Dynamic Scheduling of Real-Time Tasks in Heterogeneous Multicore Systems

Kalyan Baital, *Member, IEEE*, and Amlan Chakrabarti, *Senior Member, IEEE*

***Abstract*—The shift from homogeneous multicore to heterogeneous multicore introduces challenges in scheduling the tasks to the appropriate cores maintaining the time deadline. This letter studies the existing scheduling schemes in a heterogeneous multicore system and finds an approach to enhance the homogeneous system model to heterogeneous scheduling architecture. The proposed model increases the overall system utilization by accommodating almost all the tasks (low power task and high power task) into appropriate cores (big high power and small low power). It further enhances the system performance by allocating rejected jobs from small cores into the big cores through a dispatcher.**

***Index Terms*—Big core, heterogeneous system, multicore, performance, real-time task, scheduling, small core.**

I. Introduction

HETEROGENEOUS multicore systems are increasingly entered into the semiconductor technology as an alternative design to traditional homogeneous multicores to improve both system throughput and energy efficiency [1]. Research has demonstrated that two types of cores namely big high-performance core and small low power core are able to implement most of the power and performance benefits from heterogeneity of cores [2]. In such systems, the real-time scheduling becomes more challenging because processors have different speeds [3]. Enhancement of energy and performance can be accomplished by allocating each task to the particular core type (big or small) that is best suited. Hence scheduling the task to the most appropriate core in heterogeneous multicore architectures opens up new challenges and possibilities for power management, task scheduling, and load balancing. The main contributions of our proposed model are as follows.

1) An efficient scheduling approach is proposed in the heterogeneous multicore system to map the tasks into appropriate cores based on the type of cores (big high power and small low power) as well as type of tasks (low power and high power) by extending the homogeneous system model of the existing work [4] and using the existing work of [5] and [6].
2) A run-time dispatcher has been incorporated in the work [4] to dispatch different jobs into small cores.
3) Task migration is used to migrate the rejected jobs from small cores into big cores through dispatcher.

This letter is organized as follows. Section II provides background and related works for heterogeneous multicore scheduling. Section III introduces our proposed task scheduling model for heterogeneous multicore architecture. Sections IV–VI verify the proposed model, provides the experimental results, and summarizes our contribution, respectively.

II. Related Works

Recent research proposes several scheduling schemes for heterogeneous multicore architectures. Kinsy and Devadas [7] presented an integer linear programming (ILP) formulation for scheduling tasks into heterogeneous multicore system. Tan *et al.* [8] proposed an approximation-aware scheduling framework for soft real-time tasks on the heterogeneous multicore architectures. Yan *et al.* [9] found a new Hadoop scheduler, DyScale, to utilize the potential of heterogeneous cores for achieving variety of performance objectives. More research work on scheduling tasks in heterogeneous multicore systems considering different parameters, namely task deadline and response time, task splitting, parallel multithread, processing power and functionality, task migration, temperature constraints can be found in [10]–[12].

III. Proposed Task Scheduling Model for Heterogeneous Multicore System

A. Problem Statement

Our objective is to introduce a real-time scheduling algorithm in heterogeneous multicore system where tasks are allocated to cores based on the type of tasks as well as cores. There are two types of tasks and cores in terms of power, namely, low power task/core and high power task/core. In general, a task is called low/high power task if it has the following properties.

1) Lower/higher data rate, hence lower/higher energy spent per unit of time to process the data.
2) Less/more complex in nature, so numbers of computation steps are less/more which consumes low/high energy.

Manuscript received April 8, 2018; accepted June 8, 2018. Date of publication June 11, 2018; date of current version February 26, 2019. This manuscript was recommended for publication by Y. Chen. *(Corresponding author: Kalyan Baital.)*

K. Baital is with the National Institute of Electronics and Information Technology, Kolkata 700032, India, and also with the A. K. Choudhury School of Information Technology, University of Calcutta, Kolkata 700 098, India (e-mail: kalyan_baital@yahoo.co.in).

A. Chakrabarti is with the Faculty of Engineering and Technology, University of Calcutta, Kolkata 700 098, India, and also with the A. K. Choudhury School of Information Technology, University of Calcutta, Kolkata 700 098, India (e-mail: amlanc@ieee.org).

