Towards a Software Transactional Memory for heterogeneous CPU-GPU processors

Alejandro VILLEGAS [1], Angeles NAVARRO, Rafael ASENJO and Oscar PLATA
[a] *Universidad de Málaga, Andalucía Tech.*
Dept. Computer Architecture, 29071 Málaga, Spain
email: {avillegas, angeles, asenjo, oscar}@ac.uma.es

Abstract. The heterogeneous Accelerated Processing Units (APUs) integrate a multi-core CPU and a GPU within the same chip. Modern APUs provide the programmer with platform atomics, used to communicate the CPU cores with the GPU using simple atomic datatypes. However, ensuring consistency for complex data types is a task delegated to programmers, who have to implement a mutual exclusion mechanism. Transactional Memory (TM) is an optimistic proposal to implement mutual exclusion. With TM, shared data can be accessed by multiple computing threads speculatively, but changes are only visible if a transaction ends with no conflict with others in its memory accesses. TM has been studied and implemented in software and hardware for both CPU and GPU platforms, but an integrated solution has not been provided for APU processors.

In this paper we present APUTM, a software TM designed to work on heterogeneous APU processors. The design of APUTM focuses on minimizing the access to shared metadata in order to reduce the communication overhead via expensive platform atomics. The main objective of APUTM is to help us understand the tradeoffs of implementing a sofware TM on an heterogeneous CPU-GPU platform and to identify the key aspects to be considered in each device. In our experiments, we compare the adaptability of APUTM to execute in one of the devices (CPU or GPU) or in both of them simultaneously. These experiments show that APUTM is able to outperform sequential execution of the applications.

Keywords. transactional memory, APU processors, parallel programming

1. Introduction

Nowadays, multi-threaded programming has become essential in order to take the most of multi-core CPUs, GPUs, and many other kinds of processors. One of the main issues programmers face when designing their algorithms is the management of shared data: if two or more computing threads share a piece of data, then programmers have to implement a mechanism that ensures mutual exclusion. Usually, mutual exclusion is implemented employing locking mechanism that are inefficient (if coarse-grained locks are implemented) or hard to program (if the solution uses fine-grained locks).

[1] Corresponding Author