GreenMM: Energy Efficient GPU Matrix Multiplication through Undervolting

Hadi Zamani
University of California, Riverside
Riverside, CA
hzama001@ucr.edu

Yuanlai Liu
University of California, Riverside
Riverside, CA
yliu158@ucr.edu

Devashree Tripathy
University of California, Riverside
Riverside, CA
devashree.tripathy@email.ucr.edu

Laxmi Bhuyan
University of California, Riverside
Riverside, CA
bhuyan@cs.ucr.edu

Zizhong Chen
University of California, Riverside
Riverside, CA
chen@cs.ucr.edu

ABSTRACT

The current trend of ever-increasing performance in scientific applications comes with tremendous growth in energy consumption. In this paper, we present GreenMM framework for matrix multiplication, which reduces energy consumption in GPUs through undervolting without sacrificing the performance. The idea in this paper is to undervolt the GPU beyond the minimum operating voltage (V_{min}) to save maximum energy while keeping the frequency constant. Since such undervolting may give rise to faults, we design an Algorithm Based Fault Tolerance (ABFT) algorithm to detect and correct those errors. We target cuBLAS Matrix Multiplication (cuBLAS-MM), as a key kernel used in many scientific applications. Empirically, we explore different errors and derive a fault model as a function of undervolting levels and matrix sizes. Then, using the model, we configure the proposed FT-cuBLAS-MM algorithm. We show that energy consumption is reduced uHigh Performance Computing (HPC) applications like molecular dynamics, weather prediction and drug discovery demand parallel processing environments. General Purpose Graphics Processing Units (GPGPUs) have evolved as high performance accelerators due to their SIMD (Single Instruction Multiple Data) processing architecture. Modern GPUs with hundreds of computing cores are capable of 7.8 $TFLOPs$ of double precision floating-point (FP64) and 15.7 $TFLOPs$ of single precision (FP32) [21]. Moreover, GPUs are equipped with huge memory bandwidth as high as 1 TBs. These characteristics make them well-suited for use as accelerators in HPC applications, especially for numerical computations and vector processing. Given their high computational capabilities, the GPUs consume a significant portion of the total system energy.

Matrix multiplication (MM) is heavily used in many important numerical computations. The matrix-multiplication kernel, referred to as GEMM in the Basic Linear Algebra Subroutines (BLAS) [20], is frequently used as a basic numerical calculation library in CPUs. GEMM routine is critical to the performance of High Performance LINPACK benchmark (HPL) and many software packages solving problem in linear algebra such as LAPACK, ScaLAPACK, MUMPS and SuperLU.p to 19.8%. GreenMM also improves the *GFLOPS/Watt* by 9% with negligible performance overhead.

KEYWORDS

Undervolting, Matrix multiplication, Fault tolerance, Energy efficiency

1 INTRODUCTION

High Performance Computing (HPC) applications like molecular dynamics, weather prediction and drug discovery demand parallel processing environments. General Purpose Graphics Processing Units (GPGPUs) have evolved as high performance accelerators due to their SIMD (Single Instruction Multiple Data) processing architecture. Modern GPUs with hundreds of computing cores are capable of 7.8 $TFLOPs$ of double precision floating-point (FP64) and 15.7 $TFLOPs$ of single precision (FP32) [21]. Moreover, GPUs are equipped with huge memory bandwidth as high as 1 TBs. These characteristics make them well-suited for use as accelerators in HPC applications, especially for numerical computations and vector processing. Given their high computational capabilities, the GPUs consume a significant portion of the total system energy.

Matrix multiplication (MM) is heavily used in many important numerical computations. The matrix-multiplication kernel, referred to as GEMM in the Basic Linear Algebra Subroutines (BLAS) [20], is frequently used as a basic numerical calculation library in CPUs. GEMM routine is critical to the performance of High Performance LINPACK benchmark (HPL) and many software packages solving problem in linear algebra such as LAPACK, ScaLAPACK, MUMPS and SuperLU.

Over the past few years, there have been significant efforts to study different techniques improving energy efficiency of GPUs such as Dynamic Voltage and Frequency Scaling (DVFS) [11][24], and load balancing in the CPU-GPU heterogeneous systems [23] [33]. However, DVFS techniques result in performance degradation due to lowering of the frequency. Also, they do not reduce static power consumption, which is becoming predominent in today's technology.

ICS '19, June 26–28, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6079-1/19/06…$15.00
https://doi.org/10.1145/3330345.3330373

The impact of undervolting for energy saving has thoroughly been analyzed recently by reducing the voltage down to the safe minimum voltage [15][16]. Leng et al. [15] explore energy benefits of reducing voltage of the GPU chip down to the safe limit. We aim to save even more energy through undervolting the GPU beyond the safe minimum operating voltage and tackling the possible GPU faults by employing a configurable low-overhead fault tolerant (FT) algorithm.

According to [15] and our observations, different applications affect the V_{min} at which the program executes correctly but fails when the voltage is reduced any further. The errors can be classified into Silent Data Corruption (SDC), Run-time Faults, Segmentation Faults, and Operating System (OS) crash. Some types of errors lead to divergence in the application control flow, and as a result, increase the execution time and even in some rare cases end up in an infinite loop [15]. The most predominant error is SDC. In Fast Fourier Transform (FFT), Matrix Multiplication and Hotspot benchmarks, the SDC errors lead to 24%, 42% and 55% faulty executions, respectively [27].

The reliability loss due to undervolting is not acceptable for most scientific computing cases. There are software level fault tolerant techniques such as DMR (dual modular redundancy) [26] and TMR (triple modular redundancy) [22], which take advantage of redundancies for handling erroneous cases, and checkpointing that tolerate errors in a checkpoint-restart manner [30]. These techniques are not very efficient for large scale scientific applications due to large energy and performance overheads [9]. In such cases, algorithm based fault tolerance (ABFT) [13], which tolerates errors at the application level, plays a crucial role in error detection and correction in the systems.

We introduce an energy efficient and ABFT framework, ***GreenMM***, which tolerates system errors due to undervolting. ABFT techniques in GPUs were introduced for MM [8], Cholesky [6] and Fast Fourier Transform [29]. Tan et al. [38] proposed a technique for undervolting CPUs and correcting errors through ABFT techniques. In our proposed framework, we use a combination of undervolting and ABFT for GPUs to guarantee energy, power, reliability, and performance efficiency of the system. First, we experimentally determine $V_{safeMin}$, which is the undervolting level beyond which the Operating System crashes for different applications. The proposed GreenMM framework exploits the voltage slack between V_{min} and $V_{safeMin}$ using a lightweight offline profiling to accurately predict the needed fault coverage capability as a function of matrix size, undervolting level and architectural details. We modify the offline ABFT algorithm by incorporating a number of faults. Online ABFT algorithms have also been proposed to reduce the overhead for detection and correction of large number of faults [13]. The basic idea is to decompose the large matrix into several blocks, which are individually protected through checksums. Unlike the offline algorithm, the overhead is lower and faults are not propagated to the output. GreenMM framework is developed for both offline and online algorithms. GreenMM achieves comparable performance (with 1.5% performance overhead) to highly optimized cuBLAS-MM in the cuBLAS library, but needs a lot less energy, which enhances the performance per watt of the GPU.

To summarize, GreenMM has two parts, GPU Undervolting model and **F**ault **T**olerant **cuBLAS-MM**. In *GPU Undervolting model* we determine the fault rate, V_{min} and $V_{safeMin}$ for the cuBLAS-MM.The undervolting is started from nominal voltage till V_{min}, during which no fault is encountered. However, when we undervolt further from V_{min} till $V_{safeMin}$, FT-cuBLAS-MM corrects the errors on the fly.

This paper makes the following contributions:

- We experimentally determine the Vmin and $V_{safeMin}$ for different applications, including matrix multiplication.
- We develop a fault model for GPU undervolting and determine number of faults as a function of matrix size and degree of undervolting.
- We design a fault tolerant framework, "*GreenMM*", for matrix multiplication that provides peak performance on GPUs. We incorporate the number of faults and modify the original cuBLAS-MM to implement offline and online FT-cuBLAS-MM algorithms.
- *GreenMM* is transparent to applications which utilize the matrix multiplications, i.e. it uses the same programming interface as cuBLAS-MM and GreenMM users do not need to modify source code of the cuBLAS (closed source).
- *GreenMM* is portable, i.e. it can be used with any GPU architecture just by changing some architecture specific parameters in the model.
- We present various experimental results in terms of energy, power, performance and reliability. GreenMM achieves up to 19.5% energy reduction compared to the original MM. Beside that, it improves the *GFLOPS/Watt* of the GPU up to 9%.

