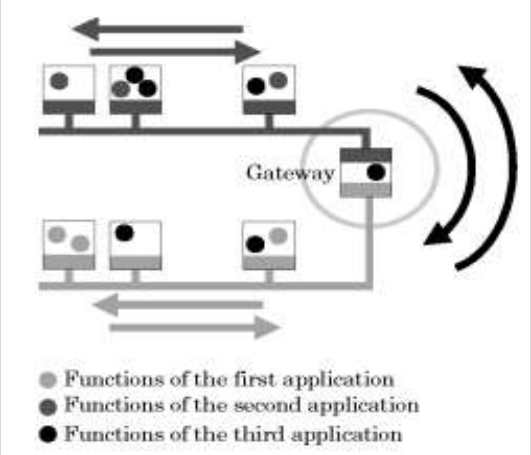

Fig 1: Distributed safety-critical applications

HDES hardware setup

The HDES hardware environment as consists of nine the Parallella boards, known as HEM, and a 16-port of D-Link DGS-1016D

Fig 2: The HDES hardware setup

gigabit switch connects with CAT5e LAN cable. The users can monitor on the laptop through wireless LAN. The operating system is the Parabuntu 2016.11.1, embedded Linux, official Ubuntu distro for Parallella installed that is open source from the community of www.parallella.org. The APIs are operated using a collection of COPRTHR SDK from Brown Deer Technology with the Epiphany SDK from Adapteva.

3. STATIC VARIABLE-VOLTAGE SCHEDULING FOR MULTIRATE PERIODIC TASK GRAPHS

This section presents a variable-voltage static scheduling algorithm for multirate periodic task graphs under a given PE/link allocation and task/communication assignment. For a multirate system, the schedule is generated for all the instances of tasks and communication edges in the hyperperiod. It is worth mentioning that in our approach, we decouple the problems of allocation/assignment and scheduling. Allocation/assignment can be done using any algorithm, e.g., constructive, iterative improvement, genetic, etc. Our scheduling algorithm is based on critical-path-based list scheduling, which has been proven to be able to provide near-optimal solutions for many applications (when no voltage scaling is done). The list-scheduling algorithm maintains a pending list of ready tasks whose parent tasks have been scheduled. The pending list is ordered by task priorities. Each time, the task with the highest priority in the pending list is selected for scheduling. First, all its incoming edges are scheduled, and then, it is scheduled in the earliest possible slot without violating precedence relationships. The pending list is then updated with new ready tasks. List scheduling has been widely accepted for scheduling of distributed systems. To adapt a list-scheduling algorithm to variable voltage scheduling, two aspects need to be addressed. First, a priority assignment, which determines the execution order of events, should be able to maximize the slack available in the schedule. Second, slack needs to be allocated efficiently to maximize energy savings. In this section, the first aspect is addressed through initial priority assignment and execution-order optimization, as discussed in Section IV-A and B, respectively. The second aspect is addressed through power-profile and timing-constraint driven slack allocation, as discussed in Section IV-C. There is a rich literature for deriving different priority assignments for list scheduling in order to optimize schedule length or schedulability. Priority assignment can be based on earliest start time, latest start time, and relative mobility (which is defined as the difference between the latest start time and earliest start time), as summarized. More sophisticated priority-assignment schemes have been proposed too. Dynamic level scheduling utilizes a priority function-based dynamic level of a node, which is defined as the difference between the critical-path length of the node and its earliest start time. Here, the critical-path length of a node is defined as the longest path from the node to a sink node. The work in uses a priority function based on a node's earliest start time plus its critical-path length. The priority function for list scheduling is based on the partial length of a node's critical path, which only takes into account the execution times of nodes not assigned to the same PE as this node. The tradeoffs involved in different list-scheduling algorithms are discussed. Normally, priorities are assigned statically based on the characteristics of task graphs before the scheduling process starts. A variant of list scheduling is to determine priorities dynamically during the scheduling process. The priorities of events are recomputed after an event is scheduled, in order to capture the changes in the relative criticality of the event. However, even with dynamic adjustment, it is still impossible to capture all resource contentions during the scheduling process since bus contention and resource sharing on each PE and communication link can only be fully determined after the schedule is generated.

