Efficiency Considerations of Cauchy Reed-Solomon Implementations on Accelerator and Multi-Core Platforms

Thomas Steinke, Kathrin Peter, and Sebastian Borchert
Zuse Institute Berlin
Takustr. 7, D-14195 Berlin-Dahlem, Germany
Email: {steinke, kathrin.peter, borchert}@zib.de

***Abstract*—The Cauchy variant of the Reed-Solomon algorithm is implemented on accelerator platforms including GPGPU, FPGA, CellBE and ClearSpeed as well as on a x86 multi-core system. The sustained throughput performance and kernel rates are measured for a 5+3 Reed-Solomon schema. To compare the different technology platforms an efficiency is introduced and the platforms are categorized according to their Reed-Solomon efficiency.**

I. Introduction

Storage subsystems in petascale compute facilities will include a very large number of disks [1]. In such large-scale deployments parity RAID schema can deliver a MTTDL of a few days only which is clearly not acceptable for production systems. Reed-Solomon (RS) codes are applied successfully to build fault-tolerant storage systems and ensuring reliability. In contrast to traditional RAID configurations the RS algorithm has advantages due to its flexibility in terms of the level of acceptable fault-tolerance, and its efficiency regarding storage resource utilization. Since RS encoding/decoding adds substantial overhead in the data processing pipeline ASIC implementations were used in the past.

Recent algorithmic and technological developments enable a transition from the ASIC to the COTS space. Using commonly available processing devices for RS en-/decoding an alternative approach towards fault-tolerance in large parallel applications is feasible.

In this study we demonstrate the potential of multi-core and application accelerator platforms for RS encoding. Platforms include GPGPUs, CellBE, FPGA and ClearSpeed. To evaluate the quality of our implementations across these varying platforms we define a simple metric to measure the efficiency of our RS encoding.

Due to space limitations discussion of power efficiency are not considered in this paper.

II. The Cauchy Reed-Solomon algorithm

Reed-Solomon (RS) codes are non-binary cyclic block codes [2]. Compared to other codes the Reed-Solomon codes have the following advantages:
(1) Its maximum distance separable (MDS) property guarantees an optimal ratio of check words to data words in terms of tolerable failures. Using a $k + m$ RS schema with k data words and m check words the system can tolerate up to m simultaneous failures.
(2) RS codes are flexibel, i. e. the coding schema can be adjusted to different fault-tolerance requirements by calculating an arbitrary number of check words for a given number of data words.
RS encoding is a matrix vector multiplication with the help of a special generator matrix built on Vandermonde matrices. In the standard RS algorithm [3] most of the time is consumed by Galois-field operations over finite fields. In the Cauchy Reed-Solomon coding [4] the Galois-field operations over $GF(2)$ translates to bit-selection and subsequent XOR operations which are fast on all CPU architectures. Moreover, one can optimize the Cauchy Reed-Solomon by minimizing the number of XOR operations [5].

III. Cauchy Reed-Solomon implementations

A. Platforms and implementation models

The accelerator classes and hardware platforms used in this study include:
– x86: Intel X5570 (Nehalem),
– GPGPU: NVIDIA Tesla C870 and C1060,
– CellBE: IBM PowerXCell8i.
– FPGA: SGI RC100 / Altix 450 and
– SIMD: ClearSpeed CSX e620.
On all platforms a 5+3 RS schema was implemented meaning that three check data blocks are generated out of five input data blocks. A data parallel implementation schema is applied for the RS encoding by splitting the original user data set into data blocks which are processed independently.

All hardware accelerator platforms in this study (except CellBE) implement a co-processor model, i. e. the individual hardware accelerator device is attached to a host system and has a separate address space. For the RS encoding the user data residing in host memory needs to be transferred to memory locally attached to the accelerator device, and consequently, the computed check data are transferred back to the host memory. In the co-processor model an overlap of computation and communication is critical to achieve a high RS throughput. In general, multi-buffering or a streaming model are proper recipes to enable the desired overlap.

B. Platform specific optimizations

The optimization efforts were different for each of the platforms – less time consuming on x86 and more expensive on any of the accelerator platforms. Starting from a common parallelization model the platform-specific optimization focus on optimal blocking and DMA schemas and the manual tuning of their parameters.

a) x86 multi-core CPU implementation: The x86 code is based on the published implementation by Plank [6]. The XOR operations were completely unrolled. For the SSE support the internal data layout and alignment was adjusted and prefetch functions were applied. The parallelization was achieved by means of OpenMP. Memory affinity was ensured using the NUMA API and the thread placement was realized via `KMP_AFFINITY=granularity=thread,scatter` process environment supported by the Intel compiler.

b) GPGPU implementation: Two communication schema were implemented, the input data are i) copied synchronously block-wise from host to GPU global memory, or ii) transferred using streams in asynchronous mode. The GPU kernel is called as a two-dimensional grid of thread blocks where each of the thread block is organized as an one-dimensional thread pool. The kernel reads two 4-byte integers per input data stripe and the kernel operations are implemented to work on `short` data types.