2 GPU UNDERVOLTING MODEL

Microprocessor manufacturers usually append an operating guard-band (a static voltage margin) as high as 20% of the nominal voltage, to ensure that the microprocessor functions reliably over varying load and environmental conditions [42]. The guard-bands also account for errors occurring from the load line, aging effects, noise and calibration error[32]. The guard-band grows with increase in variations in technology scaling. However, because we do not encounter these errors every time; significant energy saving can be achieved by reducing guard-band to a much lower supply voltage [17]. In our work, we aim at using the voltage slack between the nominal voltage and the actual OS safe voltage to save energy while preserving the performance. We use a similar approach as in [1] to reach $V_{safeMin}$, we also build a fault model empirically as a function of the undervolting level and matrix size. In GreenMM, we go a step further by aggressively undervolting and correcting subsequent errors using the ABFT. Shrinking microprocessor feature size and diminishing the noise guard-band increase the transient fault rate. We undervolt till the safe minimum voltage V_{min} without experiencing any faults. Going beyond V_{min}, system may experience soft errors. Although, GreenMM works for all kinds of soft errors, main focus is specifically on transient and computation errors such as SDCs [15]. SDC occurs when the program finishes its execution normally without any error message but results in a wrong output. These errors can be covered at the application level. CUDA run-time errors such as driver faults or segmentation faults caused by memory management drivers can be detected by inspecting the standard error output. Operating System crash occurs after a specific undervolting

level (application-dependent), and it is not possible to undervolt the GPU below the "OS crash point voltage" or $V_{safeMin}$.

2.1 Fault Distribution in GPU

In order to determine the number of faults to tolerate, we profile the application. We perform sensitivity analysis of different applications by reducing the voltage beyond V_{min} and by recording the faults at each voltage. The sensitivity analysis results help us to reach the minimum voltage at which we can tolerate errors for a given application. First, we execute an application at nominal voltage and record the output as "golden output". Then, starting from base voltage of 1.075V, the underlying GPU (GTX 980) is undervolted in step sizes of 10mV. The application is executed 100 times for each level of undervolting and the corresponding output is compared with the golden output to verify correctness. If the output does not match with the golden output, then the application has experienced a failure for that execution. To force the GPU to reduce its voltage at a fixed frequency, we reduce the target power limit of GPU. Fault distribution of different applications such as FFTD3D, FFTD2D, Histogram, MergeSort and BlackScholes on NVIDIA GTX 980 are shown in Figure 1. Applications that belong to Rodinia benchmark, are used extensively for performance evaluation of GPU architectures [4]. X-axis denotes the undervolting level starting at 1.075V, and Y-axis denotes the fault types along with their frequencies. Each application experiences different types of errors at different voltages. Some applications such as FFT2D, and FFTD3D show more number of SDC errors as compared to BlackScholes and MergeSort benchmarks. Since SDC errors can be handled at the application level, we only focus on SDC errors.

2.2 GPU Fault Model

The probability of failure is given by,

$$P_f = \frac{Number\ of\ failures}{Number\ of\ application\ runs} \tag{1}$$

P_f is derived by counting number of failures in Figure 1. Figure 2 shows P_f for different applications as a function of undervolting. V_{min} is the minimum voltage at which the program executes correctly. ($V_{safeMin}$) refers to theoretical lowest safe supply voltage under which the system can operate without crashing. As shown in Figure 2, applications have different undervolting levels for V_{min} and $V_{safeMin}$, which means different amounts of energy can be saved through undervolting while working with different applications. We observe a significant voltage guard-band whose margin varies from one application to another. As shown in Figure 2, we have more voltage guard-bands in Matrix Multiplication in when compared with the guard bands in other applications which means we can save more energy in case of MM.

Reliability of application R(t) at time t is the probability that there is no failure in the system until time t. We find R(t) where t is the execution time in equation 2.

$$R\left(t\right) = 1 - P_f\left(t\right) \tag{2}$$

The failure rate is obtained using Weibull lifetime reliability model, a well-accepted model for transient and permanent soft errors as in equation 3 [25]. Since we consider undervolting at a fixed frequency, the failure rate model is a function of supply voltage [38].

$$R\left(t\right) = e^{-\lambda t} \tag{3}$$

The failure rate calculated for different applications is shown in Figure 3, where X-axis represents the undervolting level and Y-axis represents the failure rate per minute. The failure rate of applications BlackScholes, FFTD2D, Histogram, FFTD3D, Mergesort and cuBLAS-MM are obtained experimentally. As shown in Figure 3, V_{min} for CUDA applications at a specific level of undervolting are different. In [15], it is observed that programs have different activity patterns which can lead to different voltage droops. The voltage droop is the main reason of GPU voltage noise. So, at a specific voltage, different intra and inter-kernel activities can lead to different failure rates. It is shown that the voltage noise, and specifically $\frac{di}{dt}$ droop, has the largest impact on $V_{safeMin}$ in [15]. Microarchitectural events, such as cache misses, cause pipeline stalls and large $\frac{di}{dt}$ droops lead to different guard-bands and $V_{safeMin}$. Because cuBLAS-MM is highly optimized, and all GPU components are active most of the time, there is no large $\frac{di}{dt}$ droop which could lead to lower voltage noise margin and larger guard-band.

3 GREENMM: ENERGY SAVING METHODOLOGY

GreenMM introduces an adaptive FT-cuBLAS-MM algorithm; which aggressively saves energy and power on GPUs through undervolting with a negligible performance overhead. GreenMM works with NVIDIA GPUs irrespective of the underlying GPU architecture.

Figure 4 shows the overview of GreenMM. GreenMM finds the maximum level of undervolting for the underlying GPU and configures the adaptive FT-cuBLAS-MM to tolerate the potential faults with regards to the failure rate of the underlying GPU at the maximum level of undervolting. To find the failure rate of cuBLAS-MM, GreenMM reduces the voltage of GPU progressively up to $V_{safeMin}$ and according to the fault model which is described in Section 2 find the failure rate of the GPU at each undervolting level. Then, based on the failure rate and execution time of given matrix, estimates the number of faults. Since these phases should be done before MM computation, execution time of MM is not determined. So, GreenMM uses an estimation model to predict the execution time of any arbitrary size. With multiplying the estimate execution time and failure rate of the GPU, the number of faults is determined and now we can configure the FT-cuBLAS-MM. It uses NVML library commands to reduce voltage of the GPU by changing the GPU target power limit and voltage offset.

3.1 Offline Profiling

GreenMM finds the optimum working voltage of the GPU for cuBLAS-MM, going beyond the V_{min} and correcting the potential errors. Incorporating fault tolerance mechanism increases the execution time, which in turn increases the energy consumption. GreenMM, carefully calibrates the level of undervolting so that the energy saving is more than the energy overhead. Optimum working voltage is found through an offline profiling phase which is done only once for each GPU. Offline profiling creates the failure rate model and MM execution time estimation model to estimate the number of faults for any MM sizes with regards to the underlying