Digital Object Identifier 10.1109/LES.2018.2846666

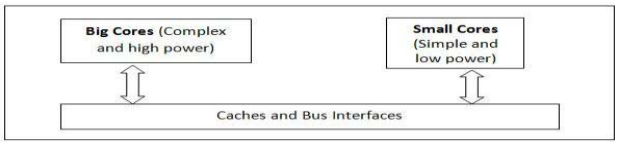

Fig. 1. Block diagram of heterogeneous multicore system.

As per our assumption, a core is called low/high power core if its circuit complexity is less/high or operates at lower/higher clock frequency or has low/high numbers of functional blocks in hardware.

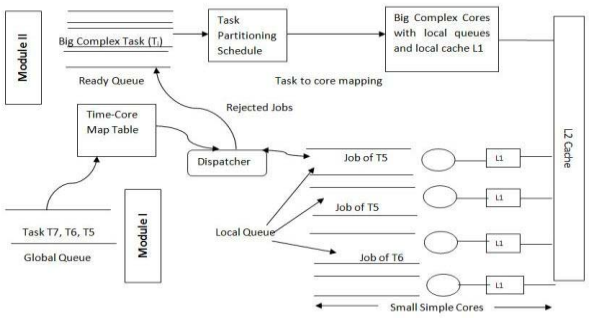

Fig. 2. Architectural model of heterogeneous multicore system—simple small cores and big complex cores.

B. Problem Solution With Description

Heterogeneous multicore consists of a few high-power bigger complex cores, coupled with several simpler, small low-power cores [2], [13] as illustrated in Fig. 1.

We propose following heterogeneous system model which extends the existing homogeneous scheduling model as explained in work [4].

1) Proposed System and Task Model–System Model: The proposed heterogeneous multicore architecture as depicted in Fig. 2. consists of following two modules.

1) *Module I:* Consisting of simple small cores with low power. The cores are homogeneous in nature. This module is dedicated for executing low power tasks and the module is operational upto a certain low power level ($0 < P <= P_l$ where P is the power level and P_l is the maximum power level to operate the cores).
2) *Module II:* Consisting of complex big cores with high power and is responsible for executing high power tasks and the module is operational from a certain high power level ($P_l < P <= P_m$ where P_m is the maximum power level to operate the cores).

Total number of cores is C where $C = C_s + C_b (C_s =$ total number of small cores; $C_b =$ total number of big cores).

Each small and big core has their own first level cache $L1$ and a second level $L2$ cache which is shared by all cores. The architecture of each module is described below.

a) Module I: The architecture of Module I consists of a number of small low power similar types of cores as described in the work [4]. In addition to this we have incorporated a run time dispatcher in the architecture of [4]. The roles of the online dispatcher are as follows:

1) to dispatch the matching jobs into small cores;
2) to migrate the rejected jobs from small cores into big cores for execution.

These simple small cores are designed for high parallel performance with low power.

b) Module II: The architecture of Module II has been incorporated additionally where a few big complex cores are placed for high serial performance. These cores are designed for highly specialized application-specific task with high power. Each big core has their own local queue and local cache ($L1$) to store the task for allocation into the core. Ready Queues equal to the number of big cores have been incorporated to store all the instance (or jobs) of a particular big complex task to a particular Ready Queue as we consider the partition schema where all the instances (or jobs) of a task are executed on the same core [14]. The instances (or job) of the task are next forwarded from the Ready Queue to local queue and subsequently into the big core corresponding to the Ready Queue using task partitioning scheduling [5].

c) Task Model: As per the works in [5], [6], and [15] we may consider the following task model.

Consider U_{ij} is the utilization of task T_i executing in core C_j and it can be denoted as $U_{ij} = X_{ij}/P$ where X_{ij} is the worst case execution time (WCET) of task T_i when executing on core C_j to meet its deadline and P is the period of T_i for all i. Further assume that each task T_i has three parameters: 1) period P for T_i (for all i); 2) WCET on big core X_{ijB}; and 3) WCET on small core X_{ijS}, where $X_{ijB} <= X_{ijS}$ for all T_i and the utilization of the big cores satisfies $0 <= X_{ijB}/P <= 1$ when $X_{ijB} < P$. If $X_{ijB} > P$, then we set U_{ij} to ∞ to indicate that task T_i cannot be mapped into the core C_j. This results in an utilization matrix $U = [U_{ij}]_{nxm}$ where n is the number of tasks and m is the number of cores.

