The Case for Custom Parallel Memories: an Application-centric Analysis

Giulio Stramondo, Cătălin Bogdan Ciobanu, Ana Lucia Varbanescu
Informatics Institute, University of Amsterdam, The Netherlands,
{g.stramondo,c.b.ciobanu,a.l.varbanescu}@uva.nl

I. Introduction

The use of Field-Programmable Gate Arrays (FPGAs) in heterogenous high-performance compute architectures enables the exploration of alternative computing models. FPGAs can be tailored to accelerate applications with custom architectures aiming at reducing control overhead and power consumption with respect to Graphics Processing Units (GPUs) and Central Processing Units (CPUs). Effectively, compared with standard accelerators, FPGAs have the advantage of flexibility, enabling better application-centric designs.

One direction in which this flexibility can be invested is the memory system. In traditional accelerators, the memory system only supports limited parallelism by using distributed memories. However, it is the task of the application programmer and/or the compiler to (re-)group the memory accesses such that the bandwidth is maximized. In this paper, we argue that it is time to investigate the performance impact of smart parallel memory systems, able to adapt themselves to the memory access pattern of the application.

II. Background

A Polymorphic Register File (PRF) is a parameterizable register file, which can be logically reorganized by the programmer or the runtime system to support multiple register dimensions and sizes simultaneously [3], [4]. The PRF can be used as a parallel scratchpad, to store data structures that need high speed access. The PRF optimizes the memory throughput for a set of predefined memory access patterns. When implemented in reconfigurable technology, PRFs allow additional runtime customization of capacity, throughput, number of read/write ports, and supported access patterns.

For the PRF implementation, we consider $p \times q$ memory modules and five parallel access schemes [3]. Each scheme supports dense, conflict-free access to $p \cdot q$ elements arranged in one or more of the following access patterns: rows, columns, rectangles, diagonals, and secondary diagonals.

In this work, we employ the PRF as a parallel memory that supports a mix of the five access schemes mentioned above. Therefore, by analyzing the stream of accesses from the application to the memory system, and decomposing it into combinations of PRF patterns, we effectively enable parallel memory accesses.

III. Implementation and Preliminary Results

This paper focuses on the Design Space Exploration (DSE) for custom parallel memories based on the PRF idea. Specifically, we investigate how to determine the best organization of the parallel memory for a given application, using memory throughput and efficiency as optimization criteria.

For DSE we assume a PRF configured with M memories and a set of access schemes, S, and we explore the optimal sequence of PRF parallel accesses that covers the application memory accesses. For this exploration, we need two stages: generation and selection. More specifically, assuming a 2D region of memory, R, that covers all application access patterns, the access generator produces all possible PRF parallel accesses of M elements. The selection stage chooses, from all these accesses, the sequence of parallel accesses of length M that *optimally* cover the parallel access of the application. Covering a given parallel access with a set of smaller parallel accesses, like the ones obtained by the access generator, is a particular instance of the *set covering problem*. Therefore, we use an ILP solver to minimize the canonical formulation of such a problem [5]:

$$\text{minimize} \sum_{S \in C} x_s s.t. \sum_{S:e \in S} x_s \geq 1,\ \forall e \in U, x_s \in \{0,1\},\ \forall S \in C.$$

In our case, the universe U contains all the elements accessed by the application, the coverage C is the minimum set of parallel accesses that covers the universe, S is an access produced by the simulator, and x_s is a binary codification of S being part of the solution.

For evaluation, we used the access pattern of the kmeans algorithm (one of the 13 computational dwarfs[1]), as extracted from the Rodinia implementation of the algorithm[2]. As the original pattern was too regular for our purposes (row-based only), we have degenerated it into 4 new patterns: a "chess" pattern (stride 2 regular access), a "dense" pattern (read 2, skip 1), a "sparse" pattern (read 1, skip 2), and a "random" pattern (read 1, skip random). Our results demonstrate that: (1) there is significant speedup to be obtained using parallel memories for all these patterns, and (2) the trade-off between efficiency, speed-up, and size of the memory system can determine the best application-centric organization of a parallel memory system. We support these claims with Fig. 1 and 2 included in the Appendix [1].

[1]Due to space limitations, we cannot include a more in-depth analysis. More graphs and data are available at https://github.com/giuliostramondo/prf-simulator.

IV. Summary

In this work we investigate the idea of using the PRF [3] as a smart parallel memory system, to enable less-than-regular patterns be served by a parallel memory. Our focus is on the design space exploration for the organization of such an application-centric parallel memory, and our goal is to maximize speed-up and/or efficiency. Our preliminary results show interesting trade-offs between speed-up and efficiency, and ultimately indicate the best design points for this organization. We are currently working on a prototype implementation, using the Maxeler toolchain, on which we can validate the proposed designs. Results from this step will be included in the full version of the paper.

