Journal of Systems Architecture 59 (2013) 1083–1094

Contents lists available at SciVerse ScienceDirect

Journal of Systems Architecture

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

Multi-objective exploitation of pipeline parallelism using clustering, replication and duplication in embedded multi-core systems

Chih-Sheng Lin [a], Chao-Sheng Lin [a], Yu-Shin Lin [a], Pao-Ann Hsiung [a,*], Chihhsiong Shih [b]

[a] Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi County 62102, Taiwan
[b] Department of Computer Science, Tunghai University, Taichung City 40704, Taiwan

ARTICLE INFO

Article history:
Available online 11 June 2013

Keywords:
Embedded multicore system
Latency
Throughput
Power consumption
Design patterns

ABSTRACT

With the popularity of mobile device, people require more computing power to run emerging applications. However, the increase in power consumption is a major problem because power is quite limited in embedded systems. Our goal is to consider power consumption along with latency and throughput. We proposed a heuristic algorithm, called *Parallel Pipeline Latency Optimization for high performance embedded systems* (*PaPiLO*), based on clustering, replication and duplication, to minimize latency under power and throughput constraints. Experimental results show our method can get 15% latency reduction and 10% improvements for random task graphs and MPEG-2 decoder, respectively.

1. Introduction

Nowadays, embedded systems have become ubiquitous in our daily life. Since embedded processors need more computing power to process emerging applications, including network, digital signal processing, and multimedia, embedded multicore processors were introduced. For example, Freescale 8641D (dual-core), ARM Cortex-A9 MPCore (supporting 2 to 4 cores), Plurality's Hypercore processor (capable of supporting 16 to 256 cores), and Tilera's TilePro64 processor (64 cores) are state-of-the-art processors [2]. It is believed that embedded multicore systems will become a mainstream in the near future. A recent example is that the launch of the first quad-core *smartphone* was officially announced.

From the programmers' perspective, parallel programming plays a key role in the multicore era. That is, using the appropriate parallel design patterns for the target application can affect the performance of multicore processors. Three common parallel programming paradigms investigated in previous studies include data parallelism, task parallelism, and pipeline parallelism [3–6]. Data parallelism takes a given data set and applies the same task to data items in parallel. Task parallelism takes a sharing data set and applies different tasks to each data item of the data set in parallel. Since it is hard to exploit only pure data parallelism or pure task parallelism, pipeline parallelism is considered as an emerging paradigm which uses data and task parallelism in a specific data flow pattern.

Pipelined workflows can be classified to *single-path and multi-path* pipelined workflows. As shown in Fig. 1, a multi-path pipelined workflow can be considered as a combination of several single-path pipelined workflows. Both of them illustrate the task dependence graphs, each node of which is represented as a stage. Compared to the single-path pipelined workflows, the multi-path pipelined workflows can exploit task parallelism naturally. However, multi-path pipelined workflows are more sophisticated to tackle than single-path ones since there are a larger combinatorial space to explore while searching for feasible and efficient solutions. In this paper, we assume our target applications as multi-path pipelined workflows.

To measure performance of a given pipelined workflow, *latency* and *throughput* are typical performance related metrics. Latency is the maximum time a data item spends in the pipelined workflow, and throughput is the number of data items processed per time unit. To process data items in parallel, two basic approaches are commonly used. One approach uses data parallelism to process one single data item in parallel, that is, a data item is cut into small pieces of data, each of which is processed by the corresponding core. The other approach uses task and pipeline parallelism to process multiple data items in parallel. For task parallelism, the same data are processed by inter-independent tasks concurrently. As to pipeline parallelism, different segments of data are processed concurrently in multiple stages which are multiple tasks. The first approach can achieve minimal latency, while the second approach can increase the throughput but worsen the latency. Tradeoff between latency and throughput can be explored by using parallel loop (data parallelism) to reduce latency, parallel tasks (task parallelism) to increase throughput, and parallel pipeline (pipeline parallelism) to hide latency [7]. In addition, the increment of power consumption is also a significant problem, especially for embedded

* Corresponding author. Tel.: +886 5 272 0411x33119; fax: +886 5 272 0859.
E-mail address: pahsiung@cs.ccu.edu.tw (P.-A. Hsiung).

1383-7621/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sysarc.2013.05.024

systems. In this paper, we investigate not only the performance optimizations but also the influence of power consumption on the above mentioned three types of parallelism for multi-path pipelined workflows.

In order to meet performance and power consumption requirements for a given pipelined application, we exploit three parallel design patterns: *clustering*, *replication*, and *duplication*. Clustering can avoid heavy communication costs by mapping two or more communicating tasks to the same core. Since clustering does not require extra cores to process tasks in parallel, it can reduce the power consumption; however, it reduces the throughput and increases the latency due to the reduced task parallelism.

Fig. 2 illustrates a motivating example, the core allocation, and the values of latency, throughput, and power consumption, without using any patterns. Figs. 3–5 are examples using clustering, replication, and duplication, respectively. We have an application which can be represented as a pipelined workflow, two data items to be processed, and three cores. This application has three stages, namely t_1, t_2 and t_3, whose execution times are 12, 5 and 6 units of time, respectively, and has three communication links between each pair of consecutive stages, namely $d_{1,2}, d_{1,3}$ and $d_{2,3}$, whose communication times are 10, 7 and 3 units of time, respectively. To estimate the power consumption, we use a metric, denoted as α, which is defined as the ratio of the power consumed for inter-core communication and the power consumed for computation on a core. The value of α depends on the target platform. Here we assume the value of α is 0.5 for calculating the power consumption in the motivating examples.

