Journal of Systems Architecture 58 (2012) 247–256

Contents lists available at SciVerse ScienceDirect

Journal of Systems Architecture

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

Virtualization of reconfigurable coprocessors in HPRC systems with multicore architecture

Ivan Gonzalez *, Sergio Lopez-Buedo, Gustavo Sutter, Diego Sanchez-Roman, Francisco J. Gomez-Arribas, Javier Aracil

High-Performance Computing and Networking Group, Escuela Politecnica Superior, Universidad Autonoma de Madrid, 28049 Madrid, Spain

ARTICLE INFO

Article history:
Received 8 February 2011
Received in revised form 10 February 2012
Accepted 12 March 2012
Available online 23 March 2012

Keywords:
High Performance Reconfigurable Computing
Coprocessor virtualization
Multicore programming
Reconfigurable hardware

ABSTRACT

HPRC (High-Performance Reconfigurable Computing) systems include multicore processors and reconfigurable devices acting as custom coprocessors. Due to economic constraints, the number of reconfigurable devices is usually smaller than the number of processor cores, thus preventing that a 1:1 mapping between cores and coprocessors could be achieved. This paper presents a solution to this problem, based on the virtualization of reconfigurable coprocessors. A Virtual Coprocessor Monitor (VCM) has been devised for the XtremeData XD2000i In-Socket Accelerator, and a thread-safe API is available for user applications to communicate with the VCM. Two reference applications, an IDEA cipher and an Euler CFD solver, have been implemented in order to validate the proposed architecture and execution model. Results show that the benefits arising from coprocessor virtualization outperform its overhead, specially when code has a significant software weight.

© 2012 Elsevier B.V. All rights reserved.

1. Introduction

Nowadays, most computing systems have processors capable of executing several tasks in parallel. Even low-cost PCs or smartphones have multicore processors, which include two or more processing units. There are conventional x86 processors with up to twelve cores, such as AMD "Magny-Cours" [1], while non-conventional many-core architectures such as Tilera can have up to 100 cores [2]. There are also specialized architectures having hundreds of cores such as the GPUs (Graphics Processing Units) Nvidia Fermi or AMD FireStream. Typically, each core is capable of executing one task, composed by one or more threads, reaching up to thousands of concurrent threads in common GPU applications [3]. Multicore processing is the only option for performance increases, now that instruction-level parallelism and frequency scaling have reached to its limits. However, programming these multicore architectures is not always an easy task. It is necessary to parallelize the code among the different cores, using programming APIs such as OpenMP, MPI or CUDA [4].

HPRC systems [5] are composed by conventional processors together with FPGA coprocessors. In these systems, acceleration is obtained at two levels. Firstly, the application is partitioned into many parallel tasks, each running in one processor core of the HPRC machine. For this step, conventional HPC techniques apply, such as MPI or OpenMP. Secondly, the most computationally intensive kernels of each task are ported to the FPGA, which acts as a custom coprocessor. If every processor core has a FPGA coprocessor available, application partitioning is homogeneous and therefore straightforward. However, a problem arises if a 1:1 mapping between processor cores and FPGA coprocessors is not available. In this case, partition is no longer homogeneous: chances are that some tasks finish sooner because they can access a coprocessor, while others take longer time because all the computations are performed in software. This configuration brings forth severe synchronization and load balancing issues, but nonetheless it is the most typical one, because multicore processors are widespread and the number of FPGAs in a HPRC system is limited due to economic and communication constraints. This paper proposes a mechanism to transparently share FPGA coprocessing resources among all processor cores, that is, to divide coprocessors into many virtual devices. This concept has been successfully explored in other works, not only in the field of reconfigurable computing but also in other heterogeneous computing technologies. In 1998, Fornaciari et al. [6] proposed some operating-system techniques to virtually enlarge the size of the FPGA from the point of view of the applications, although they did not show any experimental results. There are no more references to FPGA virtualization until 2008, when El-Araby et al. [7,8] studied the viability of implementing virtualization of reconfigurable coprocessors in HPRC systems

* Corresponding author. Address: Department of Electronic and Communications Technology, Escuela Politecnica Superior, Universidad Autonoma de Madrid, 28049 Madrid, Spain. Tel.: +34 914976212.