Appendix

Due to space limitations, we only include two figures to support our preliminary results in this section. Figure 1 shows the advantage of using a parallel memory over a sequential one. This graph shows the speedup obtained for all our test cases and different PRF access schemes. R stands for parallel row accesses, RD for a combination of row and diagonal accesses, and RE for rectangular accesses. The speedup metric used in this work is defined as:

$$Speedup = \frac{SequentialAccesses}{ParallelAccesses}$$

. Figure 2 shows how efficient is the usage of all the memory computed over the same test cases; we claim that this metric is correlated with the power consumption of the entire memory system. The efficiency metric used in this work is defined as:

$$Efficiency = \frac{SequentialAccesses}{ElementsAccessedinParallel}$$

. These results prove that parallel memory systems deliver performance even for sparse accesses, but choosing the right configuration for the memory system can have a high impact on efficiency.

References

[1] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, et al. The landscape of parallel computing research: A view from berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.

[2] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 44–54. IEEE, 2009.

[3] C. Ciobanu. *Customizable Register Files for Multidimensional SIMD Architectures*. PhD thesis, Delft University of Technology, Delft, Netherlands, March 2013.

[4] C. Ciobanu, G. K. Kuzmanov, A. Ramirez, and G. N. Gaydadjiev. A Polymorphic Register File for Matrix Operations. In *Proceedings of SAMOS*, pages 241–249, July 2010.

[5] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

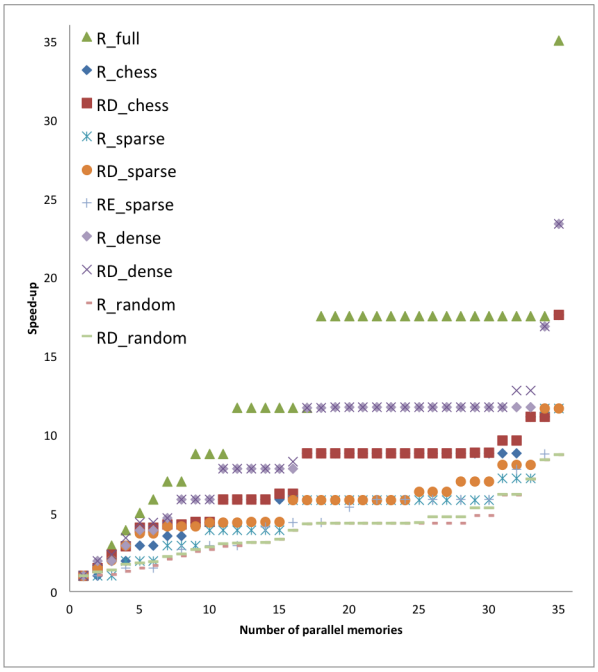

Fig. 1. Speed-up of the parallel memory system for different access patterns. Note that the number of accesses differs for the different access patterns: ”full” has the most accesses, while ”random” has the least. Also note that the R-, RD-, RE- refer to the PRF schemes.

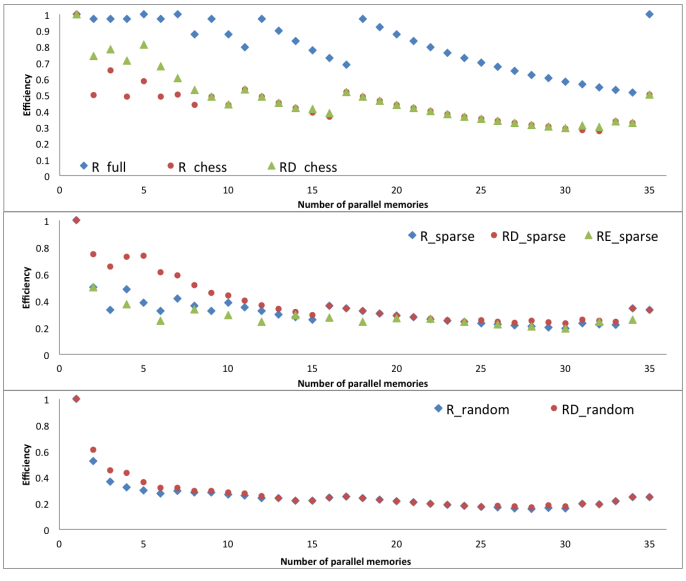

Fig. 2. Efficiency of the parallel memory system for different access patterns. note that the R-, RD-, RE- refer to the PRF schemes.