Therefore, after a valid schedule is initially generated, there should be room for execution-order adjustment of scheduled events, which may lead to better schedule flexibility and, correspondingly, more power savings. The execution order of scheduling events can be optimized through greedy iterative improvement. However, the solution may be trapped into local minima. In order to achieve a globally optimal solution, we chose to use a simulated-annealing approach to further optimize the execution order of events based on the initial priority assignment.

This section is organized as follows. In Section IV-A, we discuss briefly the initial priority assignment. In Section IV-B, we discuss in detail execution-order optimization based on simulated annealing. In Section IV-C, we present the power-profile and timing-constraint driven slack-allocation algorithm for variable-voltage scaling. In Section IV-D, we present an example to illustrate the different steps.

A. Initial Priority Assignment

Initially, we utilize a priority assignment using the inverse of the aslate-as-possible start time (ALAP_start). ALAP_start refers to the latest time at which a scheduling event can start, without violating deadlines of any task and the precedence relationships specified in the task graphs when no resource constraint is considered. Let di denote the deadline specified in the task graphs for task i, outedges(i) denote the set of out-going edges of task i in the task graphs, successor(e) denote the successor node of an edge e, and wcetj denote the wcet of the task or communication edge j. The wcet of a task is a function of its working clock frequency, as discussed in Section II. We assume that intra-PE communication takes zero time. We define ALAP_start of a task and edge as follows.

For a task i

$$ALAP\_start_i = \min\left(d_i, \min_{j \in out_{edges}(i)}(ALAP\_start_j - wcet_j)\right) - wcet_i.$$

For a communication edge e

$$ALAP\_start_e = ALAP\_start_{successor(e)} - wcet_e.$$

Let si denote the arrival time specified in the task graphs for task i, inedges(i) denote the set of incoming edges of task i in the task graphs, and predecessor(e) denote the predecessor node of an edge e. Similarly, we define the as-soon-as-possible start time (ASAP_start) of a task i as.

$$ASAP\_start_i = \max\left(s_i, \max_{j \in in_{edges}(i)}(ASAP\_start_j + wcet_j)\right)$$

We define ASAP_start of a communication edge e as

$$ASAP\_start_e = ASAP\_start_{predecessor(e)} + wcet_{predecessor(e)}.$$

In (14) and (16), the wcet of a task is based on the maximum supply voltage and frequency. The ALAP_finish of a scheduling event i is.

$$ALAP\_finish_i = ALAP\_start_i + wcet_i.$$

In our experimental results, for comparison, we will also present the results of using the inverse of ASAP_start as the priority assignment. According to, ALAP_start-based list scheduling provides comparable or better results than most other scheduling approaches in terms of schedule length or schedulability. More sophisticated scheduling algorithms, such as dynamic critical-path scheduling, have the potential to reduce the schedule length, introduce more slack into the system, and possibly achieve better power savings. However, although in this paper we utilize an ALAP_start-based list scheduling, the slack-allocation techniques discussed later are general and also applicable to other scheduling algorithms.

B. Execution-Order Optimization of Scheduled Events

Starting from an initial schedule, we perturb the priority assignment of each event. For every event, we generate a random number δ and use the inverse of ALAP_start + δ as the new priority assignment. Then, based on the new priority assignment, we regenerate the execution order of the scheduled events on every PE and link using the list scheduling algorithm mentioned in the beginning of Section IV. We then generate the new variable-voltage schedule through a slack allocation algorithm, which is discussed in Section IV-C, and reevaluate the new power consumption after voltage scaling. A better solution (i.e., with lower power consumption) will always be accepted. An inferior solution will be accepted with a probability 1/(1 + exp((new power − power)/T)), where T is the annealing temperature and new power and power are the power

consumptions after and before the perturbations, respectively, of ALAP_start. The overall framework for execution-order optimization. Note that in the step in which the new power consumption is evaluated, the slack-allocation algorithm discussed in Section IV-C is called to generate the variable-voltage schedule. Under a given temperature, the perturbations stop if the number of consecutive perturbations that was not accepted, or the total number of perturbations, has exceeded some predefined threshold.

C. Power-Profile and Timing-Constraint Driven Slack Allocation and Variable-Voltage Scaling

This section discusses power-profile and timing-constraint driven slack allocation for a given execution order to maximize power reduction using an iterative approach based on Theorem 1. The slackallocation algorithm is presented as Algorithm 1. It addresses how to evaluate the validity of the schedule when multiple tasks need to update their execution times (due to voltage scaling) as well as to annul any invalid updates in linear time (O(n + k)), where n is the number of tasks and k is the number of inter-PE communication edges.

