Microprocessors and Microsystems 42 (2016) 165–179

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

Pipelining data-dependent tasks in FPGA-based multicore architectures

Ali Azarian*, João M.P. Cardoso

Faculty of Engineering, University of Porto and INESC-TEC, Rua Dr. Roberto Frias, Porto, Portugal

ARTICLE INFO

Article history:
Received 15 May 2015
Revised 5 October 2015
Accepted 8 February 2016
Available online 15 February 2016

Keywords:
Multicore architectures
Task-level pipelining
FPGA
Producer/consumer data communication

ABSTRACT

In recent years, there has been increasing interest in using task-level pipelining to accelerate the overall execution of applications mainly consisting of producer/consumer tasks. This paper proposes fine- and coarse-grained data synchronization approaches to achieve pipelining execution of producer/consumer tasks in FPGA-based multicore architectures. Our approaches are able to speedup the overall execution of successive, data-dependent tasks, by using multiple cores and specific customization features provided by FPGAs. An important component of our approach is the use of customized inter-stage buffer schemes to communicate data and to synchronize the cores associated with the producer/consumer tasks. We propose techniques to reduce the number of accesses to external memory in our fine-grained data synchronization approach. The experimental results show the feasibility of the approach in both in-order and out-of-order producer/consumer tasks. Moreover, the results using our approach reveal noticeable performance improvements for a number of benchmarks over a single core implementation without using task-level pipelining.

1. Introduction

Techniques to speedup processing are becoming more and more important. Task-level pipelining (TaLP) is an important technique for multicore based systems, especially when dealing with applications consisting of producer/consumer *(P/C)* tasks (see, e.g., [1]). It may provide additional speedups over the ones achieved when exploring other forms of parallelism. In the presence of multicore based systems, TaLP can be achieved by mapping each task to a distinct core and by synchronizing their execution according to data availability. It can accelerate the overall execution of applications by partially overlapping the execution of data-dependent tasks (herein: Computing Stages).

Many applications, such as image/video and signal processing, are structured as a sequence of data-dependent computing stages, use the P/C pair communication paradigm, and are thus amenable to pipelining execution [2,3]. Using TaLP, a consumer computing stage (e.g., consisting of a loop or a set of nested loops) may start execution, before the end of the producer computing stage, based on data availability. Performance gains can be achieved as the consumer can process data as soon as it becomes available.

There are two types of data synchronization granularity between the producer and consumer: *Fine-grained* and *Coarse-grained*. In fine-grained schemes, each data element is used to synchronize computing stages. In coarse-grained data synchronization schemes, instead of each data element, chunks of elements or an entire array of elements (e.g., an image) is considered to synchronize computing stages.

The simplest implementation of TaLP uses a FIFO channel between cores implementing P/C pairs. The FIFO can store an array element or a set of array elements to establish data synchronization between the producer and consumer. FIFO is sufficient when the order of producing data is the same as the order of consuming data (referred herein as in-order data communication pattern or simply in-order). In this case, the data communication between the producer and consumer can use a FIFO storing one data element in each stage. Although using FIFO channels between producers and consumers is an efficient solution for in-order P/C pairs, it may not be efficient or feasible for out-of-order P/C pairs and it might be necessary to use other data communication mechanisms [4].

In the presence of out-of-order P/C pairs, recent approaches address software-based TaLP (see e.g., [4,5] and [6]). In those approaches, an extra storage and buffer memory are introduced, based on the order of the communication pattern between the producer and consumer, determined at compile time [4]. The data communication in these studies uses unbounded FIFOs which may prevent them to be used in a number of implementations. In addition, they use memory access reordering techniques. One of the approaches for TaLP including support for out-of-order P/C pairs has been introduced in [5] in the context of application specific architectures and using a fine-grained execution of data-dependent tasks.

* Corresponding author. Tel.: +351962868369.
E-mail addresses: azarian@fe.up.pt, azarian.ali@gmail.com, pro10002@fe.up.pt (A. Azarian), jmpc@acm.org (J. M.P. Cardoso).

http://dx.doi.org/10.1016/j.micpro.2016.02.008
0141-9331/

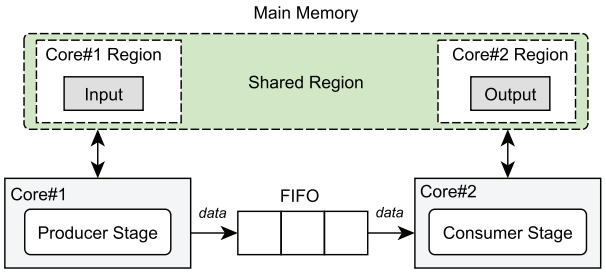

Fig. 1. Fine-grained data synchronization scheme using a FIFO between P/C pairs.