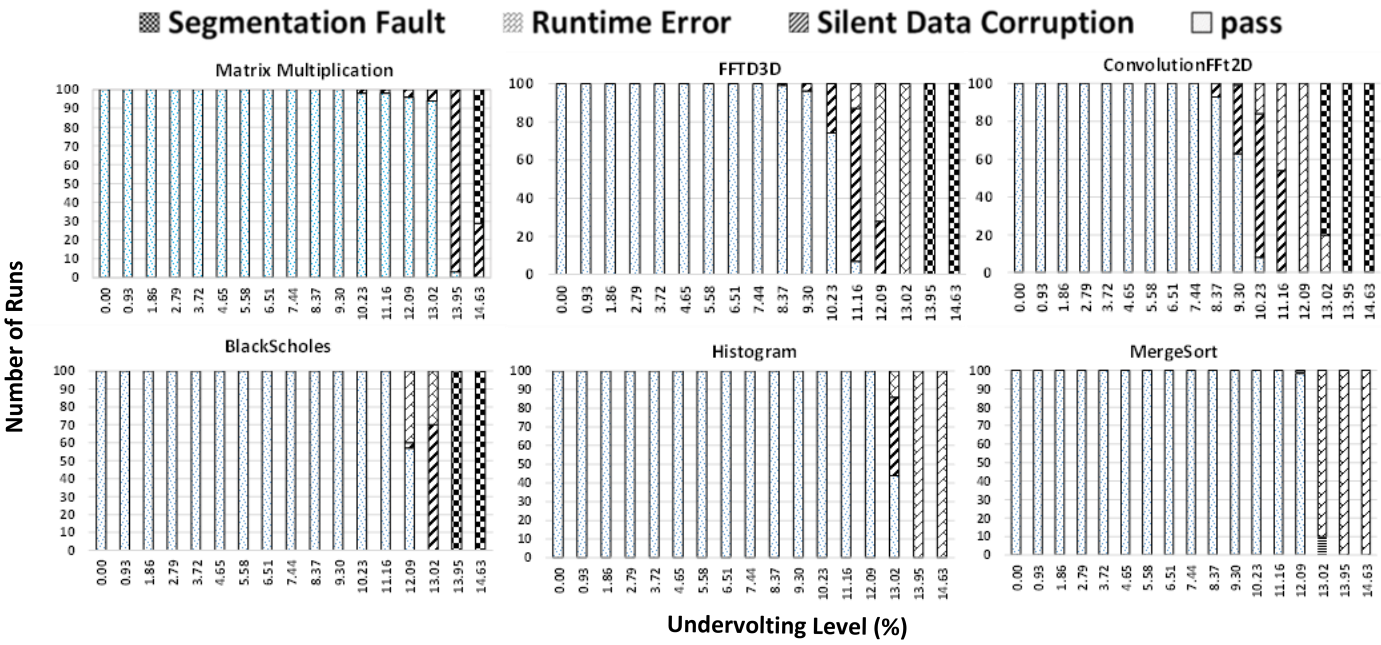

Figure 1: Error distribution below nominal voltage for different benchmarks using GTX 980

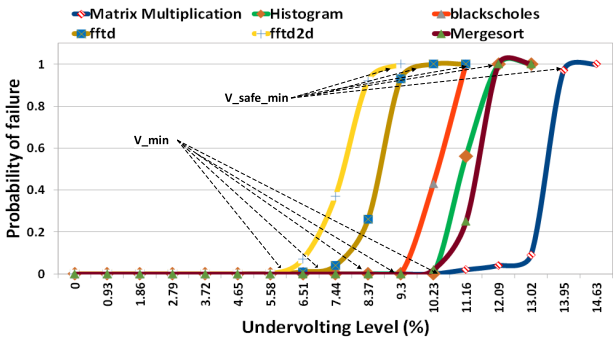

Figure 2: Probability of failure for different Rodinia benchmarks and cuBLAS-MM from cuBLAS library

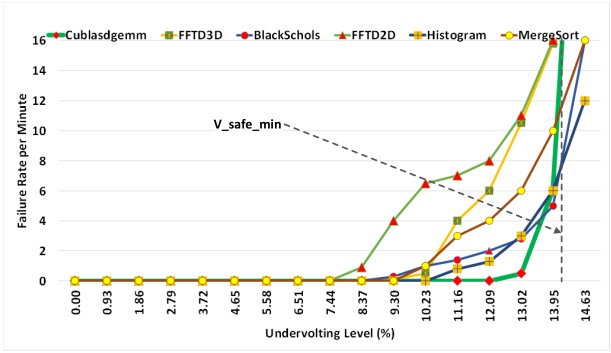

Figure 3: Failure rate of different Rodinia benchmarks and cuBLAS-MM from cuBLAS library

GPU. The offline profiling phase which is shown in Figure 5 is split into two parts:

3.1.1 Phase 1: Determine the maximum undervolting level ($V_{safeMin}$) and fault rate (λ). We execute matrices of small sizes on the GPU to minimize the profiling time and obtain maximum undervolting level ($V_{safeMin}$) and fault rate (λ), as described in Section 2.

In GreenMM, the offline profiling phase takes into account the aging effect. Also, the effects due to process variation and temperature were explored on various applications on different GPU cards in [15]. They concluded that process variation and temperature have a relatively uniform impact on $V_{safeMin}$ across all applications in a given GPU card; and the effect of aging is negligible (1-2 % in the long term). Moreover, The effect of temperature rise is already included in the number of faults, as plotted in Figures 1 and 8, because our fault model already considers the increase in temperature during long executions. However, we performed additional experiments to measure temperature while undervolting. Due to limited resources, we run MM in a loop to have enough time to observe the temperature changes. The variation in temperature of the GPU over time with and without undervolting is shown in Figure 6. It is observed that, after running kernel continuously, the temperature of GPU remains the same after a period of time and there is only about 11 °C variation in the temperature in presence of maximum level of undervolting. Ref. [15] shows that $V_{safeMin}$ at 70 °C is about $20mV$ higher than the values at 40 °C for various GPU cards. Temperature variation in GreenMM is about 11 °C which is not big enough to make a sensible change on $V_{safeMin}$.

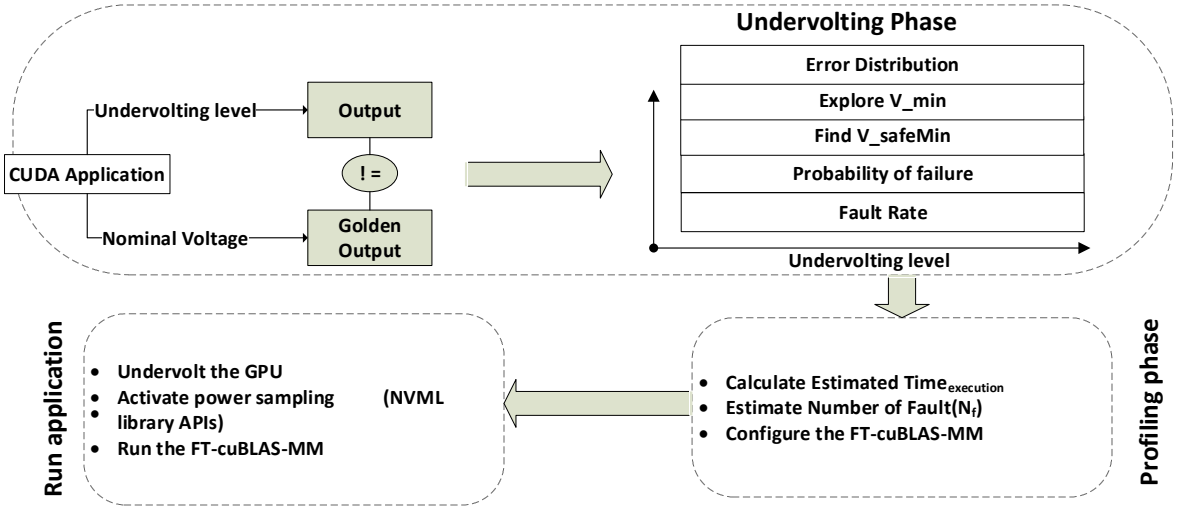

Figure 4: GreenMM overview

3.1.2 Phase 2: Estimate number of faults based on matrix size and fault rate (λ). The number of faults in an application can be obtained by multiplying λ with the execution time, as shown in equation 5. Failure rate remains same irrespective of the input data size for a given application as in equation 3. Hence, we estimate the execution time of MM for a given matrix size on a specific GPU through a simple profiling.

$$T = ax^3 + b \tag{4}$$

$$F = \lambda * T \tag{5}$$

Due to different compute resources like SM, register file size, cache sizes and shared memory size, execution time of the MM for a given size could be vary in different GPUs. Due to memory constraints, the GPU cannot handle matrix multiplication of any arbitrary size. The time complexity of cuBLAS-MM as a function of matrix size is provided in equation 4, where a and b are architecture-specific constants [36][31]. We run MM for different sizes to calculate the values of a and b for the underlying GPU. Figure 7 shows the experimental execution time (blue) and the execution time calculated theoretically from equation 4. Moreover, the red line shows sample points that were used to derive values of a and b which can be used for prediction of execution time for larger matrices, shown as green dashed line. Then, we compare the estimated execution time with the real experimental results for bigger matrices. As the results show, the estimation error is negligible.