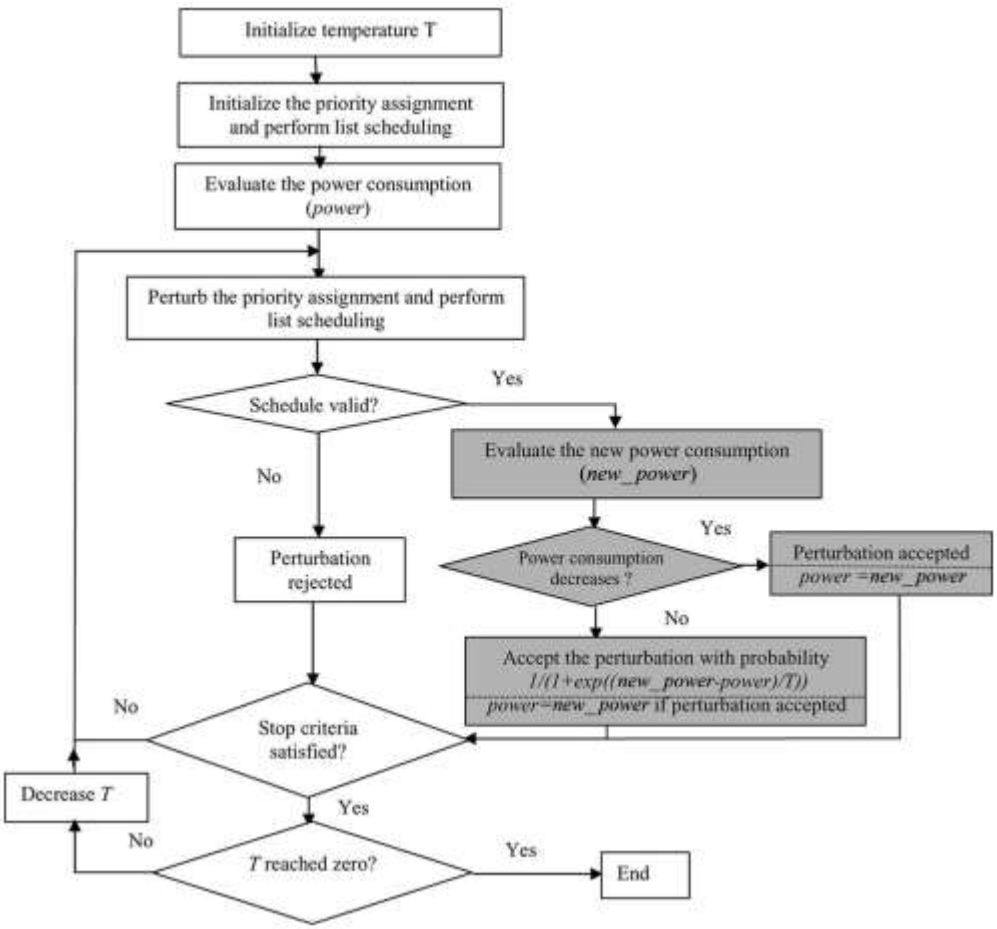

Fig 3: Framework for execution-order optimization of scheduled events

power_prof ile_slack_allocation(G(V,E))

1) Initialize $v_l[i] = V_{\max}^{\mathrm{PE}_i}, \forall tasks\ i$

2) initialize task list T_s containing all the tasks on voltage-scalable PEs

3) sort T_s in the order of decreasing energy gradients

4) initialize active task list A_s with the set of tasks on voltage-scalable PEs with the highest energy gradient

5) while A_s is not empty or T_s is not empty do

6) for each event k in the order of *top_sort*($G(V, E)$) do

7) compute *ASAP* _start$_k$ of event k based on $v_l[i]$, i

8) end for

9) $v^{\text{old}} = v_l$

10) j = the element in A_s with the highest voltage level

11) $v_l[j] = dv$

12) *reference_gradient* = $G(v_l[j], N_j)$

13) for each task i in A_s do

14) $v_l[i] = \min_V G(V, N_I)$ *reference_gradient*

15) end for

16) for each event k in the order of *reverse_top_sort*($G(V, E)$) do

17) compute *ALAP* _finish$_k$ of event k based on $v_l[i]$, i

18) if (*ALAP* _finish$_k$ wcet$_k(v_l[k])$ < *ASAP* _start$_k$)