When the utilization on small core, $U_{ij} = X_{ijS}/P > 1$, then small cores are not suitable to accommodate the jobs into cores to meet the computational demands of the workload. In such cases the small cores reject the jobs and migrate these jobs from small cores into big cores through online dispatcher.

2) Working Principle: The working principle for the simple small core as depicted in Module I of Fig. 2 is same as described in the work [4] with some additional functions of run-time dispatcher. Based on the pseudo-release time of new job of task (these tasks are low power tasks) in the Global queue, searching to time-core map table is done and accordingly forwarded to local queue of the matching cores (small simple cores) through the run-time dispatcher. Subsequently the job has been allocated from local queue to small cores maintaining the constraints. The role of the run-time dispatcher is to forward the matching jobs to the local queues of the cores and migrate the rejected jobs (from Global queue or/and local queue) to the Ready Queue of the Module II. The jobs are rejected due to the following reasons.

1) *Case I:* No available idle core at the release of new job, which restricts the entry of the job in the local queue.

2) *Case II:* Release time of more than one job is same but number of idle cores at that time is less than number of matching jobs. In this case the jobs with highest priorities are allocated to cores and the rest are rejected.
3) *Case III:* Matching job cannot be forwarded to core from local queue if the job cannot be completed before starting of next instance of the existing task of the core.

We consider each of these rejected jobs as complete task (may be denoted as T_R) with only one job [16] and these are stored in Ready Queue of Module II with readjusting task parameters as per assumptions of scheduling tasks into big complex cores in Module II.

The working principle for the big complex cores as depicted in Module II has been taken from the existing work [5], [6] with additional functions of allocating T_R of Module I as well. The following assumption can be drawn for scheduling tasks (including T_R) into these big cores.

Assumptions:

1) At a particular instance of time, a set of periodic big complex tasks or/and T_R share the same arrival time and deadline in the Ready Queue.
2) All tasks (including T_R) have same period T, and in each period they have the same arrival time 0.
3) We consider partitioned scheduling, in which each task (including T_R) is taken from Ready Queue and is assigned onto a suitable big core.
4) As we consider partitioned scheduling, task migration among cores is not allowed.
5) We also assume that all tasks are independent.

Now, to fulfill the objective of scheduling tasks into big complex cores, we adopted partition scheduling from the work [5], [6] for mapping tasks (including T_R) from Ready Queue into suitable big cores. The key concepts of the work are as follows.

1) Consider that we have given a set of n tasks, a set of m processors and the utilization matrix $[U_{ij}]$; $i = 1, 2, \ldots, n; j = 1, 2, \ldots, m$.
2) For any mapping of the n tasks on m processors ($n >= m$), we find partition matrix $R = [R_{ij}]_{nxm}$ as output of the partition algorithm where R_{ij} is the indicator variable; $i = 1, 2, \ldots, n; j = 1, 2, \ldots, m$.
3) R_{ij} is equal to 1 if task T_i is assigned to be executed on core C_j and 0 otherwise.
4) The algorithm may not use all the given m cores for assigning the tasks, that is, task partitioning may group the tasks into less than m cores.
5) We can formulate above partitioning problem as following ILP problem:

$$1. \sum_{j=1}^{m} R_{ij} = 1 \quad (i = 1, 2, \ldots, n)$$

$$2. \sum_{i=1}^{n} (R_{ij}.U_{ij}) \leq U(j = 1, 2, \ldots, m).$$

6) As a solution, we have a feasible matrix for task (including T_R) allocation into big cores as output.

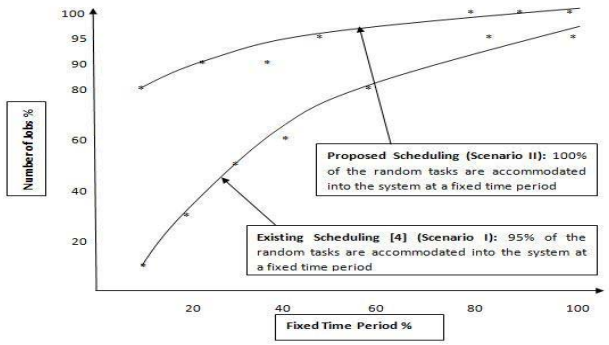

Fig. 3. In the proposed model, almost 100% of all the new random tasks are accommodated (Scenario II).