Due to memory constraints on NVIDIA GTX 980, we use matrix of size 10K for GreenMM. The $V_{safeMin}$ is 86.05% of nominal voltage (undervolting level is 13.95%); the number of faults is 1.2 as shown in Figure 9. Hence, FT-cuBLAS-MM should tolerate at least 2 faults with the input size of 10K * 10K. If GPU memory supports matrices bigger than 10K * 10K, they may experience more number of faults.

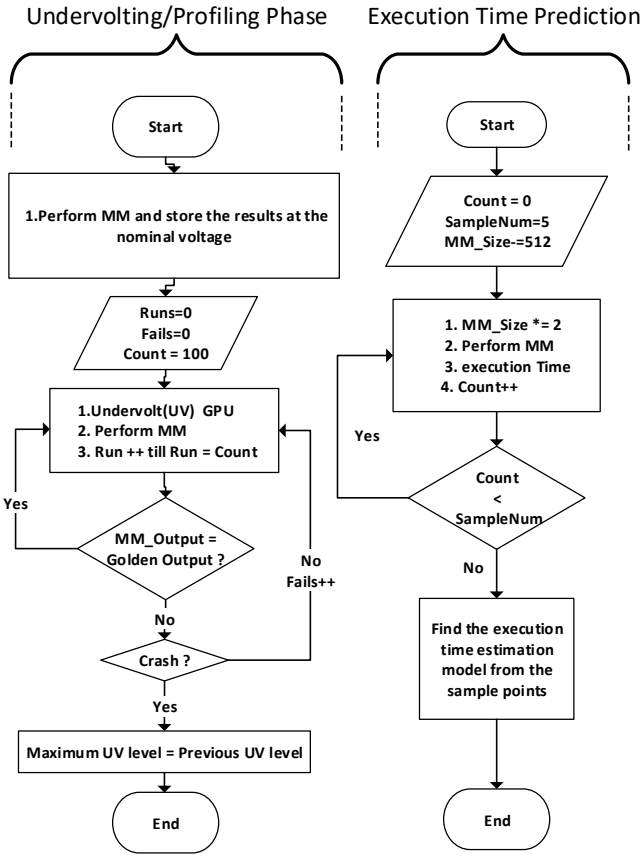

Figure 5: Overview of offline profiling

As the size of the matrix increases, the execution time as well as the number of faults also increase as shown in Figure 8. The

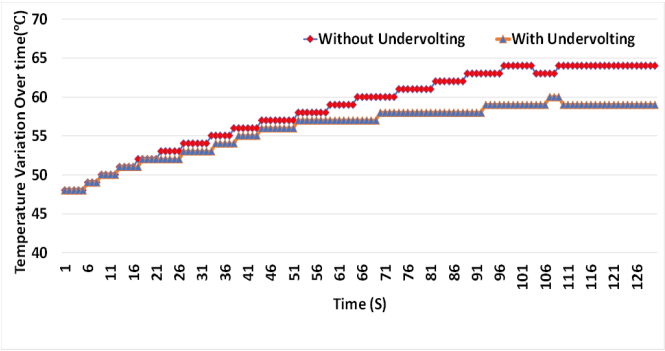

Figure 6: Temperature variation over time without and with maximum level of undervolting

matrix size varies between 10K and 100K as the undervolting level is changed from 0% to 13.95% in Y-axis; Z-axis shows the number of faults. For a given undervolting level, the number of faults for large matrix sizes is more than the number of faults in small matrices.. In the following, we propose an adaptive FT algorithm than can be configured to handle different number of faults.

3.2 Offline FT-cuBLAS-MM

The ABFT for Matrix Multiplication has a very low performance overhead when compared with other techniques [3]. The basic idea of ABFT is to encode input matrices with checksums to detect and correct the corrupted data. The traditional ABFT introduced by Huang et al. [13] is capable of correcting one fault by checking correctness at the very end of computation. In our work, we introduce an enhanced offline version, FT-cuBLAS-MM, which is capable of tolerating any arbitrary number of faults by increasing the number of weighted check-sum vectors. Algorithm 1 describes the pseudo-code for the offline FT-cuBLAS-MM.

Generating the weights of the checksum vectors, encoding the column checksums, and the row checksum are done according to algorithms into the matrix is done according algorithms 2, 3 and 4 respectively. The result of $C^f = A^c * B^r$ is a full checksum matrix. At the end of computation, we check full checksum relationship again and if the relationship does not hold, then our result is faulty; thereafter, faults are detected and corrected using equation 6.

$$C_{ij} = \Sigma_{j=1}^{n} c_{ij}^{f} - \Sigma_{k=1,k\neq j}^{n} c_{ik}^{f} \tag{6}$$

Algorithm 1 The pseudo-code for Detection Phase

1: Generate checksum weights vectors v_1 and v_2
2: Encode $A \rightarrow A^c$
3: Encode $B \rightarrow B^r$
4: $C^f = A^c \times B^r$
5: Recompute the checksum for C^f
6: Verify full checksum relationship of C^f

3.3 Online FT-cuBLAS-MM

Offline FT-cuBLAS-MM only checks correctness of results at the end of computation. We design an online version of FT-cuBLAS-MM to check correctness of MM during computation, so that we can prevent faults to be propagated. We introduce an Online FT-cuBLAS-MM that can handle different number of faults. Fault coverage capability of FT-cuBLAS-MM is determined before starting the MM computation. However, the key problem here is that we must use MM algorithm such that it maintains the checksum relationship even in the middle of the computation. In [7], it is proved that outer product Matrix Multiplication maintains checksum relationship in each iteration of computation. For a matrix with size of N, we have at most N opportunities to tolerate faults during the entire MM computation. The fault detection phase, which is always active, increases the performance overhead. So, to achieve high performance, we can invoke the FT-cuBLAS-MM routine once in every several iterations. There is a trade-off between the number of iterations and overhead of online FT-cuBLAS-MM. The number of iterations to invoke FT-cuBLAS-MM is closely related to the number of faults that may happen during the computation. If the failure rate of system increases, then we should check more frequently, otherwise, there is no need to employ an algorithm with higher fault coverage capability. The algorithm to perform MM has several steps. The detailed steps of the algorithm are shown in algorithm 5.

During each iteration, we update checksum of the result matrix to maintain full checksum relationship. Then, we compute sum of each row and column in the result matrix and compare it with the row and column checksum. If the check is passed we move to the next iteration, otherwise, if any checksum does not match, we locate the exact position of error through comparing the checksums. To correct the error (C_{ij}), we simply add the difference of j_{th} checksum column and the sum of j_{th} column to the result matrix element at location (i, j). GreenMM corrects two errors at the same time regardless of the error patterns. Also, it corrects any number of errors which may happen in the same row or column.

Algorithm 2 Generating weighted checksum vectors for each block

1: **for** $i = 0, 1, ..., nb$ **do**
2: $v_1[i] = 1$
3: **end for**
4: **for** $i = 0, 1, ..., nb$ **do**
5: $v_2[i] = 1 + i$
6: **end for**

Algorithm 3 Column checksum update for A(m * k)

1: **for** $j = 0, 1, ..., k-1$ **do**
2: **for** $i = 0, 1, ..., m-1$ **do**
3: $ColChk_{v1}[j] = \Sigma_{i=0}^{m-1} V_1[i] * A[i][j]$
4: $ColChk_{v2}[j] = \Sigma_{i=0}^{m-1} V_2[i] * A[i][j]$
5: **end for**
6: **end for**

As shown in algorithm 5, online FT-cuBLAS-MM algorithm consists of following steps:

(1) Move input matrices to the GPU using cudaMemCpy API.
(2) Generate checksum weights vectors in the CPU and move them to the GPU. The weights are generated according to algorithm 2. Due to frequent accesses to weights vector in GPU, to get peak performance, pitched device memory is

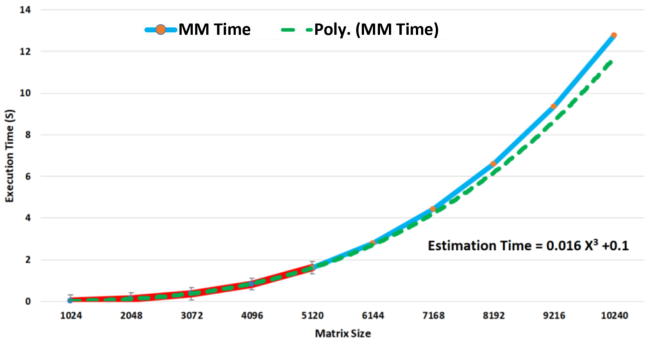

Figure 7: Execution time estimation model vs. the real execution time

Algorithm 4 Row checksum update for B(k * n)

1: **for** $i = 0, 1, ..., k-1$ **do**
2: **for** $j = 0, 1, ..., n-1$ **do**
3: $RowChk_{v1}[i] = \sum_{j=0}^{n-1} V1[j] * B[i][j]$
4: $RowChk_{v2}[i] = \sum_{j=0}^{n-1} V2[j] * B[i][j]$
5: **end for**
6: **end for**

allocated using cudaMallocPitch API that allocates linear memory space for better efficiency in terms of performance and power.