19) then

20) k marked as nonextensible

21) $v_l[k] = v^{\text{old}}[k]$

22) end if

23) end for

24) delete any task i from A_s, if it is marked as nonextensible, or if $v_l[i]$ has reached $V^{\text{PE}i}$

25) if A_s is empty then

26) append A_s with tasks with the highest energy gradient from T_s

27) else

28) append A_s with those tasks from T_s whose energy gradients become higher than the *reference_gradient*

29) end if

30) end while

31) return v_l

Algorithm 1 is based on an augmented directed graph G(V,E). After a valid schedule is generated through list scheduling and the order of scheduled events is determined, G(V,E) can be created based on the task graphs as well as the constraints imposed by resource sharing and bus contention after scheduling. In G(V,E), V is the set of vertices containing all the scheduled events in the initial schedule and E is the set of directed edges between vertices. An edge is inserted from one event to another if one is a direct predecessor of another in the task graphs or if one is scheduled just ahead of another on the same PE or link. Therefore, these edges represent all the precedence relationships in the original task graphs as well as execution ordering information in the initial schedule. The creation of G(V,E) can be illustrated through the following example. Example 1: Consider the embedded-system specification given in the form of two task graphs in Fig. 1. For the feasible schedule shown in Fig. 6, which is generated based on ALAP_start-based task priority assignment, the derived directed G(V,E) is shown in Fig.4. In Fig. 4, the dependencies introduced by execution ordering are represented by dotted arrows. One path with zero slack in the augmented graph, (T1, T2, E1, T3, T6, and T7), is represented by shaded nodes.

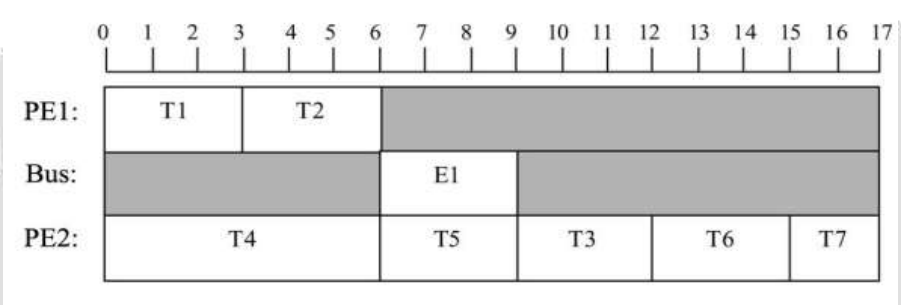

Fig 4: Initial schedule using ALAP_start-based priority assignment

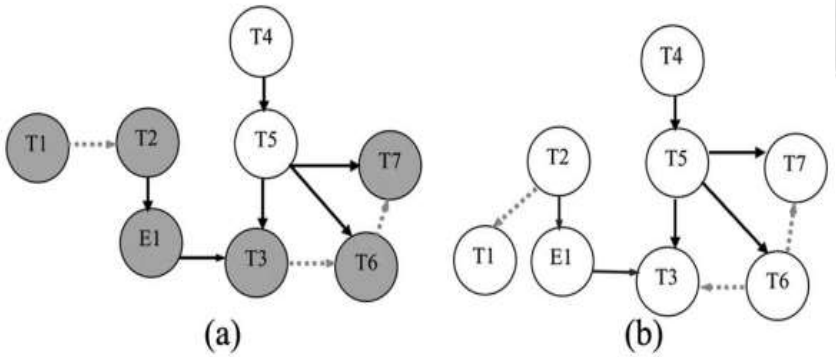

Fig 5: Augmented directed graph G(V,E). (a) G(V,E) for initial ALAP_start based priority assignment. (b) G(V,E)opt after execution-order optimization.

The distributed system consists of two PEs, PE1 and PE2, connected by a bus. The schedule is based on worst case task execution and communication times, assuming a supply voltage of 1.8 V. We assume that all PEs have communication buffers. For any scheduled event, its earliest possible start time (ASAP_start) and latest possible finish time (ALAP_finish) are calculated based on the augmented G(V,E). Note that in G(V,E), the node can be either a task or a communication event and the edge's execution time is zero. Based on G(V,E), precedence relationships introduced by task execution ordering are also taken into consideration.

