**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 ($V_{\text{min}}$) 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 up 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 predominant in today’s technology.

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 $V_{\text{min}}$ 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 $V_{\text{safeMin}}$, which is the undervolting level beyond which the Operating System crashes for different applications. The proposed GreenMM framework exploits the voltage slack between $V_{\text{min}}$ and $V_{\text{safeMin}}$ 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 Fault Tolerant cuBLAS-MM. In **GPU Undervolting model** we determine the fault rate, $V_{\text{min}}$ and $V_{\text{safeMin}}$ for the cuBLAS-MM. The undervolting is started from nominal voltage till $V_{\text{min}}$, during which no fault is encountered. However, when we undervolt further from $V_{\text{min}}$ till $V_{\text{safeMin}}$, FT-cuBLAS-MM corrects the errors on the fly.

This paper makes the following contributions:

- We experimentally determine the $V_{\text{min}}$ and $V_{\text{safeMin}}$ 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 UNDERVEROLTING 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 $V_{\text{safeMin}}$, 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 $V_{\text{min}}$ without experiencing any faults. Going beyond $V_{\text{min}}$, 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 $V_{\text{safeMin}}$.

### 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 $V_{\text{min}}$ 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: Error distribution below nominal voltage for different benchmarks using GTX 980

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 FFT2D3D 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,

\[ P_f = \frac{\text{Number of failures}}{\text{Number of application runs}} \]  \hspace{1cm} (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. \( V_{\text{min}} \) is the minimum voltage at which the program executes correctly. \( V_{\text{safeMin}} \) 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 \( V_{\text{min}} \) and \( V_{\text{safeMin}} \), 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.

\[ R(t) = 1 - P_f(t) \]  \hspace{1cm} (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].

\[ R(t) = e^{-\lambda t} \]  \hspace{1cm} (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, FFT2D2D, Histogram, FFT2D3D, Mergesort and cuBLAS-MM are obtained experimentally. As shown in Figure 3, \( V_{\text{min}} \) 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 \( \frac{\text{di}}{\text{dt}} \) droop, has the largest impact on \( V_{\text{safeMin}} \) in [15]. Microarchitectural events, such as cache misses, cause pipeline stalls and large \( \frac{\text{di}}{\text{dt}} \) droops lead to different guard-bands and \( V_{\text{safeMin}} \). Because cuBLAS-MM is highly optimized, and all GPU components are active most of the time, there is no large \( \frac{\text{di}}{\text{dt}} \) 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 3 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 $V_{\text{safeMin}}$ 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 $V_{\text{min}}$ 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 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 ($V_{\text{safeMin}}$) and fault rate ($\lambda$). We execute matrices of small sizes on the GPU to minimize the profiling time and obtain maximum undervolting level ($V_{\text{safeMin}}$) and fault rate ($\lambda$), as described in Section 2.

3.1.2 Phase 2: Estimate number of faults based on matrix size and fault rate ($\lambda$). The number of faults in an application can be obtained by multiplying $\lambda$ 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.

$$T = ax^3 + b$$ (4)

$$F = \lambda \times T$$ (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.

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 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.

![Figure 4: GreenMM overview](image_url)

![Figure 5: Overview of offline profiling](image_url)
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 \( C^f = A^c \times B^r \) 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.

\[
C_{ij} = \sum_{j=1}^{m} C_{ij}^f - \sum_{k=1}^{n} f^i_j c_{ik}^f
\]

**Algorithm 1** The pseudo-code for offline FT-cuBLAS-MM

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. Verify full checksum relationship of \( C^f \)

**Algorithm 2** Generating weighted checksum vectors for each block

1. \( \text{for } i = 0, 1, \ldots, nb \text{ do} \)
2. \( v_1 [i] = 1 \)
3. \( \text{end for} \)
4. \( \text{for } i = 0, 1, \ldots, nb \text{ do} \)
5. \( v_2 [i] = 1 + i \)
6. \( \text{end for} \)

