Towards Improving Data Transfer Efficiency for Accelerators using Hardware Compression

Max Plauth and Andreas Polze
Operating Systems and Middleware Group
Hasso Plattner Institute for Digital Engineering
University of Potsdam, Germany
{firstname.lastname}@hpi.uni-potsdam.de

Abstract—The overhead of moving data is the major limiting factor in today's hardware, especially in heterogeneous systems where data needs to be transferred frequently between host and accelerator memory. With the increasing availability of hardware-based compression facilities in modern computer architectures, this paper investigates the potential of hardware-accelerated I/O Link Compression as a promising approach to reduce data volumes and transfer time, thus improving the overall efficiency of accelerators in heterogeneous systems. Our considerations are focused on On-the-Fly compression in both Single-Node and Scale-Out deployments. Based on a theoretical analysis, this paper demonstrates the feasibility of hardware-accelerated On-the-Fly I/O Link Compression for many workloads in a Scale-Out scenario, and for some even in a Single-Node scenario. These findings are confirmed in a preliminary evaluation using software- and hardware-based implementations of the 842 compression algorithm.

Index Terms—Data compression, hardware, data transfer, accelerator architectures

I. INTRODUCTION

With the increasing prevalence of big data applications, data volumes are growing by the day and compute tasks are gaining complexity. Even though hardware accelerators such as Graphics Processing Units (GPUs) or Field-Programmable Gate Arrays (FPGAs) are vital for satisfying these demands, moving data forth and back between main memory and accelerators remains a major limiting factor, which is reflected by the many consortia working on faster interconnection technologies (e.g. OpenCAPI [1], CCIX [2], or Gen-Z [3]).

Even though several strategies exist for mitigating the performance impact of memory transfers (e.g. by overlapping computation and data transfers), these mitigation strategies are only applicable to workloads that can be pipelined. Preceding efforts of the research community have identified compression as an viable orthogonal method for improving data transfer efficiency for many application domains [4]. Taking advantage of the trend that hardware-accelerated compression is becoming increasingly available in many computer architectures [5][6], this paper re-evaluates the benefits of using compression to reduce data volumes and thus data transfer time.

While preceding work dealing with I/O Link Compression has brought forward many implementation strategies for efficient decompression facilities on the side of GPU-based accelerators, this paper complements these approaches by leveraging hardware-accelerated compression on the host side to provide further performance improvements across many workloads. Our investigations are based on commercial off-the-shelf hardware, more specifically the nx842 on-chip compression accelerator available in recent IBM POWER processors.

In this paper, we do not only consider the Single-Node scenario and thus node-local transfers (e.g. via PCI Express), but we also investigate the Scale-Out scenario which involves inter-node communication among compute nodes in a cluster (e.g. via 10GbE). Furthermore, we also distinguish between Offline I/O Link Compression and On-the-Fly I/O Link Compression. Demonstrating the feasibility of On-the-Fly I/O Link Compression for most workloads in the Scale-Out scenario, and even for some in the Single-Node scenario, the main contribution of this paper is to foreshadow the potential of using hardware-accelerated compression facilities situated near intra- as well as inter-node fabrics in order to further improve the efficiency of accelerators in heterogeneous systems.

This paper is structured as follows: Section II provides background about the employed 842 compression algorithm. Subsequently, Section III reviews related work in the field of compression techniques for accelerators. Theoretical considerations such as the requirements imposed by On-the-Fly I/O Link Compression for Single-Node and Scale-Out deployment are clarified in Section IV. All relevant details of the experimental setup are documented in Section V after which the experimental results are evaluated in Section VI. Finally, a conclusion is reached in Section VII.

