Software ECC for Heterogeneous Distributed Systems

Qijia Jin

Blackbody Research
New York, NY

***Abstract*—CEDAC is an implementation of SECDED error correcting codes in software. With the introduction of denser memory configurations in commodity hardware, the lack of error detection and correction for reliability has become a concern. The device memory associated with CEDAC can scrub its memory without interfering with existing compute kernels. The library makes the underlying parity bit generation and error correction routines transparent to the compute kernel developer. The performance of the SECDED parity generation and syndrome decoding is measured on a NVIDIA Quadro RTX 5000.**

I. Introduction

Computing machines store data in memory prior to computations. The memory can experience bit flips due to environmental factors (cosmic radiation, high temperatures, etc) and hardware failure. These memory errors can be detected and sometimes corrected using error correcting codes (ECC). In this paper, we discuss the usage of single bit error correction and double bit error detection (SECDED) ECC codes. Using the parity bits of SECDED Hamming codes [4], the device can determine how many bitflips occurred by using the parity-check matrix to compute the syndrome vector of the associated data.

II. Concepts and Approach

The CEDAC library implements M. Y. Hsiao's proposed improvements on conventional or modified Hamming codes for (22, 16) SECDED code, (39, 32) SECDED code, and (72, 64) SECDED code (version 1 and 2) [5]. These Hsiao codes correspond to device global memory allocation sizes that are multiples of 2 bytes, 4 bytes, and 8 bytes. In addition, these Hsiao SECDED codes require parity bits to be allocated in the device global memory.

CEDAC provides EDAC and memory scrubbing for both OpenCL [3] and NVIDIA CUDA [9]. In the CEDAC library, memory scrubbing occurs on a 5 minute interval managed by a separate thread in the library. Programs using the CEDAC library can manually stop and restart the memory scrubbing. For increased memory error correction, this implementation of ECC requires manual modification of user programs to generate new parity bits and error detection and correction (EDAC) for the modified global device memory objects during the execution of the user's applications [7].

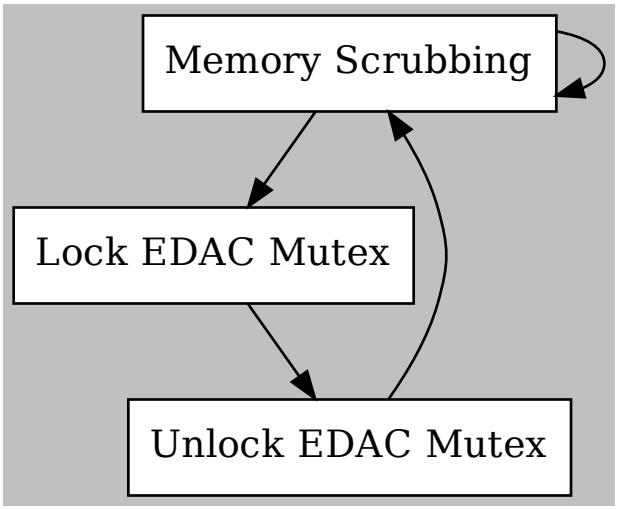

Fig. 1. Memory Scrubbing State Machine.

Using POSIX threads [1], the CEDAC library implements *patrol scrubbing* using a 5 minute interval by default. The user program can change the memory scrubbing interval using the CEDAC library APIs. To prevent memory scrubbing from occurring between compute kernel launches, the CEDAC library APIs allow the user program to lock the POSIX mutex [1] responsible for EDAC. When the launched user compute kernel(s) finishes computing, the user program can then unlock the POSIX mutex responsible for EDAC to resume memory scrubbing.

III. Implementation

CEDAC utilizes the CPU threads to compute Hsiao SECDED codes in the C language [6]. Those codes are used to error detect and correct any bitflips since the computation of the parity bits. Each CEDAC library handle has its corresponding CPU thread responsible for memory scrubbing their specified devices. Due to this threading model, the Julia language [2] implementation of CEDAC can directly call the C library without using Julia's experimental interface to multi-threading.

IV. Performance Evaluation

In order to evaluate the performance of CEDAC's NVIDIA CUDA implementation, the tests would require a NVIDIA GPU that provides native ECC support. The following benchmarks were done on a NVIDIA Quadro RTX 5000. The first test will compare the device allocation times by allocation size requested. The second test will compare the typical run time of 32-bit floating point GEMM (sGEMM).

