Journal of Systems Architecture 60 (2014) 293–304

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Colored Petri Net model with automatic parallelization on real-time multicore architectures

Chao Wang [a,b], Xiaojing Feng [a,b], Xi Li [a,*], Xuehai Zhou [a], Peng Chen [a,b]

[a] School of Computer Science, University of Science and Technology of China, Hefei, China
[b] Suzhou Institution of USTC, Suzhou, China

ARTICLE INFO

Article history:
Available online 26 September 2013

Keywords:
Colored Petri Net (CPN)
Task scheduling
Multiprocessor system-on-chip (MPSoC)
Model based design

ABSTRACT

This paper proposes a novel Colored Petri Net (CPN) based dynamic scheduling scheme, which aims at scheduling real-time tasks on multiprocessor system-on-chip (MPSoC) platforms. Our CPN based scheme addresses two key issues on task scheduling problems, dependence detecting and task dispatching. We model inter-task dependences using CPN, including true-dependences, output-dependences, anti-dependences and structural dependences. The dependences can be detected automatically during model execution. Additionally, the proposed model takes the checking of real-time constraints into consideration. We evaluated the scheduling scheme on the state-of-art FPGA based multiprocessor hardware system and modeled the system behavior using CPN tools. Simulations and state space analyses are conducted on the model. Experimental results demonstrate that our scheme can achieve 98.9% of the ideal speedup on a real FPGA based hardware prototype.

1. Introduction

Real-time multiprocessor system-on-chips (MPSoCs) are commonplace recently, and the number of processors on an MPSoC is growing steadily. However, the performance potential of MPSoCs cannot be tapped out unless applications running on them have been highly parallelized. One common approach of parallelizing applications is task scheduling. Task scheduling in real-time multiprocessor systems is to assign tasks to different processors, allowing them execute in parallel so that all time constraints imposed on tasks are satisfied.

Parallel programming models have been widely used to exploit parallelism, such as OpenMP [1], MPI [2], Intel's TBB [3], CUDA [4], OpenCL [5] and Cilk [6]. They are perceived to result in high performance gains. However, the major drawback of these models is that they impose too many burdens to programmers, and therefore lead to low application-development productivity. To lighten the burden of programmers, automatic parallelization technologies have been intensely studied. Recent studies such as task superscalar [7] explore a new research direction into extracting Task Level Parallelism (TLP).

There are numerous approaches for modeling a scheduling system on MPSoCs, such as directed graph, Petri net, UML activity diagram and so on. Of these approaches, model based design provides a promising approach for tackling these problems. Benefiting from verification and simulation on models, design errors can be detected as soon as possible and various paradigms of designs can be evaluated in early design phases. Furthermore, by performing time-based simulations on models, any violation of time constraints can be revealed prior to the implementation of real-time systems.

Model based design methodology has been widely applied in solving task scheduling problems for real-time multiprocessor systems. Nonetheless, little research has been conducted on modeling dynamic scheduling schemes. It is mainly because there are two vital behaviors hard to describe using modeling languages:

(1) *Dependence detecting:* In a dynamic scheduling scheme, tasks are usually reordered to perform out-of-order execution for exploiting task-level parallelism. However, the scheduling scheme can lead to correct task execution order only when all inter-task dependences are maintained. It demands that system models are able to describe and detect different types of dependences at run time. Since the out-of-order task execution is characterized by concurrent and asynchronous, It is difficult to model the dynamic behavior of dependence detecting. Most related work takes inter-task dependency as a priori, while rarely addresses the problem of dynamic dependence detecting.
(2) *Task dispatching:* Task dispatching here refers to assigning a particular processor to a task when there are multiple processors capable to execute it. Task dispatching strategy also has significant effect on system performance, since the exe-

* Corresponding author. Address: School of Computer Science, University of Science and Technology of China, Room 421, Electronic Building 3rd, West Campus USTC, Hefei 215123, China. Tel./fax: +86 0512 62888062.
E-mail address: llxx@ustc.edu.cn (X. Li).

1383-7621/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sysarc.2013.08.016

cution and communication costs of each individual task vary among different dispatching strategies. In a dispatching strategy, the mapping relationship between tasks and processors must be established and modified dynamically. Therefore, modeling the strategy is challenging, especially on the heterogeneity MPSoCs architectures.

In this paper, we construct CPN-based models for Task-Level Score boarding, and consider both partitioning and dependence detection simultaneously. Simulation experiments are performed on CPN Tools [8] finally.

Furthermore, we also implement a prototype system on a single Xilinx Virtex-5 FPGA board and perform a series of experiments to compare the performances between task superscalar and our proposed Task-Level Scoreboarding.