II. BACKGROUND: 842 COMPRESSION

The main use case for which IBM has integrated the nx842 on-chip compression accelerator in their POWER processors is the Active Memory Expansion (AME) feature available in the AIX operating system, which offers transparent memory compression. Hence, the main design goal of the 842 algorithm [7] is to allow high-throughput/low-latency hardware implementations that can be placed directly on transmission channels [8]. These properties do align perfectly with our intent of evaluating the feasibility of hardware-accelerated On-the-Fly I/O Link Compression. Hereinafter, the basic procedure of the 842 algorithm is outlined.
As illustrated in Figure 1, the 842 algorithm [7] implemented on the nx842 on-chip accelerator operates on units of 8 bytes, treating the input data as sub-phrases of 8, 4, and 2 bytes length, respectively. For each phrase length, a hash function and a hash table with offsets to a sliding window buffer of past encoded data are used to detect possible matches of the sub-phrase therein. Based on the lookup, a template is chosen that encodes 8 bytes of raw data. Each 5-bit template encodes a permutation of offsets or literals of 8, 4, and 2 bytes length, followed the actual offsets and literals. With a clock frequency of 2.3GHz, and the ability to ingest 8 bytes per cycle, one nx842 on-chip accelerator can achieve a maximum throughput of roughly 18 GB/sec [5]. Being equipped with two nx842 on-chip accelerators [9], the total compression throughput of a POWER8 processor can be as high as 36 GB/sec.

The 842 algorithm can be attributed to the family of Lempel-Ziv derivatives [7]. The compression process deviates from the original Lempel-Ziv algorithm [10] in several aspects. However, decomposition works almost identical compared to LZ’77 [7]. With a sufficient number of approaches available that have demonstrated efficient decomposition of Lempel-Ziv derivatives on GPUs [11, 12, 13, 14, 15], we assume that decompression of 842-encoded data can be implemented efficiently on GPUs. Regarding FPGAs, an efficient implementation of 842 for both compression and decompression has been demonstrated as well [8], providing further indication that efficient implementations of the algorithm are feasible for most popular accelerator types. More detailed descriptions of the algorithm are available in [7, 8].

III. RELATED WORK

Using compression as means for improving utilization of main memory has a well-established history that goes back many years. This fact is well reflected by the work of Mittal et al. [16], which provides a comprehensive survey of the widely researched field of data compression mechanisms for main memory and cache systems. Narrowing down the field of approaches to GPU- or FPGA-based compression techniques, the existing approaches can be broken down into four major categories:

1) Compression Offloading: Compression implemented on the accelerator is used to speed up compression tasks on the host system.

2) Accelerator-sided Memory Compression: Analog to main memory compression, compression is used on the accelerator to improve utilization of resources such as registers or device memory.

3) I/O Link Compression: Compression is applied to improve the overall utilization of the I/O link connecting the accelerator and the host system. This category can be subdivided into two further categories:

   a) Offline I/O Link Compression: Data compressed at an earlier point in time is transferred to the accelerator in compressed form, where it is being decompressed efficiently (send-then-decompress).

   b) On-the-Fly I/O Link Compression: Raw data is being compressed ad-hoc when data transfers to the accelerator are issued (compress-send-decompress). In addition to accelerator-based decompression, this approach requires high performance compression mechanisms on the host side to be feasible.

A. Compression Offloading

In the space of GPU-based approaches, several attempts have been made to port well-established compression algorithms to graphics hardware in order to free up host resources by offloading compression operations. Popular approaches include CULZSS [11] and GLZSS [12], with CULZSS achieving up to 3× speed-up compared to parallel, Central Processing Unit (CPU)-based implementations and GLZSS yielding 2× speed-up over CULZSS. Both approaches employ the LZSS algorithm [17], which is also a derivative of the original Lempel-Ziv [10] algorithm.

Regarding FPGA-based approaches, Spoorthi and Udayashekaret [8] have implemented the 842 algorithm on an FPGA using VHDL. With nx842-equipped POWER CPUs being just announced at the time of writing, the authors did not have access to the nx842 compression accelerator. Stepping in for the not-yet-available nx842, their work aims at using the accelerator-based implementation to speed-up
B. Accelerator-sided Memory Compression

Targeting memory-bound applications, Sathish et al. [18] have proposed using hardware-based compression to increase efficiency of access to off-chip device memory, yielding up to 37% better performance compared to the uncompressed case. A similar, more recent approach has been published by Vijaykumar et al. [19], who are also employing memory and register compression to increase the utilization of all GPU resources, yielding up to 2.6× speed-up across a variety of memory-bound applications. Focusing entirely on the register level, Lee et al. [20] have explored register compression with the goal of reducing energy consumption of graphics hardware. All approaches have in common that they are using custom compression algorithms instead of general purpose algorithms.