(3) Divide input matrices into blocks given the number of faults and do MM without checksums.
(4) Invoke cuBLAS-MM to update column checksum for each block according to algorithm 3.
(5) Invoke cuBLAS-MM to update C
(6) Invoke cuBLAS-MM to update row checksum of B given the equations described in algorithm 4.
(7) Update row checksum of C by invoking cuBLAS-MM
(8) Recalculate column and row checksums of C by invoking a simple kernel which adds elements of the result matrix.

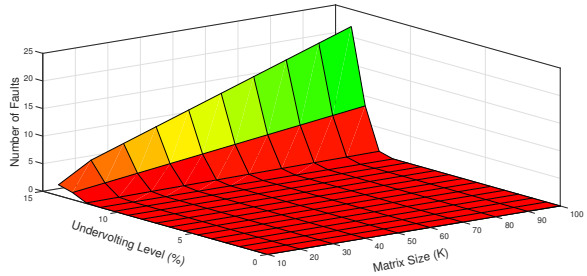

Figure 8: Estimated Number of faults for different matrix sizes given the undervolting levels.

Algorithm 5 Pseudo-code for online FT-cuBLAS-MM

1: Initialization
NB = N (Matrix Size)/ B (Block Size)
2: **for** $i = 1, ..., NB$ **do**
3: $A_B, B_B \rightarrow GPU$
4: Update $C_B \rightarrow$ cuBLAS-MM(A_B, B_B, C_B)
5: Update $A_B{}^c \rightarrow$ cuBLAS-MM$(A_B, ColChk_v, A_B^c)$
6: Update $C_B^c \rightarrow$ cuBLAS-MM(A_B^c, B_B, C_B^c)
7: Update $B_B^r \rightarrow$ cuBLAS-MM$(B_B, RowChk_v, B_B^r)$
8: Update $C_B^r \rightarrow$ cuBLAS-MM(A_B, B_B^r, C_B^r)
9: Recalculate $\rightarrow$ $C_B\_ColChk2$
10: **while** $C_B\_ColChk1 \neq C_B\_ColChk2$ **do**
11: Do Correction
12: **end while**
13: Recalculate $\rightarrow$ $C_B\_RowChk2$
14: **while** $C_B\_RowChk1 \neq C_B\_RowChk2$ **do**
15: Do Correction
16: **end while**
17: Update C
18: **end for**
19: $C \rightarrow CPU$

(9) Compare recalculated checksums and old checksums to locate the potential error. Any potential errors can be located by comparing the column and the row checksums. Since computers do floating point calculations in finite precision, the checksum relationship can not hold exactly due to round-off errors. So, we need a threshold to distinguish between round-off errors and computation errors. Too large thresholds may hide the computation errors, while, too small thresholds may interrupt correct computation. In comparison phase, according to [40], e^{-10} has been chosen as a conservative threshold to distinguish between round-off and computation errors.
(10) Correct any potential errors according to equation 6

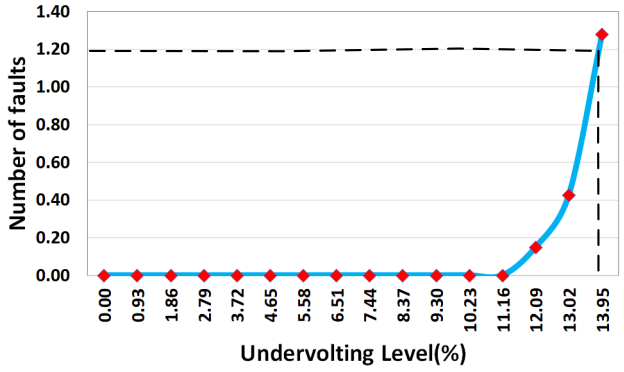

Figure 9: Number of faults according to the undervolting level for matrices with size of 10K on NVIDIA GTX 980

4 EVALUATION

4.1 Experimental Setup

All experiments are performed on NVIDIA GTX 980 [1], the architectural specifications can be found in Table 2. Given the limited memory size of the GPU, we were able to evaluate the results for up to a matrix size of 10K. We reduced nominal voltage of the GPU in step sizes of 10 - 12mV until the $V_{OSCrashpoint}$ using the MSI After Burner [15]. By decreasing the target power limit of the GPU, we can enforce specific operating voltage. We use NVIDIA System Management Interface (Nvidia-smi), a widely used command line utility on top of NVIDIA Management Library (NVML), to measure power consumption of the GPU at 10ms intervals. Some important commands on power management in NVIDIA GPUs are shown in Table 1.

The execution times of cuBLAS-MM and FT-cuBLAS-MM are shown in Figure 10. The overhead of fault tolerance is large for small matrices, however, the overhead decreases with increase in the matrix size. For small matrices, there is 8% performance overhead, while in case of bigger matrices (10K), performance overhead of FT-cuBLAS-MM comes down to 1.5%.

There is no need for fault tolerance till V_{min} as the probability of error occurrence is zero. The detection phase is activated when undervolting beyond V_{min} to detect potential errors, however, the correction phase is activated only if an error is detected in the detection phase. The detection phase accounts for majority of the overhead in the FT-cuBLAS-MM. For instance, when the matrix size is 10K, the detection phase takes 139ms while the correction phase takes only 0.24ms, which means the number of faults to be corrected has low impact on the performance. In case of 10K matrix size, the maximum number of faults we need to tolerate is 1.2 as shown in Figure 9; which can be handled by offline FT-cuBLAS-MM. Offline FT-cuBLAS-MM is a special case of online FT-cuBLAS-MM when the number of faults is less than or equal to 2. Here, the block size is the same as the matrix size.

4.2 Performance and Energy Saving Evaluation of FT-cuBLAS-MM

When the matrix size increases, the failure rate remains the same. However, the number of errors increases. To evaluate the overhead of FT-cuBLAS-MM, faults are injected directly into partial sum results at random locations and in random iterations according to fault model described in Section 2. Fault injection in a controlled manner emulates the impact of hardware transient faults on MM computation. We observed errors in the output, however, they were detected and corrected by the offline or online FT-cuBLAS-MM depending on the size of matrix and the number of faults.

FT-cuBLAS-MM as described in Section 3, improves the reliability of computation and tolerates any arbitrary number of faults. This is because, we check correctness of the partial results in each iteration. There is a trade-off between reliability of computation, energy consumption and performance overhead. We measure the performance (*GFLOPS*) of the cuBLAS-MM and FT-cuBLAS-MM on NVIDIA GTX 980 in the presence of different number of faults for a 10K matrix. Since the actual number of faults at $V_{safeMin}$ (i.e. 13.95%) undervolting level is 1.2, we evaluate the performance overhead by manually injecting faults into 10K matrix. Increase in the number of faults results in increased performance overhead, as shown in Figure 11. The performance is 165 *GFLOPS* in presence of 2 errors and 162 *GFLOPS* in presence of 16 faults. On average, the performance overhead for different number of faults is 1.5%.

The energy consumption of the GPU is calculated by multiplying power (at each undervolting level) with the execution time of MM. Figure 12 shows the energy saving in FT-cuBLAS-MM versus the original cuBLAS-MM in presence of different undervolting levels and number of faults. Since no fault occurs till V_{min}, fault detection phase is disabled.