IV. Verification of the Proposed Model

Random low power tasks are stored in the Global queue of Module I and are accommodated in the simple small cores of the module as per the working principle of [4]. Module I has been verified through simulation in C programming environment [4]. It has been shown in the work that the Module I performs excellently well in all the conditions and the CPU utilization of the module at a fixed time interval is greater than 95%. In case of Module II of the proposed model, two scenarios are there.

1) All the instances (or jobs) of a particular complex task are stored in a particular Ready Queue corresponding to a particular big core.
2) Allocation of rejected simple low power jobs (that are migrated from Module I and considered as task T_R) from Ready Queue into big cores.

We consider the partition schema for the above two scenarios and formulate the partitioning problem as ILP problem. As per the work in [5] and [6] we find the solution to the problem and there exist a complete feasible mapping of big complex tasks into big cores—this verifies that all the tasks (big complex as well as T_R that are taken from Ready Queue) can be accommodated into the big cores at a particular instance of time. The work also demonstrates that the utilization of the tasks assigned on cores is less than or equal to 100%, and implementing any scheduling scheme in core individually ensures that the tasks can meet the deadlines.

V. Experimental Result

We draw a graph (Fig. 3) with the experimental results shown in the above works, illustrating the variation in number of jobs getting accommodated at a fixed time period. From Fig. 3, we may conclude that our proposed model will increase the performance of the work [4] by 5% where system utilization is nearly 100% and the model can accommodate all the tasks into cores (small or big) depending upon the task types.

Further, we compare the average system utilization percentage of our proposed scheduling scheme with the average system utilization percentage of existing relatively new scheduling algorithm [17]. For comparison, we consider average utilization percentage with respect to the number of cores which we get by varying the number of cores for Module I and Module II in our proposed scheduling scheme. We plot the comparison result as shown in Fig. 4.

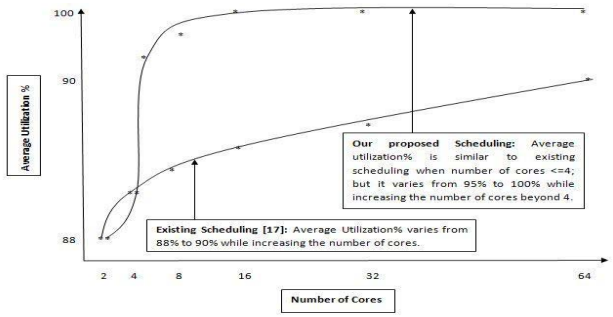

Fig. 4. Comparison of average system utilization%.

From Fig. 4, we find that the average utilizations are almost same for existing scheduling scheme as well as our proposed scheduling scheme when the number of cores is less than equal to four, i.e., when the system is made for under loaded condition. However, when the number of jobs are greater than four, i.e., when the system is made for over loaded condition, the rate of increase of CPU utilization percentage for our proposed scheduling scheme is very high (9% more in average cases) comparable to the existing scheduling model [17]. It may be concluded that the total system utilization of our proposed scheduling scheme is much more (9% more in average cases) than that of other existing scheduling scheme when the number of cores is large, i.e., system is made for over loaded condition.

VI. Conclusion

This letter incorporates heterogeneity in dynamic scheduling of tasks for multicore real-time systems [4]. The proposed model has two modules—one is for small simple cores designed for high parallel performance with low power and another is for big complex cores designed for highly specialized application-specific task with high power. The working principle for the first module is same as per work in [4] with some additional functions of run-time dispatcher and the working principle for the second module has been taken from the existing work [5], [6] with additional functions of allocating rejected jobs. Both the modules perform excellently well in all the conditions and can accommodate all the tasks into small core as well as big cores depending upon the task types. The other important issues like resource sharing, load balancing, and other performance parameters will also be developed in different phases of this model development.