C. Offline I/O Link Compression

Targeting efficient decompression of LZ77 on GPUs, Gompa [15] achieves a 2× speedup in decompression performance compared to a multithreaded, CPU-based implementation. Funasaka et al. have demonstrated multiple approaches for efficient decompression facilities on GPUs with the goal of increasing the data transfer efficiency either from host main memory or from non-volatile storage. The approaches are using different compression algorithms, including industry-standard algorithms such as LZW [13, 14] as well as custom algorithms such as the Light Loss-Less Data Compression (LLL) [21] and Adaptive Lossless Data Compression (ALL) [4] approaches. Especially the custom algorithms LLL and ALL optimized for efficient decompression on GPUs can outperform CULZSS [11] and their GPU-based LZW implementation [13] [14] significantly.

D. On-the-Fly I/O Link Compression

Aiming at high compression rates with space savings below 0.5, Patel et al. [22] have explored the feasibility of On-the-Fly compression for data transfers between host and GPU. The authors conclude that On-the-Fly compression is not feasible using their software-based compression approach implemented on the host CPU.

IV. THEORETICAL CONSIDERATIONS

This paper is focused on the On-the-Fly mode of use, covering both Single-Node and Scale-Out deployment scenarios. The Offline scenario is largely disregarded in this work, as it merely requires hardware compression facilities to outperform software-based compression regardless of the deployment model. Hence, this section particularizes the properties of both the Single-Node and the Scale-Out deployment model and the requirements imposed by On-the-Fly compression for each deployment scenario.
B. Scale-Out Scenario

The Scale-Out scenario addresses workloads exceeding the compute capacity of a single node. The majority of small to mid-ranged scale-out clusters is restricted to comparatively slow commodity interconnection technologies such as Ethernet, especially when dynamic on-demand cluster setups are formed using cloud computing infrastructures (e.g. using dOpenCL [23] or CloudCL [24, 25]). Therefore, the Scale-Out scenario illustrated in Figure 4 evaluates the use of hardware-based compression in distributed environments in order to mitigate the low bandwidth of the inter-node network.

![Diagram](image)

Fig. 4: In the Scale-Out scenario, compression may increase the effective bandwidth of inter-node networks such as 10GbE.

Even though 10 Gigabit Ethernet (10GbE) is widely available in many on-premise clusters, slower network speeds are commonly found in cloud computing infrastructures. Using Amazon EC2 instances for example, 10, 5, 2.5 or even 1 Gbps of sustained network bandwidth are available in many instance types [26]. As a result, our investigation assumes the aforementioned link speeds among cluster nodes. Figure 5 explicates the relationship between space savings, network bandwidth and the compression throughput, with filled surfaces indicating the space savings and compression throughput required for a given network speed that needs to be met in order to make On-the-Fly compression feasible.

![Chart](image)

Fig. 5: Filled areas indicate feasibility of On-the-Fly compression for the available network bandwidth. The white line indicates theoretical compression throughput per CPU socket.

V. EXPERIMENTAL SETUP

The considerations elaborated in Section IV show that with link speed being a constant determined by the deployment scenario, compression throughput and the space savings for a given workload are the decisive variables deciding whether On-the-Fly compression is feasible for the respective scenario or not. Hereinafter, this section elucidates the experimental procedures used to obtain practical values for both variables.

A. Space Savings

In order to ascertain the space savings achieved by the 842 algorithm for a wide set of versatile workloads, we compiled a well-defined suite of test data. The test data suite is comprised of the Large Text Compression Benchmark [27], the Silesia Corpus [28], the first chromosome of the GRCh38.p12 Human Reference Genome [29], two grayscale images illustrated in Figure 6 (4240 x 2832 pixels, TIFF file), as well as a rasterized image of Figure 1 (3206 x 910 pixels, TIFF file). The space savings yielded by the 842 algorithm for the test data set are reported in Figure 7.