The fault detection and correction phases are activated when we undervolt from V_{min} till $V_{safeMin}$. The X-axis denotes the undervolting level and the corresponding number of faults. Figure 12 (a) shows the energy saving at different undervolting levels for matrix size of 10K with and without fault tolerance. Undervolting level at V_{min} for the original cuBLAS-MM is 10.23% without any faults. Undervolting beyond V_{min} results in faults; and the maximum number of faults is 1.2 at undervolting level of 13.95% as shown in Figure 9. So, offline FT-cuBLAS-MM is used to correct the faults. For a matrix of size 10K, the cuBLAS-MM can save energy up to 14% just by undervolting and without any fault tolerance, but with GreenMM the energy saving is increased up to 19.8% due to undervolting beyond V_{min}.

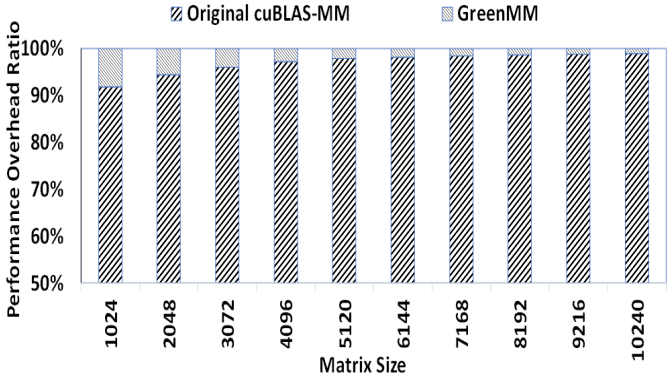

Figure 10: Performance overhead of matrix multiplication for different matrix sizes in presence of two errors

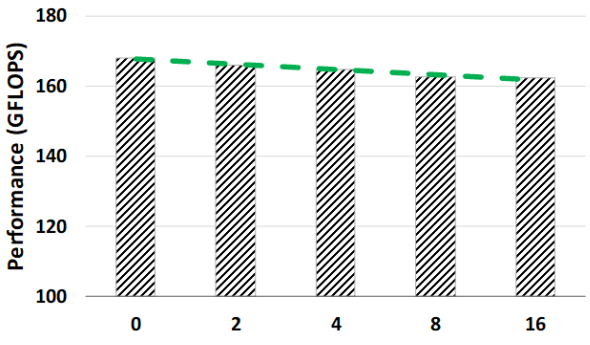

Figure 11: Performance evaluation of the FT-cuBLAS-MM

Table 1: Power management commands using the NVML library

Command	Description
nvmlDeviceGetPowerUsage	Retrieves power usage for the GPU and its associated circuitry in milliwatts
nvmlDeviceSetPersistenceMode	Enables persistent mode to prevent driver from unloading
nvmlDeviceSetPowerManagementLimit	Sets new power limit for the device
nvmlDeviceSetApplicationsClocks	Sets clocks that applications will lock to
Accuracy	Power Measurement Accuracy & Reading is accurate to within +/- 5% of the current power draw

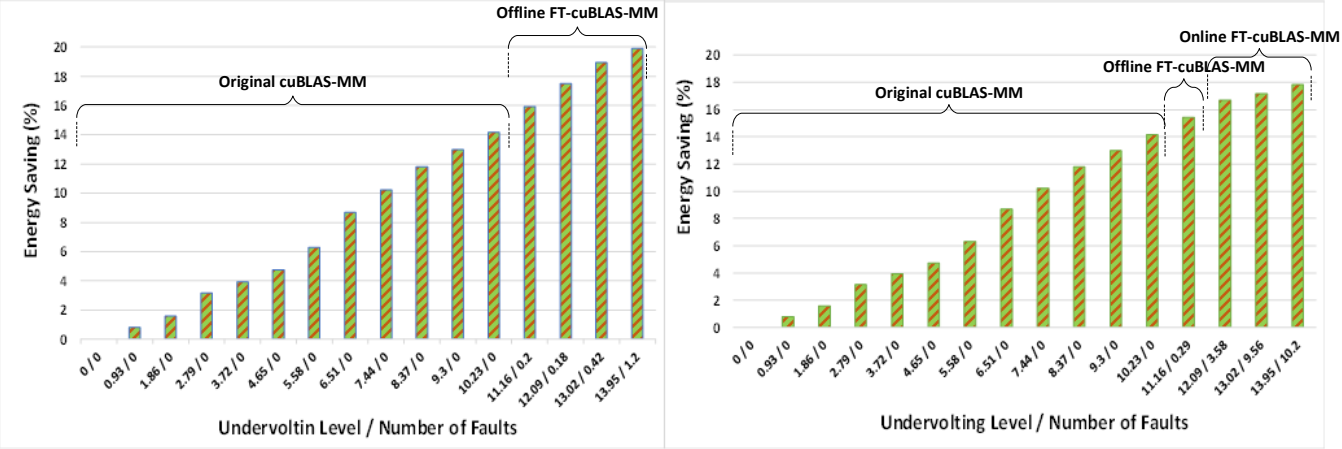

Figure 12: Energy saving in the FT-cuBLAS-MM versus the original cuBLAS-MM given different undervolting levels and number of faults

Table 2: NVIDIA GTX 980 specifications

Processor	2048 CUDA-core NVIDIA Maxwell GeForce GTX 980
Peak Perf.	4.6 TFLOPs
Memory	4 GB GDDR5
Base Clock	1126 MHz
Boost Clock	1216 MHz
Memory Clock	7 GHz
Default Voltage	1.075 V

Figure 12 (b) shows the energy saving at different undervolting levels for matrix size 40K with and without fault tolerance. Undervolting beyond V_{min}, results in faults; and the maximum number of faults is 10.2 at undervolting level 13.95%. Offline FT-cuBLAS-MM with two weighted-check sum vectors can not cover those number of faults. Hence, we activate the Online FT-cuBLAS-MM to tolerate the faults. Although, going beyond V_{min} results in more number of faults, we can still save 4% additional energy by activating the FT-cuBLAS-MM. For a matrix of size 40K, when there are no faults, the cuBLAS-MM saves energy up to 14% with undervolting, but when using undervolting in combination with FT-cuBLAS-MM the energy saving increases to 18%.

4.3 Performance/Watt and Total Energy Consumption Evaluation

cu-BLAS-MM is not open source; so, the number of operations cannot be calculated accurately; however, the number of floating point operations that take place when multiplying 2 matrices can be estimated according to equation 7.

$$N_{fp} = 2n^3 - n^2 \tag{7}$$

With assuming the same amount of operations in both cases (with and without ABFT), and measuring the extra execution time which is needed for ABFT part, we can have a fair comparison.

As shown in Figure 13, despite the performance overhead of ABFT, GreenMM has higher performance per watt (*GFLOPS/Watt*) in comparison to the original cuBLAS-MM. This is because, we can save significant power by just undervolting the GPU. Figure 13 shows *GFLOPS/Watt* of the GPU. X-axis shows the number of faults and the Y-axis shows *GFLOPSWatt* improvement ratio when compared with the performance of the original cuBLAS-MM without undervolting. When there are two faults, at 13.95% undervolting level, GreenMM improves *GFLOPSWatt* of the GPU by 9%. When the number of faults increases to 16, there will be 7% improvement in *GFLOPSWatt* over the original cuBLAS-MM without undervolting.

To have an explicit comparison, we also plot the total energy consumption for multiplying two matrices with input size of 10K in presence of different number of faults. X-axis shows the number of faults and the Y-axis shows the total energy consumption. The first left column shows the original cu-BLAS-MM energy consumption,

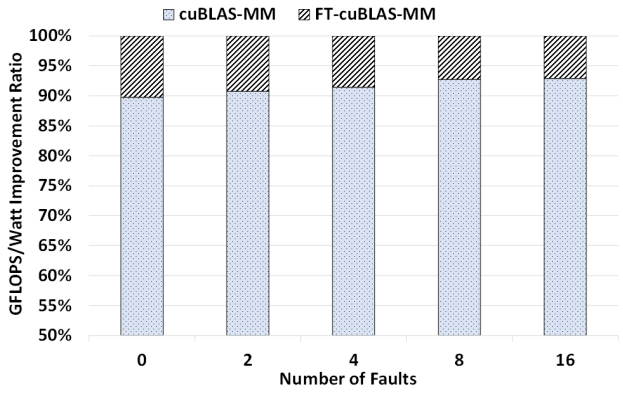

Figure 13: Comparing performance in GPU with default voltage versus undervolted GPU in presence of different number of faults.

