Hard Real-time Scheduling Framework on CellBE

Bach D. Bui, Deepti K. Chivukula , Marco Caccamo, Lui Sha
Department of Computer Science
University of Illinois at Urbana Champaign
{bachbui2, dchivuk2, mcaccamo, lrs}@uiuc.edu

Introduction

In this research, we propose a real-time scheduling framework on the CellBE microprocessor [5], in order to handle the *problem of unpredictable task execution time* in hard real-time systems. A hard real-time system has strict requirements on timing behavior of tasks: (1) task worst case execution times (WCET) must be reliably estimated and (2) task deadlines must be guaranteed. Unfortunately, the modern computer architecture has been designed in such a way that the estimation of a tight bound WCET is very difficult. The problem lies in the uncontrollability of both the Front Side Bus (FSB), which is shared between CPU and DMA-enabled peripherals, and the cache. In a cache-based multitasking microprocessor system, a task execution time can be unexpectedly extended by the execution of other tasks or DMA-enabled peripherals [3, 4]. In our experiments [3], we observed that the extension in the execution time to be as high as 44%. This problem is potentially more severe in multiprocessor systems as they have more entities simultaneously competing for bus access.

Recently, there have been several advanced multiprocessor systems being developed, possessing real-time-amicable features. Among many, the Cell Broadband Engine Architecture (CellBE) [5] is most distinguished for its remarkable high performance and low cost. The CellBE is a heterogeneous chip multiprocessor. It consists of an IBM 64-bit Power Architecture core called the Power Processing Element (PPE), augmented with eight co-processors called Synergistic Processor Elements (SPE), which contain 256KB of Local Storage (LS). All processing units and peripherals communicate with main memory and other units through the Element Interconnect Bus, which acts as the FSB (Fig. 1 [5]). The EIB consists of four data rings, two of which run clockwise and the other two counter-clockwise. Each ring can allow up to three non overlapping concurrent data transfers, which permits the bus system to have a maximum of twelve concurrent transactions. More importantly, the EIB access is software controllable.

The CellBE offers two innovative features to avoid *the unpredictable task execution time problem*: (1) fast controllable on-chip local memory instead of unpredictable cache and (2) software-controllable FSB. However, the architecture introduces a new challenge of *scheduling tasks together with their bus accesses*. The goal of our research is to solve this challenge. We propose a scheduling framework that takes advantage of the multilevel parallelism of CellBE i.e. multi-processor and bus-level parallelism. To the best of our knowledge, there have been no real-time schedulers that have dealt with this scheduling problem.

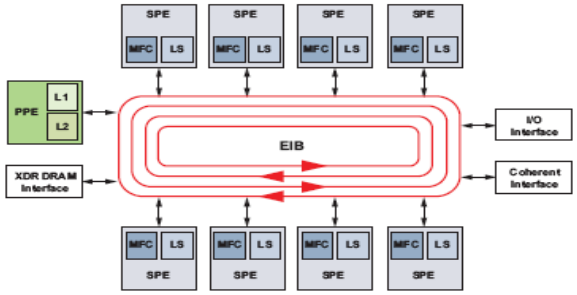

Fig 1: CellBE Architecture

We consider real-time systems where the task timing parameters (i.e. deadlines, periods), and the data *transactions* between tasks, tasks and main memory, and peripherals and main memory are specified. Each transaction has two *endpoints* which could be the tasks, the main memory or the peripherals. The proposed framework consists of a *task schedule* on the processing units (i.e. SPEs) together with a *transaction schedule* on the buses. In general, a task and its transactions on the bus are temporally and spatially dependent. The former occurs, for example, when a task loads data from memory (i.e. transaction execution) before processing it (i.e. task execution). The latter is due to the fact that for a transaction which has a task as one of its *endpoints*; the transaction route is determined by the SPE to which the task is allocated. With a practical and reasonable assumption, our task model proposed in the next section will decouple the temporal relations. The spatial relation, however, can not be isolated. Moreover, it determines the degree of parallelism on the bus, because transactions which are spatially overlapping can not be executed concurrently. For this reason, the placement of the tasks on SPEs has to take this factor into account. We call such a *task scheduling*, a *location-aware scheduling*. With regard to the *transaction scheduling*, we have proven that a PFair [2] schedule is optimal. However, due to the constraints induced by spatially overlapping transactions, it is not possible to use the PFair's original proof [2]. Therefore, we derived another proving technique by modifying the original one.

In the following sections we will discuss the proposed task model and the scheduling framework.