We claim the following contributions:

(1) This paper presents a scheduling scheme for MPSoCs, called Task-Level Scoreboarding, which can automatically detect inter-task dependences and extract TLP by allowing tasks to execute out-of-order. Compared with state-of-art approaches, our proposed approach introduces lower time overhead and achieves more significant performance.
(2) We construct a CPN-based model for the proposed scheduling scheme. The model can identify different types of dependences and simulate both scheduling and partitioning processes of tasks. Besides, various partitioning strategies can be evaluated based on the model. With the help of our CPN model, different design paradigms can be conveniently evaluated. This will lead to significant reduction on time-to-market.

The rest of this paper is organized as follows. Section 2 summarizes previous work related to task scheduling problems and relevant modeling methods. Section 3 presents the Task-Level Scoreboarding scheduling scheme. The CPN model of Task-Level Scoreboarding is elaborated in Sections 4 and 5 presents the heterogeneous multiprocessor prototype and experimental results. Finally, Section 6 concludes this paper and introduces future work.

2. Related work

In order to tap out the performance potential, there have been many approaches proposed to parallelize the workloads of multiprocessor systems. Recent works employ hardware techniques to reduce runtime overhead, such as carbon [9], ADM [10], task superscalar [7,11]. Especially, task superscalar proposes an abstraction of out-of-order superscalar pipelines. With the support of renaming mechanism, task superscalar is able to eliminate WAR and WAW hazards dynamically and thereby extract more parallelism at task level. However, the runtime time overhead introduced by renaming mechanism is considerable. For application workloads where WAW and WAR hazards do not arise frequently, the objective of applying renaming mechanism is not justified enough. Directed and undirected graphs provide an intuitive approach to model multiprocessor tasks informally. Various directed and undirected graphs based models for scheduling algorithms are surveyed in [12]. In these models, nodes in a graph are used to represent tasks, while the arcs represent precedence constraints or inter-task interactions. These graphs have the virtues of simple structure and graphical representation. However, although they are competent to modeling the workflows, they do not have the capability to describe data-flow and control-flow for modeled systems. Furthermore, since the modeling languages lack of formal definitions, these models usually lead to inconsistency and ambiguity in system specification. Therefore, there is a need to resort to formal methods, especially in context of complex system designs.

A timed automaton is a finite-state machine equipped with time concepts. It supports modeling of times by annotating state-transition graphs with clock variables and time guards. Transitions in an automaton are conditioned by time guards which compare clock variables with time constants, and firing a transition can affect the values of selected clock variables. This property enables timed automata to model time-dependent systems. When timed automaton was first introduced by Rajeev Alur and David Dill [13], its expressive power was strictly limited. Nevertheless, a lot of efforts have been made towards extensions of timed automata. For example, weighted/priced timed automata were introduced independently in [14,15], which extend cost information on locations and transitions. The timed automata in [16] is extended with deadlines and release times which are two common features in scheduling problems. These extensions increase the expressive power of original timed automata and are employed to model systems in, among others, scheduling problems. To name a few, the extended timed automata model in [16] is adopted in solving the problem of scheduling partially-ordered tasks on parallel machines, while weighted/priced timed automata are applied to optimal scheduling and planning problems in [17]. In these timed automata models, each task and resource must be represented by a single automaton. Since the model structures remain fixed during the execution of models, certain applications which require dynamic creation of new tasks cannot be modeled using timed automata. Furthermore, it is hard to use timed automata to model concurrent systems with shared resources [18]. Due to this limitation with respect to modeling power, the application scope of timed automata is greatly restricted.

On the contrast, Petri nets, especially CPN, have received much attention for modeling scheduling processes on multiprocessor platforms. For instance, Zuberek et al. model the scheduling of multiple tasks on distributed-memory multiprocessors using CPN [19]. The proposed model can be utilized to evaluate the influence of different model parameters on the system performance. In [20], CPN is used to build a model which formally describes the behavior of task distribution and execution within the grid environment. Based on the analysis of the model, the grid service reliability can be evaluated. Reference [21] studies the task scheduling of a robot system with temporal constraints, using timed Petri nets. [22] presents a Petri net based model of task scheduling on dynamically partitioned multiprocessor systems and performs a series of sensitivity analyses on the model. However, none of these models take inter-task data dependences into consideration.

Tavares et al. propose a model based scheduling scheme for multiprocessor systems with timing and energy constraints [23,24]. In the scheme, multiprocessor tasks are modeled using timed Petri nets. The model can describe precedence/exclusion relations among tasks. Hoheisel et al. develop a Petri net based model for workloads in the Fraunhofer Resource Grid (FhRG) environment [25,26]. Their model also considers the precedence constraints on grid tasks. Eskinazi applies timed Petri net within a reconfigurable environment and proposes a Petri net model responsible for task dispatching and relocation [27]. Dodd presents a CPN model for real-time task scheduling system of Seahawk helicopter [28]. The model is capable to monitor input/output resources of each task and detect inter-task dependences. However, it does not take task dispatching into account, since the modeled tasks have been statically bound to processors.

To the best of our knowledge, there are no Petri net based scheduling schemes addressing dependence detecting and task dispatching simultaneously. This paper takes both of these problems into