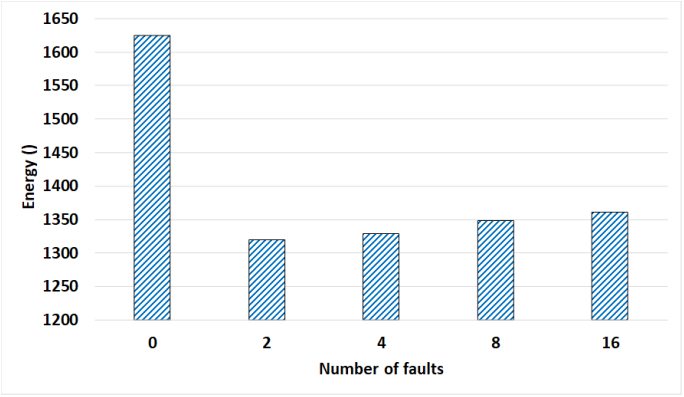

Figure 14: Energy consumption at the default voltage versus optimum level of undervolting with matrix input size of 10k.

while the other column shows the energy consumption of GreenMM in presence of different number of faults. To evaluate GreenMM with number of faults more than 2, we manually inject faults into the partial results during the computation at the optimum undervolting level. The results show that when the original cu-BLAS-MM is used without any faults, the GPU consumes more than 1600 Joules, whereas, the GreenMM consumes about 1300 Joules for multiplying the matrices in presence of 2 faults.

MM heavy applications such HPL and ScaLAPACK involves a time-consuming task to deal with MM computation. Trailing matrix updates consumes more than 90% of the computation cost in HPL [39]. GreenMM can be employed to compute this phase. Since GreenMM is transparent to the users, it can be integrated into HPL and other MM heavy applications supporting GPUs.

5 RELATED WORK

The ever-increasing popularity of GPUs has motivated development of energy efficient GPU architectures, most of which target for energy saving in general over many applications. However, very few of the architecture designs are targeted at reducing energy consumption of linear algebra basic routines such as cuBLAS-MM that are used in scientific application.

Dynamic Voltage and Frequency Scaling (DVFS), is one common approach to reduce power and energy consumption of a system [19]. Applying DVFS, based on system utilization, the processors can operate in different power states whenever high performance is not necessary. DVFS in GPU domain can behave in a very different manner compared to DVFS in CPUs in regard to energy efficiency [10]. Besides the DVFS technique, GPU undervolting is another approach for improving GPU energy efficiency. Leng et al. [16], reduce chip voltage of the GPU to V_{min} without introducing any errors; which was achieved by leveraging guard-band voltage of the GPU. However they did not go beyond V_{min} because errors would occur with any further undervolting. In our work, we show that even beyond V_{min} there is opportunity to save more energy and correct potential faults by combining undervolting and Algorithm Based Fault Tolerance (ABFT) together. GPUs/CPUs use huge number of communications links which have made them seriously prone to coupling and inductance effects [35][34]. By using undervolting, we also could relax the coupling and inductance effect and increase the reliability. In [41] [12], power gating is applied onto GPU to save energy on branch divergence and idle components respectively. [2] applies dynamic resource allocation to improve GPU energy efficiency. [37] attempts to reduce energy consumption by selecting between the CPU or GPU to run the application. In CPU domain, there are several studies which rely on hardware sensors to look for possibilities to reduce the operating voltage by monitoring critical path [14]. In [38], Tan et al. investigated the interplay between energy efficiency and reliability on the CPUs. In their approach, they combined undervolting with a fault tolerant technique to tolerate faults caused by undervolting on the CPUs. Their fault rate model is based on digital circuit failure, and not based on the CPU hardware. It is because they could not drive CPU undervolting to below the threshold value to generate faults. So, they emulated the errors and corrected them. They used an analytic fault model and only considered a single soft error to correct. In GreenMM, we introduce a fault model taking into account the real system faults during undervolting through NVML APIs. We drove the GPU voltage under the threshold ($V_{safeMin}$), so that the number of faults is practically measured. Thus, our proposed fault model is more accurate and realistic.

Fault tolerant mechanisms such as redundancy-based techniques [28] can recover from hard failures, however, at a huge performance cost. These techniques are not useful in GPU applications due to high penalty in terms of energy consumption and performance. Checkpointing has been applied to tolerate failures on the GPU through restarting application from some previously saved correct state [30]. Checkpointing suffers from significant performance and memory overheads. Compared to aforementioned techniques, ABFT provides the advantage of negligible overhead along with the capability of detecting and correcting errors with low overhead. ABFT has widely been studied for improving linear algebra library on both CPUs [40][18] and GPUs [5].

6 CONCLUSION

This paper presented a technique to save energy in GPUs through undervolting. First, we profiled error distribution of different applications from Rodinia benchmark to create an empirical fault model based on behaviour of the applications, while reducing the GPU

voltage beyond V_{min}. After this point, the most predominant error was SDC error, which can be corrected at the application level. Then, we designed a ABFT based fault tolerant matrix multiplication algorithm, called FT-cuBLAS-MM, to correct the errors dynamically. We evaluated energy consumption and performance on NVIDIA GTX 980. Our experiments showed that energy consumption can be reduced up to 19.8% using GreenMM, with performance overhead of 1.5%. Moreover, The *GFLOPS/WATT* improvement of the GreenMM in comparison to the original cuBLAS-MM for a matrix of size 10K is 9%.

ACKNOWLEDGMENT

This work is supported by NSF Grants CCF-1423108, CCF-1513201. The authors would like to thank the anonymous reviewers for their invaluable comments and suggestions.

REFERENCES

[1] [n. d.]. GTX 980Ti Specifications. https://www.anandtech.com/show/8526/nvidia-geforce-gtx-980-review/21

[2] Pedro Alonso, Manuel F Dolz, Francisco D Igual, Rafael Mayo, and Enrique S Quintana-Orti. 2012. Reducing energy consumption of dense linear algebra operations on hybrid CPU-GPU platforms. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*. IEEE, 56–62.

[3] George Bosilca, Rémi Delmas, Jack Dongarra, and Julien Langou. 2009. Algorithm-based fault tolerance applied to high performance computing. *J. Parallel and Distrib. Comput.* 69, 4 (2009), 410–416.

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

[5] Jieyang Chen, Sihuan Li, and Zizhong Chen. 2016. Gpu-abft: Optimizing algorithm-based fault tolerance for heterogeneous systems with gpus. In *Networking, Architecture and Storage (NAS), 2016 IEEE International Conference on*. IEEE, 1–2.

[6] Jieyang Chen, Xin Liang, and Zizhong Chen. 2016. Online algorithm-based fault tolerance for cholesky decomposition on heterogeneous systems with gpus. In *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE, 993–1002.

[7] Zizhong Chen. 2008. Extending algorithm-based fault tolerance to tolerate fail-stop failures in high performance distributed environments. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. IEEE, 1–8.

[8] Chong Ding, Christer Karlsson, Hui Liu, Teresa Davies, and Zizhong Chen. 2011. Matrix multiplication on gpus with on-line fault tolerance. In *Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th International Symposium on*. IEEE, 311–317.

[9] David Fiala, Frank Mueller, Christian Engelmann, Rolf Riesen, Kurt Ferreira, and Ron Brightwell. 2012. Detection and Correction of Silent Data Corruption for Large-scale High-performance Computing. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '12)*. IEEE Computer Society Press, Los Alamitos, CA, USA, Article 78, 12 pages. http://dl.acm.org/citation.cfm?id=2388996.2389102

[10] Rong Ge, Ryan Vogt, Jahangir Majumder, Arif Alam, Martin Burtscher, and Ziliang Zong. 2013. Effects of Dynamic Voltage and Frequency Scaling on a K20 GPU. In *Proceedings of the 2013 42Nd International Conference on Parallel Processing (ICPP '13)*. IEEE Computer Society, Washington, DC, USA, 826–833. http://dx.doi.org/10.1109/ICPP.2013.98

[11] JoÃčo Guerreiro, Aleksandar Ilic, Nuno Roma, and Pedro TomÃąs. 2018. DVFS-aware application classification to improve GPGPUs energy efficiency. *Parallel Comput.* (2018). https://doi.org/10.1016/j.parco.2018.02.001

[12] Sunpyo Hong and Hyesoon Kim. 2010. An Integrated GPU Power and Performance Model. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA '10)*. ACM, New York, NY, USA, 280–289. https://doi.org/10.1145/1815961.1815998

[13] Kuang-Hua Huang et al. 1984. Algorithm-based fault tolerance for matrix operations. *IEEE transactions on computers* 100, 6 (1984), 518–528.

