Increasing the robustness of CUDA Fermi GPU-based systems

Increasing the robustness of CUDA Fermi GPU-based systems

Stefano Di Carlo*, Giulio Gambardella*, Marco Indaco*, Ippazio Martella†, Paolo Prinetto*, Daniele Rolfo*, Pascal Trotta*

Politecnico di Torino
Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi 24, I-10129, Torino, Italy
*Email: {*FirstName.FamilyName*}@polito.it
† Email: {*FirstName.FamilyName*}@gmail.com

***Abstract*—Nowadays, Graphical processing Units (GPUs) have become increasingly popular due to their high computational power and low prices. This makes them particularly suitable for high-performance computing applications, like data elaboration and image processing. In these fields, the capability of properly work even in presence of faults is mandatory. This paper presents an innovative approach, that combines a Software Based Self Test & Diagnosis (SBSTD) methodology with a fault mitigation strategy, to increase the robustness of a CUDA Fermi GPU-based system.**

I. Introduction

In the last years, the increasing demand for computational power resulted in more cores integrated in a single chip. In this scenario, GPUs have replaced multi-cores processors in many High Performance Computing (HPC) applications, thanks to their ability of performing a high number of parallel operations exploiting their internal Single Instruction Multiple Data (SIMD) architecture.
Such a rapid proliferation of GPUs is also due to the advent of significant programming support for developing general-purpose applications on GPUs, which allows programmers to easily solve "general" computing problems [1], in addition to the graphic ones.
Reliability of GPUs is still an open issue. In the past, GPUs have been mainly exploited for non-critical applications, such as video or image processing. In these applications, whenever corrupted pixel values caused by soft or hard faults are computed, the user experience is in general not significantly impacted since just few pixels per frame are usually corrupted [2].
However, the computational power of GPUs is becoming an attractive solution for several safety critical applications, like automotive [3], space [4], and medical [5]. In these applications the ability to continue to properly function despite the existence of faults (i.e., robustness of the system) becomes a primary requirement.
The robustness of a system is usually enhanced exploiting fault-tolerance or fault mitigation techniques. In literature, several publications provide this kind of techniques for SIMD processors [6], but, unfortunately, they are not applicable to modern GPU architectures, since they rely on a deep knowledge of the processor internal architecture.
Other techniques are completely independent from the processor internal architecture (e.g., *Check pointing-based* [7] and *Algorithm-based fault tolerance* [8]), but they usually either introduce high performance overhead, or require custom modifications to tackle the GPU internal architecture peculiarities.
To overcome these issues, we propose a combination of Software Based Self Test and Diagnosis (SBSTD) and fault mitigation methodologies in order to increase the robustness of a CUDA Fermi GPU-based system against permanent faults. Basically, by periodically running the proposed SBSTD methodology, it is possible to identify faulty Streaming Multiprocessors (SMs) [9] in the GPU under test. This information is then exploited by the fault mitigation strategy to overcome the effect of multiple faults.
When compared with the already presented methodologies, the proposed approach: (i) does not introduce any execution time penalties during the GPU fault-free execution, and (ii) it guarantees fault-free results also in presence of a high number of faults.

II. Proposed SBSTD Methodology

The proposed SBSTD methodology aims at testing each SM inside a CUDA *Fermi* GPU. It first detects the presence of permanent faults, and then localizes the faulty SM. The basic steps performed by the proposed methodology are shown in Fig. 1.

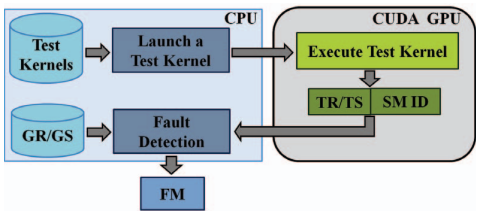

Fig. 1: Basic test approach

First, a set of *Test Kernels*, each one implementing the test procedure for one of the internal SM components, has been defined.The CPU runs the *CUDA program* [9] to start the execution of the test kernels on the GPU. In order to properly

test each internal component, the GPU must execute one parallel instance of the test kernel for each internal module of each SM. From each of these instances it computes the related test results (TR) or test signatures (TS). Moreover, to identify the SM that has generated the test response, the GPU appends an SM identifier (SM ID) to each computed TS/TR. Finally, these information items are sent back to the CPU.
The CPU checks the presence of faults comparing the obtained TR/TS with the precomputed Golden TR (GR)/Golden TS (GS),. When a fault is detected, the CPU exploits the appended SM ID to localize the faulty SM, and creates the *Faulty SM Map* (FM) (i.e., a map indicating the faulty or fault-free status of SMs)
The proposed test approach has to deal with three main issues. The first concerns how ensuring the execution of the test procedure on each component of each SM. This issue has been solved by properly configuring the compiled test kernel in terms of *Threads*, *Thread Blocks* and *Grid* [9].
The second issue involves the methodology to prevent the compiler from inserting extra operations in the test kernel (i.e., extra operations could alter the fault detection capability of the test procedure). This issue can be solved writing each test kernel exploiting the inline-assembly provided by the CUDA-ISA [10], and unrolling every *for loop* required by the test procedure resorting to the compiler directive *#pragma unroll* [11]. The last issue concerns the definition of the test procedure for each internal component of a SM. In the proposed approach, some test procedures have been implemented exploiting well-known SBST methodology modified to tackle GPU peculiarities, while others have been implemented as fully custom test procedures. Additional implementation details can be found in [12].