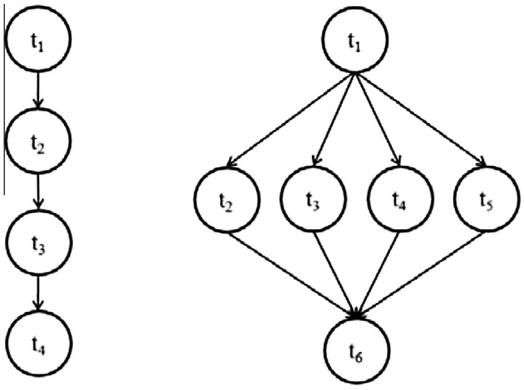

Fig. 1. The single-path and multi-path pipelined workflows.

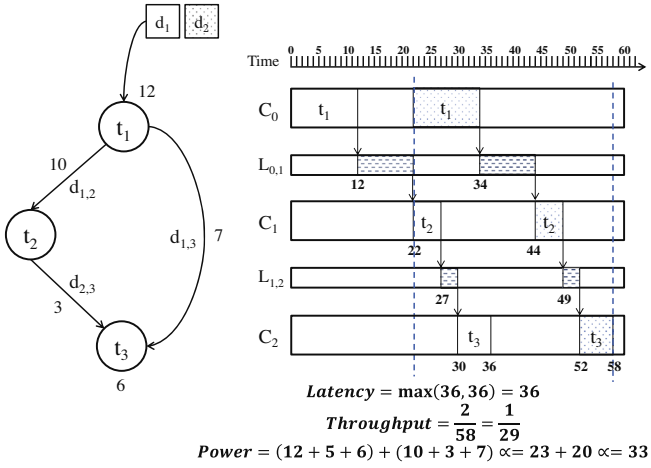

Fig. 2. A pipelined workflow example without using clustering, replication, and duplication.

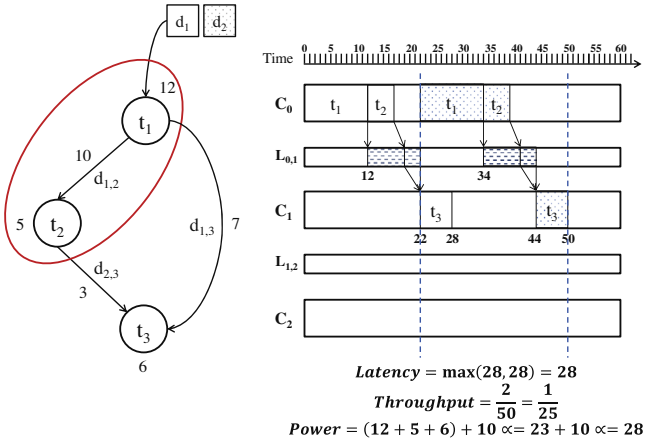

Fig. 3. A pipelined workflow example using clustering.

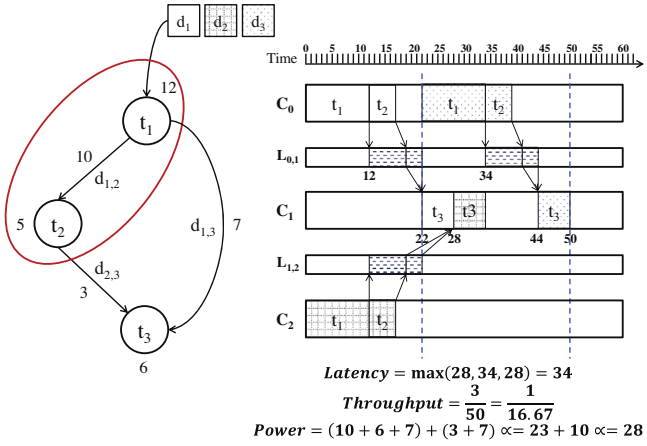

Fig. 4. A pipelined workflow example using replication.

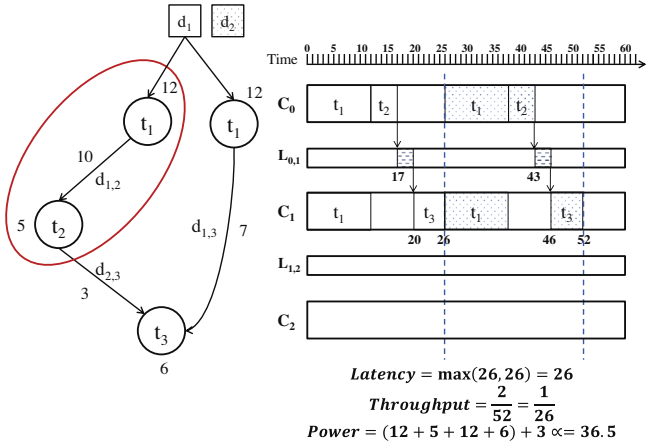

Fig. 5. A pipelined workflow example using duplication.

The core dispatch strategy of tasks is assume to allocate each cluster on an independent core. Therefore, as shown in Fig. 3, t_1 is clustered with t_2 to avoid the large communication cost in $d_{1,2}$ so that it reduces the latency.

Replication can be used to increase throughput by adding replicas of some tasks. Since throughput is limited by data transfer rate, tasks could be replicated to process the next data items if there are