A Fine-Grained Parallel Snappy Decompressor for FPGAs Using a Relaxed Execution Model

Fang, Jian; Chen, Jianyu ; Lee, Jinho; Al-Ars, Zaid; Hofstee, Peter

DOI
10.1109/FCCM.2019.00076

Publication date
2019

Document Version
Final published version

Published in
2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)

A Fine-grained Parallel Snappy Decompressor for FPGAs Using a Relaxed Execution Model

Jian Fang*, Jianyu Chen*, Jinho Lee†, Zaid Al-Ars* and H. Peter Hofstee*†
{j.fang-1, z.al-ars, h.p.hofstee}@tudelft.nl, j.chen-13@student.tudelft.nl, leejinho@us.ibm.com
*Delft University of Technology, Delft, the Netherlands
†IBM Research, Austin, Texas, USA

I. Introduction

Snappy [1] is a widely used (de)compression algorithm in many big data applications. Such a data compression technique has been proven to be successful to save storage space and to reduce the amount of data transmission from/to storage devices. In this paper, we present a fine-grained parallel Snappy decompressor on FPGAs running under a relaxed execution model that addresses the following main challenges in existing solutions. First, existing designs either can only process one token per cycle or can process multiple tokens per cycle with low area efficiency and/or low clock frequency [2], [3]. Second, the high read-after-write data dependency during decompression introduces stalls which pull down the throughput. This paper improves on our previous work [4] and integrate it in a CAPI 2.0-enabled system.

II. Architecture Highlight

The proposed method assumes all the bytes of an input stream line (16B) is a starting byte of a token and parses them in parallel. The correct branch is selected afterward to locate the boundary of tokens. A bit map structure is utilized for this correct branch selection to reduce resource usage and to keep a high clock frequency.

The proposed method splits the tokens into single-BRAM read/write commands that can independently operate on different BRAMs, which allows executing multiple tokens in parallel without duplicating the history when a BRAM bank conflict occurs.

The proposed methods utilize a relaxed execution model, a similar method as in [5], that executes all the BRAM commands immediately and recycles those with invalid data to reduce the penalty caused by the read-after-write data dependencies.

III. Experiments

We evaluate the proposed methods in the Xilinx Virtex Ultrascale VU3P-2 device on an AlphaData ADM-PCIE-9V3 board integrated with the POWER9 CAPI 2.0 [6] interface. The CAPI 2.0 interface on this card supports the CAPI protocol at an effective data rate of approximately 11GB/s.

Table I lists the resource utilization of a single engine design timing at 250MHz. The decompressor only takes around 14% of the LUTs and 6% of the BRAMs, while the CAPI 2.0 interface logic implementation takes up around 21% of the LUTs and 26% of the BRAMs. Multi-unit designs can share the CAPI 2.0 interface logic between all the decompressors, and thus can support up to 5 engines in such device.

TABLE I
Resource utilization of design components

Resource	LUTs	BRAMs	Flip-Flops
One engine	56K(14.2%)	42(5.8%)	37K(4.7%)
CAPI2 interface	82K(20.8%)	187(26%)	119K(10%)
Total	138K(35%)	229(31.8%)	116K(14.7)

TABLE II
Benchmarks used and throughout results

Files:	Original size (MB)	Compression ratio	CPU	FPGA
Matrix	771.3	2.75	0.74 GB/s	4.8 GB/s
Wiki	953.7	1.97	0.53 GB/s	5.7 GB/s
TPC-H	54.1	3.86	1.05 GB/s	2.2 GB/s

The design performance was evaluated on a set of compressed files (Table II) including matrix data (Matrix), text (Wiki), and table (TPC-H). The FPGA design was compared with a software implementation running on the POWER9 CPU. The results show that the throughput of the proposed method reaches upto 5.7GB/s which is around an order of magnitude larger than a single-threaded implementation on the POWER9 processor. Two of these decompressor engines, operating on independent streams, can saturate a PCIe Gen4 or CAPI 2.0 x8 interface, utilizing less than a third of the available resources, and the design is efficient enough to easily support data rates for an OpenCAPI 3.0 x8 interface.

References

[1] Google, "Snappy," https://github.com/google/snappy/, Accessed: 2018-12-01.
[2] Y. Qiao, "An FPGA-based Snappy Decompressor-Filter," Master's thesis, Delft University of Technology, 2018.
[3] K. B. Agarwal, H. P. Hofstee, D. A. Jamsek, and A. K. Martin, "High bandwidth decompression of variable length encoded data streams," Sep. 2 2014, uS Patent 8,824,569.
[4] J. Fang, J. Chen, Z. Al-Ars, P. Hofstee, and J. Hidders, "A high-bandwidth Snappy decompressor in reconfigurable logic: work-in-progress," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*. IEEE Press, 2018, pp. 16:1–16:2.
[5] E. Sitaridi, R. Mueller, T. Kaldewey, G. Lohman, and K. A. Ross, "Massively-parallel lossless data decompression," in *Proc. of the International Conference on Parallel Processing*. IEEE, 2016, pp. 242–247.
[6] J. Stuecheli, "A new standard for high performance memory, acceleration and networks," http://opencapi.org/2017/04/opencapi-new-standard-high-performance-memory-acceleration-networks/, accessed: 2018-06-03.

2576-2621/19/$31.00 ©2019 IEEE
DOI 10.1109/FCCM.2019.00076