![Image](image)

(a) Castle
(b) Snow Monkey

Fig. 6: To evaluate the space savings achieved by the 842 algorithm for image data, the two exemplary grayscale images illustrated in (a) and (b) are included in the test data suite.

![Chart](image)

Fig. 7: Space savings achieved by the 842 algorithm for the test data suite. Solid lines indicate feasibility for On-the-Fly compression in the deployment scenarios, assuming the theoretical 36 GB/sec compression throughput of a socket.
TABLE I: Specifications of bare-metal test environment.

<table>
<thead>
<tr>
<th>CPU</th>
<th>2 × IBM POWER8, 10 Cores / 80 Threads, 3.42 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>16 × 64GB DDR3 ECC CDIMM, 1600 MHz</td>
</tr>
<tr>
<td>GPU</td>
<td>1x NVIDIA Tesla K80</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu 16.04.4 64 Bit</td>
</tr>
</tbody>
</table>

TABLE II: Specifications of virtualized test environment.

<table>
<thead>
<tr>
<th>CPU</th>
<th>20 Cores / 160 Threads, 4.024 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>1TB DDR3 ECC memory, 1600 MHz</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu 18.04 64 Bit / AIX 7.2</td>
</tr>
</tbody>
</table>

B. Compression Throughput

With the values elaborated in Section IV being purely theoretical, we also conduct a practical evaluation to determine effective compression throughput. With this work focusing on hardware-based compression, the prime target of this evaluation is the nx842 compression accelerator. However, to put the performance of a hardware-based approach in perspective with software-based approaches, our evaluation also includes software-based implementations of the 842 algorithm.

At the time of writing, user space access to the nx842 accelerator on Linux is only possible using the Virtual Accelerator Switch facilities introduced with POWER9 CPUs. With the test environment at hands being based on POWER8 CPUs, throughput measurements under Linux have been performed using the comp_selftest kernel module [30] (HW/K: Linux). With AIX providing a straightforward user-space API to the compression facilities, additional benchmarks have been performed using an AIX 7.2 LPAR (HW/U: AIX).

On the side of software-based implementations of the 842 algorithm, the naïve software implementation available in the Linux kernel has been used as a baseline (SW/K: Linux). Additionally, the kernel-based implementation has been extracted as a user space application (SW/U: Naïve), serving as the foundation for an optimized software implementation (SW/U: Optimized).

Except for the AIX-based approach, all measurements are performed on the bare-metal system specified in Table I. The AIX-based measurements are conducted on the virtualized test system documented in Table II. All test procedures and the software-based implementations are freely available on GitHub [31].

VI. RESULTS & DISCUSSION

This section presents practical measurements of space savings and compression throughput as suggested in Section V. The results of the compression throughput measurements presented in Figure 8 demonstrate up to 16.83 GB/sec of compression throughput for the hardware-accelerated configuration (HW/K: Linux), lacking far behind the 36 GB/sec that the two nx842 accelerators available per socket could achieve in theory. A potential reason for the comparatively low compression throughput is that the support for compression accelerators in the Linux Crypto API has recently experienced severe modifications, thus leaving the drivers for the nx842 accelerators in a subpar state. For the AIX-based tests, the low performance may be a result of the nx842 accelerators in POWER8 being accessed using the privileged `initiate coprocessor store word indexed` (icswx) instruction, which requires additional context switches to the kernel and hypervisor space, thus causing additional overhead. Access to the nx842 accelerators has been simplified with the introduction of the Virtual Accelerator Switchboard (VAS) facilities in the POWER9 CPU series [32]. Hence, it remains to be seen whether VAS can achieve throughput closer to the theoretical performance of the nx842.

On the side of the software-based approaches, the optimized implementation (SW/U: Optimized) was able to achieve up to 2 GB/s using 20 threads. As expected, software-based approaches perform much slower than the hardware-accelerated configuration. However, the software-based approaches still offer many opportunities for further optimization.