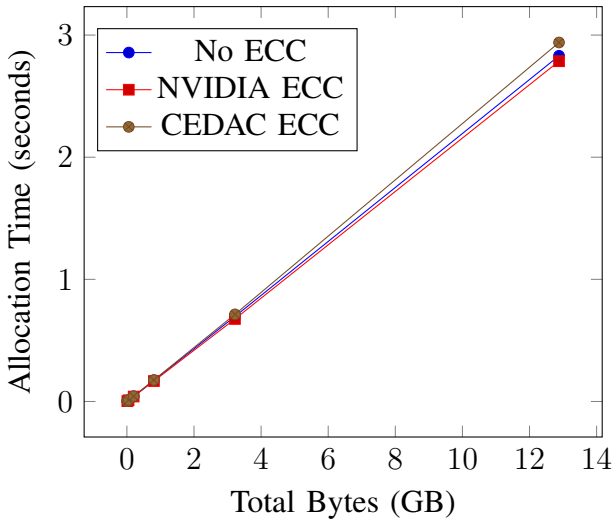

Fig. 2. Comparison of allocation times on a NVIDIA RTX 5000.

Fig. 2 is a comparison of device memory allocation times with and without ECC. When benchmarking the CEDAC ECC memory allocation time, the native ECC setting was disabled. The overhead of CEDAC's device memory allocation is slightly more than the native ECC device memory allocation as more device memory is requested.

To simulate a real workflow on the GPU, we time the execution of sGEMM in addition to the error detection and correction and the update of the parity bits for the data. Due to the nature of official BLAS libraries for CUDA and OpenCL, CEDAC cannot implement *demand scrubbing* during the compute kernel execution of sGEMM. However, the native ECC workflow can be approximately replicated with EDAC before calling sGEMM, calling sGEMM, and updating the parity bits of device memory after the completion of sGEMM. Fig. 3. shows the measurements of this proposed workflow on the RTX 5000.

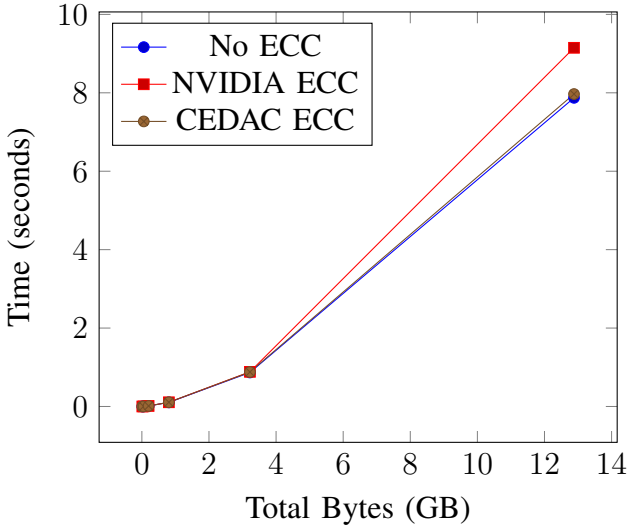

Fig. 3. Comparison of 32-bit floating point GEMM on a NVIDIA RTX 5000.

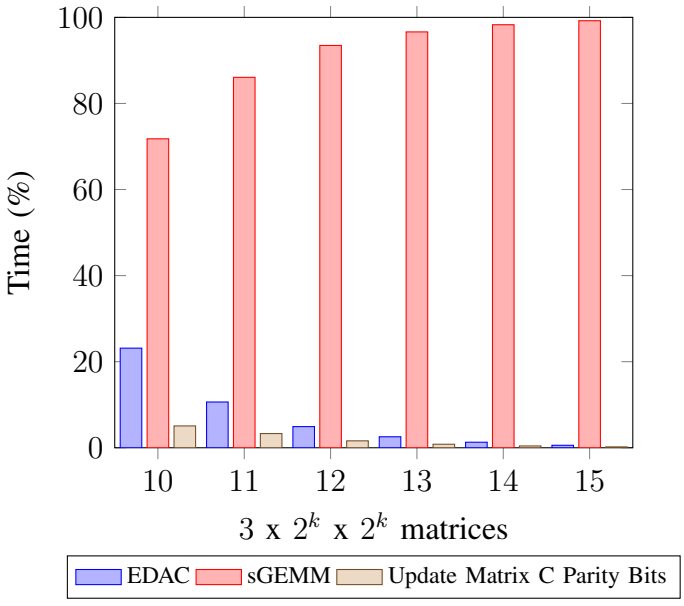

Fig. 4. Proposed CEDAC 32-bit floating point GEMM workflow on a NVIDIA RTX 5000.

