Elastic CGRA: Circumventing Hard-faults Through Instruction Migration

Ying Wang, Lei Zhang, Yinhe Han, Huawei Li, Xiaowei Li
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, P.R. China
{wangying2009, zlei, yinhes, lihuawei, lxw}@ict.ac.cn

Abstract Coarse-Grained Reconfigurable Architectures (CGRAs) are widely-adopted in embedded and high-performance computing as effective alternatives to General Purpose Processors (GPP) which suffer from severe energy-inefficiency problems. At the same time, the CGRA chips also easily fall victims to incidence of on-chip hard-faults in aggressive process-technology. In our analysis, the host processors inside the CGRAs are especially inclined to be wasted by defects, leading to a dramatic yield loss. However, we found that the intrinsic flexibility provided by the reconfigurable processing arrays (RPAs) enable us to maintain the functional integrity of the host processors affected by hardware faults. In this paper, we propose the technique of instruction migration and translation to detour the faulty logics inside the processors by leveraging the reconfigurability of RPAs, which significantly boost the hard-faults tolerance of CGRAs.

A. Introduction

Over the last years, significant change has arrived in processor workloads for the ever-increasing complexity in computing models like multimedia, scientific computing, and cloud computing, which all require powerful computing capacity in different systems. As a traditional solution, GPP is generally considered to be so energy-inefficient that it becomes increasingly unacceptable not only for embedded systems but also for certain super-computing platforms. To combat the problem, one of appealing options is so-called CGRA. This solution allows vendors or users to configure a targeted set of functions into the CGRA hardware at design time, program compile time or even program execution time. In this way, much of redundancy operations caused by software will be removed, thus the applications can make most use of the hardware to achieve energy efficiency like ASICs. These years, intensive research on reconfigurable processor has been conducted in industry and academic institutes [1~4]. Comparatively, these designs are far more silicon-efficient than FPGA assembled by CLBs and routing switches. They also avoid the complex HDL coding or high-level synthesis in FPGAs development [5]. Mostly the CGRAs are utilized to promote the performance or energy-efficiency of processors. To our knowledge, they are seldom used for fault-tolerant purpose. However, in the era of submicron technologies, hardware faults, caused by variations in critical process parameters, manufacture defects, and device aging, make the silicon chips more and more vulnerable [6]. Thus, CGRA-based processors should be able to tolerate frequently-occurred hardware failures. According to our observation, the RPAs inside the CGRAs are more elastic than the core logics in processors, because they have intrinsic faults-proof ability, which enables them to survive the distributed hard-errors in systems. Thus it is feasible to utilize the RPAs to salvage the processors that can no longer maintain functional integrity from hardware failures. As a further step, we propose instruction translation technique to migrate the critical instructions from host processor to RPA by manipulating the reconfigurable structure of RPA.

B. Decoupled fault-tolerance model for host processors and RPAs

As shown in Fig.1, a typical CGRA consists of a host processor, a RPA and the communication interface between them [7]. We assume that the fault occurrence probability of a module depends on its area and the assumed fault density. As a result, in a CGRA-based computing system, the host processor, the RPA and the on-chip memory, which occupy most of the chip, are the major victims to defects or failures. Among the three components, the host processor will probably be the most vulnerable part. Though many CGRAs, employ light-weight processor cores, such as MIPS-II, ARM9, and Microblaze, as the hosts. On one hand, these hosts consume relatively tiny portion of the silicon estate. On the other hand, the host processors have more random logics and critical resources that are hard to protect in comparison with the array-logics like SRAM memory. Therefore, hard-faults can easily influence the normal function of the host processors and eventually destroy the system.

For the memory part, there are a lot of mature solutions to protect the regular structure of SRAM. As we know, the RPAs have the similar array structure and fault-tolerant fea-
-tures as the memory. To exploit the artificial redundancy in RPAs, we use the Rippling Replacement Strategy to effectively address the problem of hard-faults [8], which is illustrated in Fig. 2. By replacing the faulty unit with the redundancy, the technique can accommodate multiple faulty elements as long as they do not exist in the same row. Thus RPAs can be orders of magnitude more reliable than the host processors of the same size.

Fig.2 The RPA under the protection of Redundant Units (RU)

C. Instruction Translation and migration

Unlike RPAs, when the host processors encounter hard-faults within certain uncritical components, they may degrade into faulty cores without function-integrity. From an architectural view proposed by M. D. Powell et al., these host processors are deemed as partially-functional cores which cannot execute specific instructions [9]. We define those fault-effected instructions as Hazard Instructions (HIs). However, the HIs can be executed smoothly on the RPA that is both reliable and flexible to act as a complement to the host processor. Thus, to hide the hardware faults from the application, we propose the instruction migration and translation mechanism by utilizing the RPA. We demonstrate the processor salvaging mechanism in the Fig. 3. After the fault-detection and diagnosis phases, the HIs set of a fault-affected processor are identified, and the correspondent binaries which instruct the RPA perform the same operations are also compiled for it. Then a small space in configure memory is occupied by the binary instructions for each HI of the host processor. At program running time, these HIs will be recognized in instruction decoding stage and mapped onto the RPA by calling the pre-determined binaries in configure memory (indexed by particular register pointers).

Fig.3 The hazard instruction migration flow

To tolerate dynamic faults occurring at run-time, an interrupt should be raised to load the specific binary patches (from the pre-compiled library for the entire ISE) into the configure memory, and to set the register pointers as well.

D. Conclusion

In this paper, we investigated a promising hard-faults tolerant mechanism in the CGRAs-based processor. Based on the observation that the RPAs can be more faults-tolerant than the core logics inside the host processors, we proposed the hazard instruction translation and migration techniques to salvage the host processors damaged by hard-faults. This technique can effectively make the whole architecture more elastic to on-line or off-line hardware faults.

References