c) PowerXCell 8i implementation: On PowerXCell8i the input data are blocked at size of 512 bytes per stripe. Double buffering is used for transferring the data to/from the 8 SPUs. In the inner kernel loop XOR's work on 128-bit wide registers. Due to observed start-up overheads, timing results are measured for the second and subsequent benchmark runs.

d) FPGA implementation: The Mitrion-C compiler [7] generates a 100 MHz hardware design which halves the maximum NUMAlink4 bandwidth to/from RC100 blade to 3.2 GBytes/s. The XOR tree was generated semi-automatically. With a fixed RS encoding matrix the compiler is enabled to optimize the XOR tree with the known constants. The read/write operations to FPGA attached SRAM are organized in 128-bit wide data blocks and therefore the internal RS pipelines operate on 128-bit words. The FPGA design consumes 14% of the flip-flops, 23% of the slices and 3% of the block RAM available on the Xilinx XCV4LX200. Synthesis, place and route required 163 min on a standard server. The double buffering feature supported by Mitrionics SDK in conjunction with the SGI RASClib is used for overlapping communication with RS encoding.

e) ClearSpeed implementation: Double buffering is used for the data transfer both between host and mono memory, and between mono and poly memory, respectively. For the later setup, two I/O threads are used on the mono unit to control the data stream between mono and poly memory. Five input data stripes are transferred from the host into the mono memory, and for the RS processing five blocks of 256 bytes each were moved from mono to poly SRAM memory.

IV. Results: Cauchy Reed-Solomon rates

Our sustained Cauchy Reed-Solomon throughput (RS rate) is given by the size of the input data set resident in the host memory divided by the time to provide the check data in the host memory, and therefore, it includes any data transfers. The benchmarks were performed with input data sets of increasing size to make sure that the data paths are saturated. Data sets of 150 – 300 MB were typically sufficient although larger data sets are processed, e. g. 2000 MB on the RC100.

Table I summarizes for each device class the achieved sustained RS rate. For each device class the infrastructure of the benchmarks (processor type, hardware platform, programming environment and year of availability) and the number of threads N_{thr} started in the benchmark are given. In the context of this study the key properties of each platform are the theoretical bandwidth to host memory $B_{mem}^{(host)}$ and to locally attached memory $B_{mem}^{(local)}$. The achieved raw kernel rate R_k and sustained RS rate R_{RS} (both in MByte/s) are brought into relation to the corresponding theoretical maximum bandwidths: The RS efficiency E_{RS} is defined by $E_{RS} = R_{RS}/B_{mem}^{(host)}$, and correspondingly the kernel efficiency E_k is given by $E_k = R_k/B_{mem}^{(local)}$.

In this study, the IBM PowerXCell8i and Intel Nehalem platforms provide the highest absolute RS rates followed by the GPGPU, FPGA and at the very end the ClearSpeed system. Based on the RS efficiency E_{RS} one can basically classify the platforms into three **efficiency categories**:
(1) a 50+ category with the Cell platform,
(2) a 40 category with the FPGA, x86, and GPGPU-C1060 platforms, and
(3) a 20 category with the GPGPU-C870 and ClearSpeed platforms.
Both Cell and x86 platform benefit from their direct attached (host) memory, but using half of the physically available cores or SPUs an almost complete saturation of the memory interfaces is observed. PCIe-based platforms, GPGPU and ClearSpeed, show a breakdown of the I/O bound RS algorithm due to the "weak" PCIe link. If asynchronous streaming buffers are used on GPGPU the RS rates lowers slightly compared to synchronous transfers. On XCV4LX200 only $1/5$ of the available resources (flip-flops, slices) are used for the RS encoding offering space for additional processing kernels [1]. The RS rate on our FPGA platforms is limited by the effective I/O bandwidth.

V. Related Work

Standard Reed-Solomon encoding rates on GPUs for varying RS schemas are reported. Curry et. al. [8], [9] achieved for standard RS $k + 3$ ($k = 13, 29$) encoding schemas a throughput of 1.5 GB/s on NVIDIA GTX 260 and 100 MB/s on Intel Core 2. This compares well with our 1.6 GB/s on NVIDIA C870 whereas on x86 their achieved performance indicate the weaker FSB performance on the Intel platform

[1] Our implementation of the Galois-field variant shows a 12% improvement compared to Cauchy variant or 1592 MB/s on the RC100 platform.

TABLE I
BEST SUSTAINED CAUCHY REED-SOLOMON RATES R_{RS} AND ITS EFFICIENCY E_{RS} PER PLATFORM.