In Algorithm 1, vl is an array of voltage levels for all the tasks. All the task voltage levels are initialized to V PEi max , where PEi is the PE to which task i is assigned and V PEi max is the maximum supply voltage on PEi. The task list is initialized with all the tasks on voltage-scalable PEs in the order of decreasing energy gradients. The active task list is initialized by appending tasks with the highest energy gradient from the task list. Whenever a task is added to the active task list, it is deleted from the task list. In each iteration step, the voltage level of the task in the active task list with the highest voltage level is decreased by dv and its new energy gradient is chosen as the reference level. The update of voltage levels of other tasks in the active task list is done such that their energy gradients do not drop below the reference level, since the rate of decrease of their energy gradients is higher, as discussed in Section III-A. In Algorithm 1, ASAP_startk (ALAP_finishk) is computed in the order of topological sort (reverse topological sort) of the augmented graph G(V,E) with the wcets based on vl[i], ∀i. During each iteration, if for a scheduled event, ALAP_finishk − wcetk(vl[k]) < ASAP_startk is true, the update of its voltage level gets invalidated and is switched back to the old value in vold l . Such a task is marked as nonextensible. All the tasks, which are marked nonextensible or have reached the minimum voltage level, are deleted from the active task list at the end of the iteration. The active task list is then updated in the following way. Whenever the active task list is empty, it is appended with tasks with the highest energy gradient from the task list. Otherwise, it is appended with those tasks from the task list whose energy gradients become higher than the reference level. The iteration stops when both the task list and active task list become empty. The overall complexity of Algorithm 1 is O((n + k) log(n + k) + M(n + k)), where M is the number of iteration steps. M is dependent

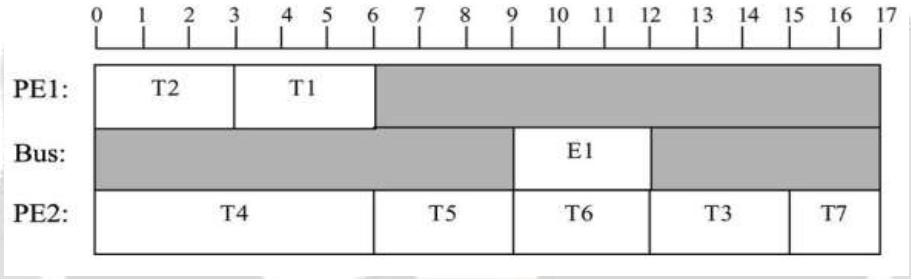

Fig 6: New schedule after execution-order optimization.

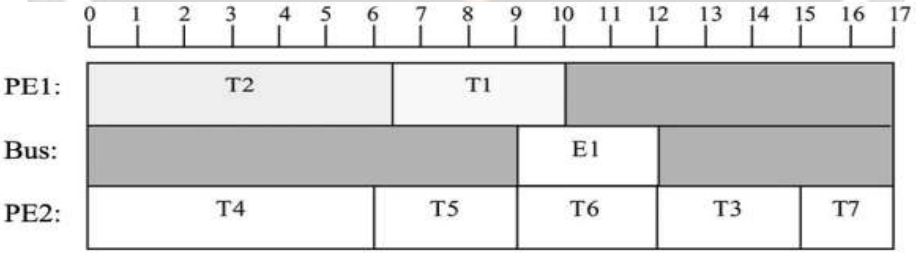

Fig 7: Variable-voltage schedule for the new schedule

on the distribution of switching activities of tasks in the system. Suppose L = (Vmax − Vmin)/dv is the number of voltage levels. In the case of Vmax = 1.8 V, Vmin = 0.75 V and Vt = 0.6 V, M should be of the same order as L when the variations of switching activities among tasks are in a normal range (i.e., the ratio of maximum to minimum switching activities does not exceed 100). When the power profile of the tasks is uniform, we have M = L. Therefore, our algorithm is also applicable to the case when the variations in power consumption of different tasks can be ignored. It is worth mentioning that Algorithm 1 can handle a fully heterogeneous system, in which different voltage-scalable PEs may have different Vmax and Vmin. The complexity of the overall scheduling algorithm is the complexity of Algorithm 1 plus the complexity of ALAP_start-based list scheduling, which is O(n(n + k) log n).