References

[1] D. Koufaty, D. Reddy, and S. Hahn "Bias scheduling in heterogeneous multi-core architectures," in *Proc. ACM EuroSys*, Paris, France, 2010, pp. 125–138.

[2] N. Chitlur *et al.*, "QuickIA: Exploring heterogeneous architectures on real prototypes," in *Proc. HPCA/IEEE*, New Orleans, LA, USA, 2012, pp. 1–8.

[3] G. Tong and C. Liu "Supporting soft real-time sporadic task systems on uniform heterogeneous multiprocessors with no utilization loss," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 9, pp. 2740–2752, Sep. 2016.

[4] K. Baital and A. Chakrabarti, "An efficient dynamic scheduling of tasks for multicore real-time systems," in *Advances in Computing Applications*, A. Chakrabarti, N. Sharma, and V. Balas, Eds. Singapore: Springer, 2016, pp. 31–47.

[5] W. Munawar *et al.*, "Peak power management for scheduling real-time tasks on heterogeneous many-core systems," in *Proc. ICPADS/IEEE*, Hsinchu, Taiwan, 2014, pp. 200–209.

[6] S. Baruah, "Task partitioning upon heterogeneous multiprocessor platforms," in *Proc. RTAS/IEEE*, Toronto, ON, Canada, 2004, pp. 536–543.

[7] M. A. Kinsy and S. Devadas "Algorithms for scheduling task-based applications onto heterogeneous many-core architectures," in *Proc. HPEC/IEEE*, Waltham, MA, USA, 2014, pp. 1–6.

[8] C. Tan, T. S. Muthukaruppan, T. Mitra, and L. Ju, "Approximation-aware scheduling on heterogeneous multi-core architectures," in *Proc. ASP-DAC/IEEE*, Chiba, Japan, 2015, pp. 618–623.

[9] F. Yan, L. Cherkasova, Z. Zhang, and E. Smirni "DyScale: A MapReduce job scheduler for heterogeneous multicore processors," *IEEE Trans. Cloud Comput.*, vol. 5, no. 2, pp. 317–330, Apr./Jun. 2017.

[10] Y. Li, J. Niu, X. Long, and M. Qiu, "Energy efficient scheduling with probability and task migration considerations for soft real-time systems," in *Proc. ComComAp/IEEE*, Beijing, China, 2014, pp. 287–293.

[11] D. Liu, J. Spasic, P. Wang, and T. Stefanov "Energy-efficient scheduling of real-time tasks on heterogeneous multicores using task splitting," in *Proc. Int. Conf. RTCSA/IEEE*, Daegu, South Korea, 2016, pp. 149–158.

[12] S. Saha, Y. Lu, and J. S. Deogun "Thermal-constrained energy-aware partitioning for heterogeneous multi-core multiprocessor real-time systems," in *Proc. Int. Conf. RTCSA/IEEE*, Seoul, South Korea, 2012, pp. 41–50.

[13] F. A. Bower, D. J. Sorin, and L. P. Cox "The impact of dynamically heterogeneous multicore processors on thread scheduling," *IEEE Micro*, vol. 28, no. 3, pp. 17–25, May/Jun. 2008.

[14] Y.-S. Chen, H. C. Liao, and T.-H. Tsai, "Online real-time task scheduling in heterogeneous multicore system-on-a-chip," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 118–130, Jan. 2013.

[15] S. Xu, I. Koren, and C. M. Krishna, "Thermal-aware task allocation and scheduling for heterogeneous multi-core cyber-physical systems," in *Proc. Int. Conf. Embedded Syst. Cyber Phys. Syst. Appl. (ESCS)*, Las Vegas, NV, USA, 2016, pp. 10–16.

[16] D. Chen, A. Mok, and S. Baruah "On modeling real-time task systems," in *Lectures on Embedded Systems. EEF School 1996* (LNCS 1494), G. Rozenberg and F. W. Vaandrager, Eds. Heidelberg, Germany: Springer, 1998.

[17] A. S. Radhamani and E. Baburaj, "Performance efficient heterogeneous multi core scheduling strategy based on genetic algorithm," *ARPN J. Eng. Appl. Sci.*, vol. 8, no. 1, pp. 26–32, 2013.