Class	Processor / Platform (Programming Environment)	Year	$B_{mem}^{(host)}$ [MB/s]	$B_{mem}^{(local)}$ [MB/s]	N_{thr}	R_k [MB/s]	E_k	R_{RS} [MB/s]	E_{RS}
x86:	Intel X5570 / SGI ICE8200+ (Intel compiler v.11)	2009	32,000	–	1	–	–	4,090	13%
				–	2	–	–	8,122	25%
				–	4	–	–	12,542	39%
				–	8	–	–	14,503	45%
GPGPU:	NVIDIA G80 / C870, Sun Ultra 27 (CUDA 2.3)	2007	8,000	76,800	16	12,924	17%	1,541	19%
					64	23,774	31%	1,630	20%
	NVIDIA T10 / C1060, SGI XE500 (CUDA 2.3)	2009	8,000	102,400	16	31,968	31%	3,181	40%
						–	–	[a]2,922	37%
					32	41,505	41%	3,255	41%
						–	–	[a]2,911	36%
SIMD:	ClearSpeed CSX 600 / CSX e620, Sun X4600M2 (ClearSpeed SDK 3.11)	2007	2,000	[b]3,200	96	[c]605	19%	[c]261	13%
						[d]406	13%	[d]384	19%
FPGA:	Xilinx XCV4LX200 / SGI RC100, SGI Altix450 (Mitrion 2.0.2, Xilinx ISE 9.2, RASClib 2.20)	2006	[e]3,200	[e]3,200	n.a.			1,442	45%
Cell:	IBM Power XCell8i / IBM QS22 (IBM SDK 3.1)	2008	25,600	–	1	–	–	5,551	22%
				–	2	–	–	10,375	41%
				–	4	–	–	14,023	55%
				–	8	–	–	14,476	57%

[a] Asynchronous I/O buffers are used for streaming.
[b] Bandwidth per CSX processor.
[c] Synchronous transfer mode.
[d] Asynchronous transfer mode.
[e] Bandwidth is valid for a 100 MHz design made by the Mitrion SDK.

and less optimization efforts compared to our implementation. Eschweiler et. al. [10] achieved 4.0 GB/s RS throughput on NVIDIA 8800 GTS card for a $5+3$ schema or 6% efficiency which may reflects different SDK versions and optimization efforts. The overall RS rate of 1 GB/s including data transfers over PCIe is only slightly less than ours.

Greenan et. al. [11] proposed an optimized approach for multiplications over Galois-fields $GF(2^l)$ and for this generalized approach a maximum RS rate of 174 MB/s on an (outdated) AMD Opteron system (2.4 GHz) was achieved.

The advantages of FPGA is manifested in existing solutions from storage vendors (BlueArc, DataDirect Networks).

VI. CONCLUSIONS

In this study IBM PowerXCell8i/QS22 and Intel Nehalem/SGI ICE+ provide the best raw RS throughput performance. The accelerator platforms PowerXCell8i, GPGPU-C1060 and FPGA can be considered as powerful off-loading engine for RS encoding. In a non-critical COTS based infrastructure the x86 platform is a convenient solution. In data intensive processing environments FPGAs being tightly integrated in the data path to the storage elements (disk, flash) provide energy-saving solutions without delimiting the flexibility.

ACKNOWLEDGMENTS

Michael Peick and Johannes Bock implemented initial versions on GPU and CellBE, and on the RC100 platform, respectively. Matthias Fouquet-Lapar (SGI) and Willi Homberg (Jülich Supercomputing Centre) provided access to benchmark platforms.

REFERENCES

[1] D. Hildebrand and R. Haskin, "IBM Almaden Research Center – Storage Systems," in *4th Petascale Data Storage Workshop Supercomputing '09, Portland, OR*, http://www.pdsi-scidac.org/events/PDSW09/resources/IBM1.pdf, November 2009.

[2] G. S. I. Reed, "Polynomial Codes over Certain Finite Fields," *Journal of the Society for Industrial and Applied Mathematics [SIAM J.]*, vol. 8, pp. 300–304, 1960.

[3] J. S. Plank, "A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems," *Software – Practice & Experience*, vol. 27, no. 9, pp. 995–1012, September 1997.

[4] J. Bloemer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An XOR-Based Erasure-Resilient Coding Scheme," International Computer Science Institute, August 1995.

[5] J. S. Plank and L. Xu, "Optimizing Cauchy Reed-Solomon Codes for Fault-Tolerant Network Storage Applications," in *NCA-06: 5th IEEE International Symposium on Network Computing Applications*, Cambridge, MA, July 2006.

[6] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, and Z. Wilcox-O'Hearn, "A Performance Evaluation and Examination of Open-Source Erasure Coding Libraries For Storage," in *FAST-2009: 7th Usenix Conference on File and Storage Technologies*, February 2009.

[7] Mitrionics AB., *Mitrion Users' Guide*, 2nd ed. Mitrionics, 2009.

[8] M. L. Curry, A. Skjellum, L. Ward, and R. Brightwell, "Accelerating Reed-Solomon coding in RAID systems with GPUs." in *IPDPS*. IEEE, 2008, pp. 1–6.

[9] ——, "Arbitrary Dimension Reed-Solomon Coding and Decoding for Extended RAID on GPUs." in *Proc. Petascale Data Storage Workshop PDSW'08*, 2008, pp. 1–3.

[10] A. Brinkmann and D. Eschweiler, "A microdriver architecture for error correcting codes inside the Linux kernel," in *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. New York, NY, USA: ACM, 2009, pp. 1–10.

[11] K. M. Greenan, E. L. Miller, and T. J. E. Schwarz, "Optimizing galois field arithmetic for diverse processor architectures and applications," in *Proc. 16th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems – MASCOTS'08*. IEEE, 2008, pp. 257–266.