4. CONCLUSION

Heterogeneous distributed real-time systems are used in several application areas to implement increasingly complex applications that have tight timing constraints. The heterogeneity is manifested not only at the hardware and communication protocol levels, but also at the level of the scheduling policies used. In order to reduce costs and use the available resources more efficiently, applications are distributed across several networks. We have introduced the current state-of-the-art analysis and optimization techniques available for such systems, and have addressed in more detail a special class of heterogeneous distributed real-time embedded systems consisting of several interconnected clusters where time-triggered and event-triggered domains can share the same resource. We proposed an efficient variable-voltage scheduling algorithm, which can address variations in power consumptions, characteristics of different voltage-scalable PEs, precedence relationships, and hard deadlines of different tasks. It shows an order-of-magnitude run-time improvement over the previous works and can achieve close to optimal power savings. The scheduling algorithm is based on execution-order optimization of scheduling events, as well as slack budgeting motivated by the fact that the per-cycle energy consumption is normally a convex function of the processor clock period. Execution-order optimization can introduce more flexibility and increase the slack available to the system. The slack-budgeting algorithm can effectively distribute the slack among scheduling events to facilitate power saving through voltage scaling.

5. REFERENCE

1. PAUL POP et.al "Analysis and optimization of distributed real-time embedded systems" ACM Transactions on Design Automation of Electronic Systems, Vol. 11, No. 3, July 2006,
2. Sethakarn Prongnuch et.al. "A Heuristic Approach for Scheduling in Heterogeneous Distributed Embedded Systems" International Journal of Intelligent Engineering and Systems, Vol.13, No.1, 2020
3. Jiong Luo and Niraj K. Jha "Power-Efficient Scheduling for Heterogeneous Distributed Real-Time Embedded Systems" Authorized licensed use limited to: HOSEO UNIVERSITY. Downloaded on October 6, 2009 at 06:23 from IEEE Xplore. Restrictions apply.
4. Guoqi Xieet.al "High performance real-time scheduling of multiple mixed-criticality functions in heterogeneous distributed embedded systems" 1383-7621/© 2016 Elsevier B.V. All rights reserved.
5. G. Xie, G. Zeng, X. Xiao, R. Li and K. Li, "Energy-Efficient Scheduling Algorithms for Real-Time Parallel Applications on Heterogeneous Distributed Embedded Systems", IEEE Trans. on Parallel and Distributed Systems, Vol.28, No.12, pp.3426- 3442, 2017.
6. Jiong Luo et.al.(2002) "Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems" 10.1109/ASPDAC.2002.995019
7. Manzak and C. Chakrabarti, "Variable voltage task scheduling algorithms for minimizing energy", Proc. Int. Symp. Low Power Electronics and Design, Aug. 2001.
8. T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors", Proc. Int. Symp. Low Power Electronics and Design, pp. 197-202, Aug. 1998.
9. Le yan et.al. "Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Heterogeneous Distributed Real-time Embedded Systems" ICCAD '03: Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design November 2003.
10. Luo and N. K. Jha, "Power-profile driven variable voltage scaling for heterogeneous distributed real-time embedded systems," in *Proc. Int. Conf. VLSI Design*, Jan. 2003
11. Jaishree Mayank∗ and Arijit Mondal "An Integer Linear Programming Framework for Energy Optimization of Non-Preemptive Real Time Tasks on Multiprocessors" Journal of Low Power Electronics Vol. 15, 162–167, 2019
12. Yu Han et.al. "Energy-Efficient Scheduling Algorithms with Reliability Goal on Heterogeneous Embedded Systems" 978-0-7381-2646-3/21/$31.00 ©2021 IEEE DOI 10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00082
13. Medhat H A Awadalla. "Hybrid Scheduling Scheme for Real Time Systems" I S S N 2 2 7 7 - 3061 V o l u m e 1 5 N u m b e r 6
14. Ziran Peng et.al. "An optimal energy-saving real-time task-scheduling algorithm for mobile terminals" https://doi.org/10.1177/1550147717707891
15. Zahaf, HE, Benyamina, AEH, Olejnik, R. Energy-efficient scheduling for moldable real-time tasks on heterogeneous computing platforms. J Syst Architect 2017; 74: 46–60.