Task Model

We propose a task model that consists of a set of n independent periodic tasks $\mathrm{T} = \left\{ \tau_i = \left(e_i, p_i, \gamma_i^{in}, \gamma_i^{out} \right) : i \in [n] \right\}$, where the task τ_i has an execution time e_i, a period p_i, and contains an infinite number of jobs $\tau_i = \left\{ \tau_i^j : j \in [k] \right\}$. The

required data transactions of a task to other tasks and main memory are modeled as γ^{in} and γ^{out}. γ_i^{in} is the input data transaction that is needed for the execution of a job of the task τ_i and γ_i^{out} is the output data transaction that occurs after a job of task τ_i ends. To decouple the temporal dependence between a task and its transaction, we use a double buffering mechanism. This means that input and output data for job τ_i^{j+1} is being transferred during the period of jobs τ_i^j and τ_i^{j+2}, respectively. This implies that the period of a transaction equals the period of its tasks. We argue that this task model is practical and reasonable as it has been used in the design of real systems, especially avionic systems. Fig. 2 shows an example of task τ_i and its input and output transactions.

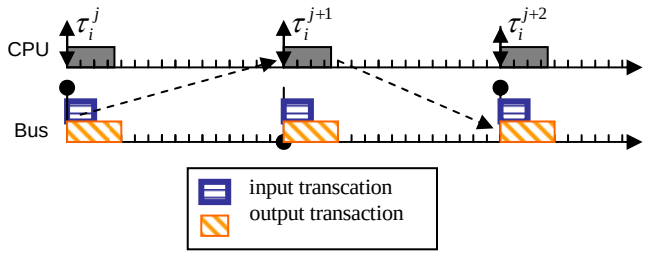

Fig 2: Task Model Example

Problem Statement

Having defined the terminologies, we are ready to formally define the problem as follows:

Given a CellBE system with a task set Υ *together with a transaction set* Γ*, find feasible schedules for* Υ *and* Γ.

A system is said to be schedulable if there exist feasible schedules for both Υ and Γ.

Scheduling Framework

Solving the abovementioned scheduling problem involves finding: (1) a schedule for the tasks on SPEs and (2) a schedule for transactions on EIB.

For task scheduling, we suggest the use of *multiprocessor partitioned scheduling algorithms*. This decision is justified by the following reasons:

1. The local-memory based microprocessor architecture has high task migration overhead since it requires both task instructions and data to be loaded into the local memory before the task can be executed. A partitioned scheduling algorithm helps to minimize this overhead.
2. The overlaps among transactions are determined by the location of their endpoints. A migration multiprocessor scheduler may cause the location of a task (i.e. a transactions' endpoint) to be changed during the course of its execution. For this reason, the overlaps among transactions in a system using migration schedulers are intractable. Consequently, the transaction scheduling problem is intractable.
3. Except for the case of *full migration, unrestricted dynamic schedulers*, the known utilization bounds of multiprocessor migration and non-migration schedulers are comparable when the number of processors is large enough [1].

It is important to notice that the traditional partitioned scheduling algorithms are not concerned with to which specific processor a task is allocated. If CPU_1 hosts tasks τ_1, τ_2, τ_3 and CPU_2 hosts tasks τ_4, τ_5, then switching τ_1, τ_2, τ_3 onto CPU_2 and τ_4, τ_5 onto CPU_1 causes no difference in terms of task schedulability. However, for the problem at hand, knowing exact task location is important for the transaction schedulability. A bad allocation occurs if all transactions are spatially overlapping, because this arrangement reduces the degree of bus parallelism to one. To deal with this problem we propose a *location-aware task scheduling algorithm* that minimizes transaction overlaps under the task deadline constraints.

Assuming tasks are already allocated onto SPEs and transaction endpoints are specified, we propose a solution for scheduling transactions on the bus. For a bus system B and a transaction set Γ on B, we define an *equivalent* homogeneous multiprocessor system M and an *equivalent* task set Λ on M respectively. We have proven that given a feasible schedule for Λ on M, we can construct a feasible schedule for Γ on B in polynomial time and vice versa. We therefore can reduce the problem of bus scheduling into the problem of multiprocessor scheduling, given the task dependence constraints. Two tasks are said to be dependent (cannot be scheduled concurrently on two processors) if their two respective transactions are spatially overlapping. Although this dependent task model departs from the independent one, which was used with the original PFair scheduler [2], we have proven that PFair scheduling is still optimal under the new constraints. The proof technique uses the argument on the existence of an *integral* optimal solution in linear programming.

References

[1] I. J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, S. Baruah, *A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms*, Handbook of Scheduling: Algorithms, Models, and Performance Analysis, Chapman Hall/ CRC Press. 2004.

[2] S. Baruah, N. Cohen, G. Plaxton, D. Varvel, *Proportionate progress: A notion of fairness in resource allocation*, Algorithmica 15(6), June 1996.

[3] R. Pellizzoni and M. Caccamo, *Towards the Predictable Integration of Real-Time COTS based Systems*, Proceedings of the 28th IEEE RTSS, December 2007.

[4] B. D. Bui, M. Caccamo, L. Sha, J. Martinez, *Design and Evaluation of a Cache Partitioned Environment for Real-Time Embedded Systems*, UIUC Technical Report 2008.

[5] http://www.research.ibm.com/cell/.