E-mail addresses: ivan.gonzalez@uam.es (I. Gonzalez), sergio.lopez-buedo@uam.es (S. Lopez-Buedo), gustavo.sutter@uam.es (G. Sutter), d.sanchez@uam.es (D. Sanchez-Roman), francisco.gomez@uam.es (F.J. Gomez-Arribas), javier.aracil@uam.es (J. Aracil).

1383-7621/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sysarc.2012.03.002

by means of partial runtime reconfiguration, obtaining promising results. Later, in 2010, Chun-Hsian et al. [9] used a hardware virtualization mechanism for dynamically linking device nodes to reconfigurable hardware functions and reduce up to 12.8% the overall system execution time. In addition, virtualization of coprocessors is gaining attention with the emerge of the GPGPU technology, where the approach of [10] is sharing GPUs through the network.

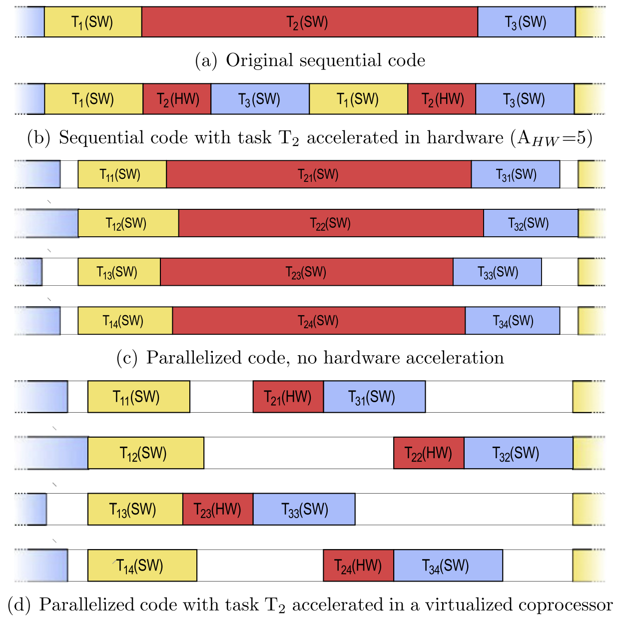
This paper presents a complementary approach to that presented in [7–9]. In those works, the FPGA is partitioned into several areas, each of them dedicated to implement a virtual coprocessor. Therefore, one single physical FPGA device appears as many coprocessors to the system. Each of these virtual FPGA coprocessors can be separately configured by taking advantage of the partial runtime reconfiguration capabilities of Xilinx' devices. While those works used spatial multiplexing to virtualize FPGA coprocessors, the approach proposed here is to use time multiplexing. That is, allow each parallel task to access the FPGA coprocessor during a fraction of the total time. Due to the high reconfiguration time of commercial FPGA devices, this time multiplexing approach is valid only for SIMD problems, where all parallel tasks execute the same code but operate on different partitions of data. However, SIMD is probably the most common mode of operation of parallel applications, and the main advantage of this approach is that can be implemented on all FPGA devices, even those not supporting partial runtime reconfiguration.

In theory, time-sharing a computation resource among elements executing parallel tasks implies synchronizing and serializing access to that resource. Consequently, time spent by the application in sequential execution is increased and, according to Amdahl's law, the maximum speedup becomes limited by that serial time. Although it seems that this situation might not be desirable, the fact is that the combination of speedups coming from both algorithm parallelization and hardware acceleration can be beneficial for many applications, even taking into consideration the waits to access the shared coprocessor. The challenge is to implement this virtualization model in such a manner that processor cores can use FPGA accelerators in a deterministic way, using an API based on simple primitives that hides virtualization details to the user. Additionally, it would be interesting that the programming model that is provided to the user is compatible with a multithreading environment commonly used in application parallelization, for example, OpenMP. This has been the goal of this paper, to implement a Virtual Coprocessor Monitor (VCM) that allows programmers to transparently use OpenMP in their HW-accelerated applications. Each of the execution threads created by OpenMP uses a virtual coprocessor, enabling that the programming model is homogeneous for all threads, regardless of the actual number of coprocessors available in the system.