During the tests of the proposed CEDAC sGEMM benchmarks, we made additional measurements for the individual steps of the workflow. Fig. 4. illustrates the impact of CEDAC's ECC implementation when used in a *patrol scrubbing* context. As the problem size increases for 32-bit floating point GEMM, the relative overhead of CEDAC's ECC falls off.

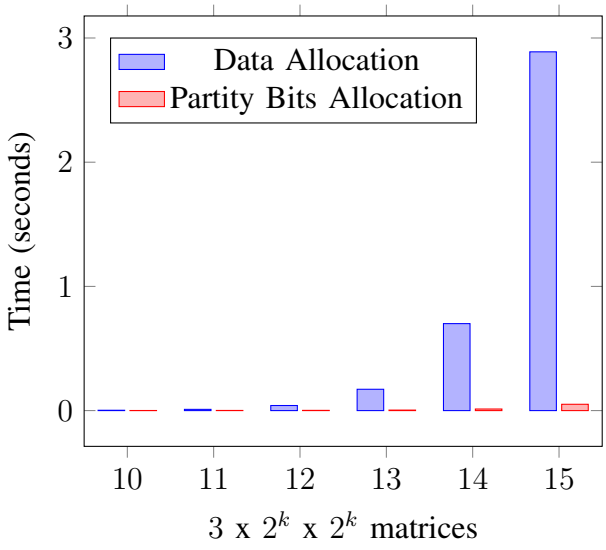

Fig. 5. CEDAC allocation size comparisons on a NVIDIA RTX 5000.

CEDAC's decreasing relative performance overhead on large device memory allocations is not limited to our proposed sGEMM workflow. In Fig. 5, the relative overhead of CEDAC's ECC during device memory allocation also decreases as device allocation size requested increases. At $k = 15$, each matrix is 4 GiB (32-bit floats are 4 bytes resulting in $4 * (2^{30})$ gibibytes) and the corresponding Hsiao(72, 64) codes (version 1 or 2) require 0.5 GiB for parity bits.

V. Conclusions

In this paper, we proposed a non-invasive *patrol scrubbing* and *demand scrubbing* implementation of SECDED error detection and correction. The CEDAC API is available in both C (CEDAC) and Julia (CEDAC.jl) to provide the CEDAC library in low level and high level languages. We measured the performance and overhead of CEDAC on a NVIDIA Quadro RTX 5000.

The CUDA implementation of ECC in the CEDAC library forces the compute kernels to use only the L2 cache by disabling the L1 cache on CUDA devices with compiler flags: -Xptxas -dlcm=cg [8]. OpenCL does not provide a similar method to disable memory access to the L1 cache. The CEDAC library does not protect against erroneous computations caused by GPU component failures or when bitfips occur in the L1 cache, L2 cache (CUDA only), and device registers.

References

[1] IEEE Standard for IEEE Information Technology - Portable Operating System Interface (POSIX(R)). *IEEE Std 1003.1-2001 (Revision of IEEE Std 1003.1-1996 and IEEE Std 1003.2-1992)*, pages 1–3678, December 2001.

[2] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017. URL https://doi.org/10.1137/141000671.

[3] K. O. W. Group. *The OpenCL 1.0 Specification*. Khronos Group, Beaverton, OR, December 2008.

[4] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, April 1950.

[5] M. Y. Hsiao. A Class of Optimal Minimum Odd-weight-column SEC-DED Codes. *IBM Journal of Research and Development*, 14(4): 395–401, July 1970.

[6] B. Kernighan and D. Ritchie. *The C Programming Language*. Prentice Hall, Englewood Cliffs, New Jersey, 1978.

[7] N. Maruyama, A. Nukada, and S. Matsuoka. Software-based ECC for GPUs. In *2009 Symposium on Application Accelerators in High Performance Computing (SAAHPC'09)*, volume 107, July 2009.

[8] P. Micikevicius. GPU performance analysis and optimization. In *2012 GPU Technology Conference*, San Jose, CA, May 2012.

[9] NVIDIA. *CUDA C Programming Guide, version 10*, October 2018.