Evaluating the feasibility of the Single-Node scenario, we were not able to artificially reduce the number of active PCIe lanes to measure the sustained throughput in cases where only 8 or 4 lanes are available per accelerator. Hence, we performed a worst-case estimation, comparing the measured transfer time of uncompressed data with transfers using On-the-Fly compression, assuming the ideal link speed of PCI Express and the actual compression throughput achieved in our benchmarks. At first glance, the estimated values denoted in Table III draw a gloomy picture when using 16 lanes. However, it should be noted that already for 8 lanes, On-the-Fly compression starts to become feasible for a growing number of workloads. Using 4 lanes, almost all workloads can gain performance from compressed data transfers. Keeping in mind the fact that the VAS facilities in POWER9 might result in improved compression throughput, even more workloads might be suited for On-the-Fly compression. Furthermore, many mid-range FPGAs and other accelerators are equipped with PCIe x4 connectivity. Hence, the full potential of On-the-Fly compression might be developed in accelerator classes other than GPUs.
TABLE III: Estimation of time (ms) required for raw data transfers compared to transfers compressed On-the-Fly in the Single-Node scenario. Values are based on compression throughput (HW/K: Linux) and theoretical PCIe link speeds.

<table>
<thead>
<tr>
<th></th>
<th>x16 raw compr.</th>
<th>x16 x compr.</th>
<th>x8 raw compr.</th>
<th>x8 x compr.</th>
<th>x4 raw compr.</th>
<th>x4 x compr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>enwik8</td>
<td>6.25</td>
<td>10.05</td>
<td>12.50</td>
<td>14.43</td>
<td>25.00</td>
<td>23.19</td>
</tr>
<tr>
<td>enwik9</td>
<td>62.50</td>
<td>100.41</td>
<td>125.00</td>
<td>144.15</td>
<td>250.00</td>
<td>231.64</td>
</tr>
<tr>
<td>dickens</td>
<td>6.64</td>
<td>10.02</td>
<td>12.27</td>
<td>14.37</td>
<td>25.55</td>
<td>23.36</td>
</tr>
<tr>
<td>mozilla</td>
<td>3.70</td>
<td>4.83</td>
<td>6.40</td>
<td>6.76</td>
<td>12.81</td>
<td>10.61</td>
</tr>
<tr>
<td>mr</td>
<td>0.62</td>
<td>0.86</td>
<td>1.25</td>
<td>1.16</td>
<td>2.49</td>
<td>1.75</td>
</tr>
<tr>
<td>nci</td>
<td>2.10</td>
<td>2.66</td>
<td>4.19</td>
<td>3.42</td>
<td>8.39</td>
<td>4.93</td>
</tr>
<tr>
<td>ooffice</td>
<td>0.38</td>
<td>0.64</td>
<td>0.77</td>
<td>0.93</td>
<td>1.54</td>
<td>1.51</td>
</tr>
<tr>
<td>osdb</td>
<td>0.63</td>
<td>1.14</td>
<td>1.26</td>
<td>1.72</td>
<td>2.52</td>
<td>2.87</td>
</tr>
<tr>
<td>reymont</td>
<td>0.41</td>
<td>0.62</td>
<td>0.83</td>
<td>0.86</td>
<td>1.66</td>
<td>1.34</td>
</tr>
<tr>
<td>samba</td>
<td>1.35</td>
<td>2.07</td>
<td>2.70</td>
<td>2.91</td>
<td>5.40</td>
<td>4.60</td>
</tr>
<tr>
<td>sao</td>
<td>0.45</td>
<td>0.78</td>
<td>0.91</td>
<td>1.15</td>
<td>1.81</td>
<td>1.90</td>
</tr>
<tr>
<td>webster</td>
<td>259.03</td>
<td>4.03</td>
<td>5.18</td>
<td>5.71</td>
<td>10.36</td>
<td>9.06</td>
</tr>
<tr>
<td>xml</td>
<td>0.33</td>
<td>0.60</td>
<td>0.67</td>
<td>0.89</td>
<td>1.34</td>
<td>1.48</td>
</tr>
<tr>
<td>x-ray</td>
<td>0.55</td>
<td>0.74</td>
<td>1.06</td>
<td>1.01</td>
<td>2.12</td>
<td>1.53</td>
</tr>
<tr>
<td>castle</td>
<td>3.25</td>
<td>3.53</td>
<td>4.51</td>
<td>5.01</td>
<td>9.01</td>
<td>7.97</td>
</tr>
<tr>
<td>monkey</td>
<td>2.75</td>
<td>3.48</td>
<td>4.50</td>
<td>4.92</td>
<td>9.00</td>
<td>7.79</td>
</tr>
<tr>
<td>drawing</td>
<td>0.73</td>
<td>0.71</td>
<td>1.46</td>
<td>0.75</td>
<td>2.92</td>
<td>0.84</td>
</tr>
<tr>
<td>hg38.chr1</td>
<td>15.87</td>
<td>21.56</td>
<td>31.74</td>
<td>28.73</td>
<td>63.48</td>
<td>43.07</td>
</tr>
</tbody>
</table>