The paper is organized at follows. The next section details the execution model of HPRC machines and how it enables virtualizing the reconfigurable coprocessors. The architecture of the VCM is detailed in Section 3. Section 4 shows the results of the experiments that have been devised in order to validate the proposed virtualization model. Finally, conclusions and future work are presented in Section 5.

2. Usage methodology and execution model

In reconfigurable computing systems, applications can be accelerated by porting to FPGA hardware a computationally-intensive section of the code [11,12]. Fig. 1(a) presents a representative application, consisting on a algorithm that iterates through three tasks, namely preprocessing T_1, computation kernel T_2 and post-processing T_3. When the computation kernel is ported to hardware, a significant acceleration may be obtained. For example, Fig. 1(b) presents a hardware implementation of the computation kernel that runs 5 times faster than its software counterpart. Data is preprocessed in software, then it is sent to the coprocessor where it is processed, and then it goes back to the CPU where it is postprocessed. Since preprocessing T_1 and postprocessing T_3 are still made in software, the overall speedup is smaller than the hardware acceleration that has been attained for task T_2, according to Amdahl's Law. In fact, the figure shows that the global speedup is just slightly bigger than 2, while the hardware acceleration of T_2 was 5.



(a) Original sequential code

(b) Sequential code with task T_2 accelerated in hardware (A_{HW}=5)

(c) Parallelized code, no hardware acceleration

(d) Parallelized code with task T_2 accelerated in a virtualized coprocessor

Fig. 1. Alternatives for application acceleration: original sequential code (a), hardware acceleration (b), code parallelization (c), and code parallelization with acceleration in a virtualized coprocessor (d).

On the other hand, the method for application acceleration in multi-core processors is code parallelization among several execution threads, one per core. This is shown in Fig. 1(c) for a 4-core processor, where it can be seen that the original tasks are being replicated among the cores. That is, the original task T_1 is replicated into four concurrent tasks T_{11}, T_{12}, T_{13}, and T_{14}. Parallelization is rarely perfect, so some of the parallel threads are faster than others. Additionally, synchronization is typically needed when each iteration ends, so faster threads have to wait for the slowest one. Even taking into account these drawbacks, acceleration is achieved because the computational load is distributed among the 4 threads concurrently running. Considering a simple model where all parallel threads take the same time, and t_1 is the time to execute T_{1i}, t_2 is the time to execute T_{2i} and t_3 is the time to execute T_{3i}, the time needed to execute the application can be expressed as:

$$t_{app} = I \cdot \frac{t_1 + t_2 + t_3}{A_P} \quad (1)$$

Where A_P is the acceleration obtained due to code parallelization, and I is the original number of iterations in the non-parallelized version of the code. Usually parallelization is not perfect, so acceleration A_P will be smaller than the number of concurrent cores N, although some rare applications may feature a superlinear behavior where A_P is bigger than N.

The approach proposed in this paper is to apply both acceleration techniques: hardware acceleration and code parallelization. This accomplished by time-sharing the FPGA coprocessor among the different threads that are being executed on processor cores. Time-sharing is possible since typical applications do not store data in the coprocessor between algorithm iterations. That is, the

دریافت فوری ← متن کامل مقاله

ISIArticles
مرجع مقالات تخصصی ایران

- ✓ امکان دانلود نسخه تمام متن مقالات انگلیسی
- ✓ امکان دانلود نسخه ترجمه شده مقالات
- ✓ پذیرش سفارش ترجمه تخصصی
- ✓ امکان جستجو در آرشیو جامعی از صدها موضوع و هزاران مقاله
- ✓ امکان دانلود رایگان ۲ صفحه اول هر مقاله
- ✓ امکان پرداخت اینترنتی با کلیه کارت های عضو شتاب
- ✓ دانلود فوری مقاله پس از پرداخت آنلاین
- ✓ پشتیبانی کامل خرید با بهره مندی از سیستم هوشمند رهگیری سفارشات