III. Proposed Fault Mitigation strategy

To ensure correct results also in presence of faults, the proposed Fault Mitigation strategy prevents kernel executions on faulty SMs. It exploits the FM, obtained by executing the proposed SBSTD methodology (see Sec. II), to perform the fault mitigation.
In the proposed approach (see Fig. 2) the fault mitigation is ensured by instrumenting the CUDA program and the kernel.

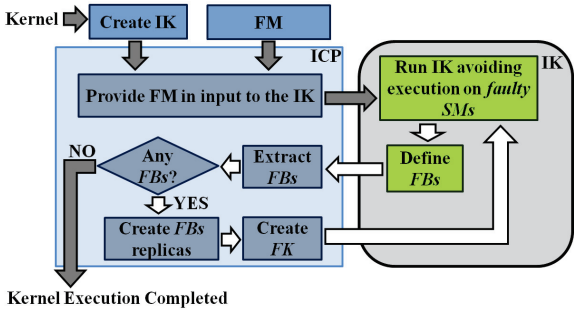

Fig. 2: Proposed Approach

First, the *Instrumented CUDA Program* (ICP), exploiting the FM, provides the kernel the information about the faulty SMs. Then, the instrumented kernel avoids the execution of the *Thread Blocks* on faulty SMs. Since there are no available instructions that ensure to directly operate on the *Thread Blocks* scheduling policy, an ad-hoc procedure has been implemented. The *Instrumented Kernel* (IK) identifies the *Thread Blocks* scheduled on faulty SMs (i.e., *Faulty Blocks* (FBs)), and stops their execution. At the end of the execution, the IK communicates to the ICP the FBs.
This information is exploited by the ICP to check whether all *Thread Blocks* have been executed on a faulty-free SM (i.e., execution completed). If the execution is not completed, the ICP creates several replicas of each FB. A new kernel (*Faulty Kernel* (FK)), containing only the generated replicas, will be created and executed on the CUDA GPU.
The presence of more than one copy of each FB ensures the execution of each FB on at least one fault-free SM. This set of steps (i.e., loop identified by the white arrows in Fig. 2) is iteratively repeated until no FBs are detected after the IK execution.
For additional information about the implementation details of the proposed strategy the reader may refer to [13].

IV. Conclusion

The paper presents an innovative approach to allow the use of a CUDA GPU, even in presence of faults. The presented approach is completely algorithm independent and it allows to reach the maximum performance during a fault-free execution.

References

[1] S. Potluri, A. Fasih, L. Vutukuru, F. Al Machot, and K. Kyamakya, "CNN based high performance computing for real time image processing on GPU," in *3rd International Workshop on Nonlinear Dynamics and Synchronization (INDS)*, pp. 1–7, 2011.

[2] J. Sheaffer, D. Luebke, and K. Skadron, "A hardware redundancy and recovery mechanism for reliable scientific computation on graphics processors," in *22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pp. 55–64, 2007.

[3] K. Haklin and H. Ho-sang, "Integrated Fault Tolerant System for Automotive Bus Networks," in *2nd International Computer Engineering and Applications Conference*, pp. 486–490, 2010.

[4] Q. Hu, B. Xiao, and M. Friswell, "Robust fault-tolerant control for spacecraft attitude stabilisation subject to input saturation," *Control Theory Applications*, vol. 5, no. 2, pp. 271–282, 2011.

[5] N. Z., "Investigation of Fault-Tolerant Adaptive Filtering for Noisy ECG Signals," in *IEEE Symposium on Computational Intelligence in Image and Signal Processing*, pp. 177–182, 2007.

[6] A. Strano, D. Bertozzi, A. Grasset, and S. Yehia, "Exploiting structural redundancy of SIMD accelerators for their built-in self-testing/diagnosis and reconfiguration," in *IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pp. 141–148, 2011.

[7] F. Sinclair, "G-cp: Providing fault tolerance on the GPU through software checkpointing," 2010.

[8] C. Ding, C. Karlsson, H. Liu, T. Davies, and Z. Chen, "Matrix multiplication on GPUs with on-line fault tolerance," in *9th Int'l Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pp. 311–317, 2011.

[9] nVidia, *nVidia's Next Generation CUDA Computer Architecture: Fermi*, 2006.

[10] nVidia, *Parallel Thread Execution ISA*, 2012.

[11] nVidia, *nVidia CUDA C Programming Guide v.4.2*, 2012.

[12] S. Di Carlo, G. Gambardella, M. Indaco, I. Martella, D. Rolfo, P. Prinetto, and P. Trotta, "A Software-Based Self Test of CUDA Fermi GPUs," in *18th European Test Symposium (ETS)*, 2013.

[13] S. Di Carlo, G. Gambardella, I. Martella, D. Rolfo, P. Prinetto, and P. Trotta, "Fault mitigation strategies for CUDA GPUs," in *International Test Conference (ITC)*, 2013.