Although approaches to pipeline sequences of data-dependent loops have been addressed previously (e.g., [6–8]), the work presented here is novel in three main aspects. First, we use a fine-grained and coarse-grained data-driven synchronization schemes between P/C stages of the pipeline. The implementation uses a hash-based scheme to limit local buffer size. Other techniques have focused on finding appropriate sized synchronization buffers [2] to enforce the same P/C order thus sacrificing concurrency. Our fine- and coarse-grained synchronization schemes are similar in spirit to the empty/full tagged memory scheme used in the context of shared memory multiprocessor architectures (see, [9,10]). Second, the control scheme decouples the control units of each stage and uses inter-stage buffers (ISB) to signal the availability of data elements to the subsequent stage. This approach allows out-of-order execution of loop iterations between tasks only constrained by data dependencies. The overall benefit of these features is that we are able to achieve almost the theoretical speedup and to reduce the size of the buffers to communicate data between computing stages. Lastly, we describe the application of this technique in the context of configurable multicore architectures showing how TaLP can be applied to multicore architectures and the impact on the results of different customized inter-stage buffers.

This paper makes the following specific contributions:

- Presents a technique for pipelining the execution of sequences of data-dependent loops using fine- and coarse-grained synchronization.
- Presents customized multicore architectures for the inter-stage communication to achieve pipelining execution of P/C pairs.
- Presents a technique to predict the presence of data previously produced in external memory and thus reduce external memory accesses.

The remainder of this paper is organized as follows. In Section 2, we describe TaLP. Section 3 presents our approach including fine-grained and coarse-grained data synchronization schemes. Section 4 presents optimization techniques for our approach. Section 5 presents experimental results. We include an overview over related work in Section 6. A discussion about our techniques and their impact is presented in Section 7. Finally, Section 8 concludes the paper.

2. Task-level pipelining (TaLP)

To pipeline computing stages, the producer and consumer can be implemented in a multicore architecture as shown in Fig. 1. In this scheme, computing stages are split into cores; e.g., one core as a producer and the other core as a consumer. For the main memory, the architecture can use a shared-memory or distributed memory. Each computing stage can be a producer for the next stage, a consumer of the previous stage or both [11]. In this case, we split the producers and consumers in cores by considering the dependencies between the stages. In Fig. 1, we consider only one P/C pair for simplicity.

The communication component between P/C pairs can be a simple FIFO. Reading and writing from/to the FIFO are blocked. When the FIFO is full, the producer waits to write into the FIFO. Similarly, when the FIFO is empty, the consumer waits until a data element is written to the FIFO. In this scheme, the producer sends data elements into the FIFO and the consumer reads data from the FIFO as soon as it is available. The bottleneck of the communication over FIFO channels is that if the order of consumption is different from the order of the produced data (i.e., when in presence of out-of-order data communication), the producer and the consumer stall and a deadlock can be expected.

Fig. 2 presents the kernel of a Gray-Histogram code which consists of two computing stages. The first stage transforms an RGB image to a gray image with 256 levels and stores the output into an array. The second stage reads the gray image and determines its histogram. These two stages can be split into producer and consumer. The producer transforms the input image from RGB to gray and the consumer computes the histogram of the gray image. This kernel has an in-order communication pattern between the producer and consumer. In this case, a simple FIFO can be used to communicate data between the two stages and to achieve the fine-grained synchronization between P/C pairs. In this model, the one-dimensional FIFO channel is being accessed by means of a blocking

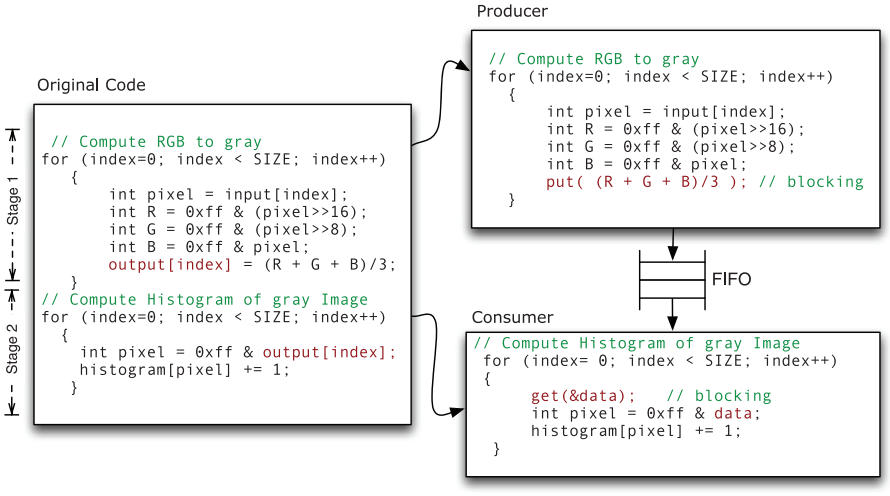

Fig. 2. Example of a simple producer/consumer pair using a FIFO channel.