To evaluate the feasibility of the Scale-Out scenario, we measured network performance between two nodes with their link rate limited to 10 Gbps, 5 Gbps, and 1 Gbps respectively. We performed an estimation, comparing the transfer time of uncompressed data with transfers using On-the-Fly compression, assuming the measured network link speeds. The transfer times reported in Table IV acknowledge consistent feasibility of On-the-Fly compression across all workloads. At 5 Gbit/s link speed, even software-based On-the-Fly compression remains feasible for many workloads (data not shown).

Even though large parts of this paper are focused on On-the-Fly compression, the compression throughput of the hardware-accelerated configuration demonstrated in Figure 8 exceeds the performance of software-based approaches by far. Hence, it is safe to assume that the hardware-based approach can provide considerable performance improvements in the pre-compressing stages of an Offline I/O Link Compression workflow. Using hardware-based compression, increased data volumes can be processed, all while freeing up CPU resources.

VII. CONCLUSION

Exploiting the trend that hardware-accelerated compression is becoming increasingly available in many computer architectures [50], this paper re-evaluated the feasibility of using high-throughput compression facilities to further improve the efficiency of accelerators in heterogeneous systems. In addition to investigating On-the-Fly I/O Link Compression in Single-Node scenarios where data needs to be transferred locally via PCI Express, this work also considered the Scale-Out scenario which involves inter-node communication among compute nodes in a cluster (e.g. 10GbE).

We have demonstrated the feasibility of hardware-accelerated compression for a wide selection of workloads in the Scale-Out scenario, and in environments using 5 Gbit/s network link speed or slower even when using software-based compression. In the Single-Node scenario, this paper fore-shadowed the potential of hardware-accelerated compression in heterogeneous systems, having displayed that On-the-Fly compression can be feasible on intra-node fabrics such PCI Express in configurations where only 8 or 4 lanes are available to attach an accelerator to the host system. This applies to systems equipped with many accelerators, where not enough lanes are available to attach all accelerators to the host CPUs at the full link width (e.g. NVIDIA DGX), or when mid-ranged FPGAs or special purpose accelerators are used.

To study the feasibility of On-the-Fly I/O Link Compression in practice, the next logical steps include completing the test setup elaborated in this paper with accelerator-based decompression facilities to cover the full compress-send-decompress workflow. Furthermore, to increase the expressiveness of our investigation, additional experiments need to be conducted using benchmarks from the heterogeneous computing domain instead of benchmarks from the data compression domain.

As a first step towards improving data transfer efficiency for accelerators using hardware-accelerated compression facilities, the main contribution of this paper is that we have demonstrated the feasibility of hardware-accelerated I/O Link Compression for most workloads in the Scale-Out scenario, and for some workloads even in the Single-Node scenario. Based on the considerations and measurements presented in this paper, our future research efforts will be directed towards evaluating On-the-Fly compression in practice on both GPU and FPGA based accelerators.

ACKNOWLEDGEMENT

We would like to thank Bulent Abali, who helped us understand all the details of the 842 compression algorithm.
REFERENCES