[14] Charles R. Lefurgy, Alan J. Drake, Michael S. Floyd, Malcolm S. Allen-Ware, Bishop Brock, Jose A. Tierno, and John B. Carter. 2011. Active Management of Timing Guardband to Save Energy in POWER7. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44)*. ACM, New York, NY, USA, 1–11. https://doi.org/10.1145/2155620.2155622

[15] Jingwen Leng, Alper Buyuktosunoglu, Ramon Bertran, Pradip Bose, and Vijay Janapa Reddi. [n. d.]. Safe Limits on Voltage Reduction Efficiency in GPUs: A Direct Measurement Approach. In *Proceedings of the 48th International Symposium on Microarchitecture (MICRO-48)*. ACM, New York, NY, USA, 294–307. https://doi.org/10.1145/2830772.2830811

[16] Jingwen Leng, Yazhou Zu, and Vijay Janapa Reddi. 2014. Energy efficiency benefits of reducing the voltage guardband on the Kepler GPU architecture. *Proc. of SELSE* (2014).

[17] J. Leng, Y. Zu, M. Rhu, M. S. Gupta, and V. J. Reddi. 2014. GPUVolt: Modeling and characterizing voltage noise in GPU architectures. In *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. 141–146. https://doi.org/10.1145/2627369.2627605

[18] Xin Liang, Jieyang Chen, Dingwen Tao, Sihuan Li, Panruo Wu, Hongbo Li, Kaiming Ouyang, Yuanlai Liu, Fengguang Song, and Zizhong Chen. 2017. Correcting soft errors online in fast fourier transform. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 30.

[19] Wenjie Liu, Zhihui Du, Yu Xiao, David A Bader, and Chen Xu. 2011. A waterfall model to achieve energy efficient tasks mapping for large scale GPU clusters. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*. IEEE, 82–92.

[20] Mark Harris Luke Durant, Olivier Giroux and Nick Stam. 2001. Basic Linear Algebra Subprograms Technical(BLAST) Forum Standard. http://www.netlib.org/blas/blast-forum/blas-report.pdf

[21] Mark Harris Luke Durant, Olivier Giroux and Nick Stam. 2017. Volta Whitepaper. https://devblogs.nvidia.com/inside-volta

[22] Robert E Lyons and Wouter Vanderkulk. 1962. The use of triple-modular redundancy to improve computer reliability. *IBM Journal of Research and Development* 6, 2 (1962), 200–209.

[23] Kai Ma, Xue Li, Wei Chen, Chi Zhang, and Xiaorui Wang. 2012. Greengpu: A holistic approach to energy efficiency in gpu-cpu heterogeneous architectures. In *Parallel Processing (ICPP), 2012 41st International Conference on*. IEEE, 48–57.

[24] Xinxin Mei, Ling Sing Yung, Kaiyong Zhao, and Xiaowen Chu. 2013. A Measurement Study of GPU DVFS on Energy Conservation. In *Proceedings of the Workshop on Power-Aware Computing and Systems (HotPower '13)*. ACM, New York, NY, USA, Article 10, 5 pages. https://doi.org/10.1145/2525526.2525852

[25] DN Prabhakar Murthy, Min Xie, and Renyan Jiang. 2004. *Weibull models*. Vol. 505. John Wiley & Sons.

[26] Sayori Nakagawa, Satoshi Fukumoto, and Naohiro Ishii. 2003. Optimal checkpointing intervals of three error detection schemes by a double modular redundancy. *Mathematical and Computer Modelling* 38, 11 (2003), 1357 – 1363. https://doi.org/10.1016/S0895-7177(03)90138-5 Stochastic models in engineering, technology, and management.

[27] D. A. G. Oliveira, P. Rech, L. L. Pilla, P. O. A. Navaux, and L. Carro. 2014. GPGPUs ECC efficiency and efficacy. In *2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. 209–215.

[28] David Patterson, Aaron Brown, Pete Broadwell, George Candea, Mike Chen, James Cutler, Patricia Enriquez, Armando Fox, Emre Kiciman, Matthew Merzbacher, David Oppenheimer, Naveen Sastry, William Tetzlaff, Jonathan Traupman, and Noah Treuhaft. 2002. *Recovery Oriented Computing (ROC): Motivation, Definition, Techniques,*. Technical Report. Berkeley, CA, USA.

[29] Laercio L Pilla, P Rech, F Silvestri, Christopher Frost, Philippe Olivier Alexandre Navaux, M Sonza Reorda, and Luigi Carro. 2014. Software-based hardening strategies for neutron sensitive FFT algorithms on GPUs. *IEEE Transactions on Nuclear Science* 61, 4 (2014), 1874–1880.

[30] James S Plank, Micah Beck, Gerry Kingsley, and Kai Li. 1994. *Libckpt: Transparent checkpointing under unix*. Computer Science Department.

[31] Ronald L Rivest and Charles E Leiserson. 1990. *Introduction to algorithms*. McGraw-Hill, Inc.

[32] N. Rohbani, M. Ebrahimi, S. Miremadi, and M. B. Tahoori. 2017. Bias Temperature Instability Mitigation via Adaptive Cache Size Management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25, 3 (March 2017), 1012–1022. https://doi.org/10.1109/TVLSI.2016.2606579

[33] N. Rohbani, Z. Shirmohammadi, M. Zare, and S. Miremadi. 2017. LAXY: A Location-Based Aging-Resilient Xy-Yx Routing Algorithm for Network on Chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36, 10 (Oct 2017), 1725–1738. https://doi.org/10.1109/TCAD.2017.2648817

[34] Z. Shirmohammadi and H. Z. Sabzi. 2018. DR: Overhead Efficient RLC Crosstalk Avoidance Code. In *2018 8th International Conference on Computer and Knowledge Engineering (ICCKE)*. 63–68. https://doi.org/10.1109/ICCKE.2018.8566456

[35] Z. Shirmohammadi, H. Z. Sabzi, and S. G. Miremadi. 2017. 3D-DyCAC: Dynamic numerical-based mechanism for reducing crosstalk faults in 3D ICs. In *2017 IEEE International High Level Design Validation and Test Workshop (HLDVT)*. 87–90. https://doi.org/10.1109/HLDVT.2017.8167468

[36] Steven S Skiena. 1998. *The algorithm design manual: Text*. Vol. 1. Springer Science & Business Media.

[37] H. Takizawa, K. Sato, and H. Kobayashi. 2008. SPRAT: Runtime processor selection for energy-aware computing. In *2008 IEEE International Conference on Cluster Computing*. 386–393.

[38] L. Tan, S. L. Song, P. Wu, Z. Chen, R. Ge, and D. J. Kerbyson. 2015. Investigating the Interplay between Energy Efficiency and Resilience in High Performance Computing. In *2015 IEEE International Parallel and Distributed Processing Symposium*. 786–796. https://doi.org/10.1109/IPDPS.2015.108

[39] Q. Wang, J. Ohmura, S. Axida, T. Miyoshi, H. Irie, and T. Yoshinaga. 2010. Parallel Matrix-Matrix Multiplication Based on HPL with a GPU-Accelerated PC Cluster. In *2010 First International Conference on Networking and Computing*. 243–248. https://doi.org/10.1109/IC-NC.2010.39

[40] Panruo Wu and Zizhong Chen. 2014. FT-ScaLAPACK: Correcting soft errors online for ScaLAPACK Cholesky, QR, and LU factorization routines. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*. ACM, 49–60.

[41] Qiumin Xu and Murali Annavaram. 2014. PATS: pattern aware scheduling and power gating for GPGPUs. In *Parallel Architecture and Compilation Techniques (PACT), 2014 23rd International Conference on*. IEEE, 225–236.

[42] Yazhou Zu, Charles R Lefurgy, Jingwen Leng, Matthew Halpern, Michael S Floyd, and Vijay Janapa Reddi. [n. d.]. Adaptive guardband scheduling to improve system-level efficiency of the POWER7+. In *Proceedings of the 48th International Symposium on Microarchitecture*. ACM, 308–321.