**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 3** Column checksum update for \( A(m \times k) \)

1. \( \text{for } j = 0, 1, \ldots, k - 1 \text{ do} \)
2. \( \text{for } i = 0, 1, \ldots, m - 1 \text{ do} \)
3. \( \text{ColChk}_{i,1} [j] = \sum_{i=0}^{m-1} V_1[i] + A[i] \times j \)
4. \( \text{ColChk}_{i,2} [j] = \sum_{i=0}^{m-1} V_2[i] + A[i] \times j \)
5. \( \text{end for} \)
6. \( \text{end for} \)

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
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: \[ \text{RowChk}_1[i] = \sum_{j=0}^{n-1} V_1[j] \times B[i][j] \]
4: \[ \text{RowChk}_2[i] = \sum_{j=0}^{n-1} V_2[j] \times B[i][j] \]
5: end for
6: end for

GPU, to get peak performance, pitched device memory is 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.

(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

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

1: Initialization
2: $\text{NB} = N \text{(Matrix Size)}/ B \text{(Block Size)}$
3: for $i = 1, ..., \text{NB}$ do
4: \[ A_B, B_B \rightarrow \text{GPU} \]
5: Update $C_B \rightarrow \text{cuBLAS-MM}(A_B, B_B, C_B)$
6: Update $A_B \rightarrow \text{cuBLAS-MM}(A_B, \text{RowChk}_k, A_B)$
7: Update $C_B \rightarrow \text{cuBLAS-MM}(B_B, \text{RowChk}_k, B_B)$
8: Update $C_B \rightarrow \text{cuBLAS-MM}(A_B, B_B, C_B)$
9: Recalculate $\rightarrow C_B, \text{ColChk}_2$
10: while $C_B, \text{ColChk}_1 \neq C_B, \text{ColChk}_2$ do
11: Do Correction
12: end while
13: Recalculate $C_B, \text{RowChk}_2$
14: while $C_B, \text{RowChk}_1 \neq C_B, \text{RowChk}_2$ do
15: Do Correction
16: end while
17: Update C
18: end for
19: $C \rightarrow \text{CPU}$

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 - 12 mV until the $V_{Osc\crashpoint}$ 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

![Figure 8: Estimated Number of faults for different matrix sizes given the undervolting levels.](image1)

![Figure 9: Number of faults according to the undervolting level for matrices with size of 10K on NVIDIA GTX 980](image2)

314
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 $V_{\text{min}}$ as the probability of error occurrence is zero. The detection phase is activated when undervolting beyond $V_{\text{min}}$ 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 139 ms while the correction phase takes only 0.24 ms, 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 ($\text{GFLOPS}$) 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 $V_{\text{safeMin}}$ (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 $\text{GFLOPS}$ in presence of 2 errors and 162 $\text{GFLOPS}$ 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. The execution times are 139 ms for a matrix size of 10K, hence, the energy consumption of the GPU is 18.82 mW ms. The detection phase takes 139 ms while the correction phase takes only 0.24 ms, 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.

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvmlDeviceGetPowerUsage</td>
<td>Retrieves power usage for the GPU and its associated circuitry in milliwatts</td>
</tr>
<tr>
<td>nvmlDeviceSetPersistenceMode</td>
<td>Enables persistent mode to prevent driver from unloading</td>
</tr>
<tr>
<td>nvmlDeviceSetPowerManagementLimit</td>
<td>Sets new power limit for the device</td>
</tr>
<tr>
<td>nvmlDeviceSetApplicationsClocks</td>
<td>Sets clocks that applications will lock to</td>
</tr>
</tbody>
</table>

Accuracy: Power Measurement Accuracy & Reading is accurate to within +/- 5% of the current power draw

Table 2: NVIDIA GTX 980 specifications

<table>
<thead>
<tr>
<th>Processor</th>
<th>2048 CUDA-core NVIDIA Maxwell GeForce GTX 980</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak Perf.</td>
<td>4.6 TFLOPs</td>
</tr>
<tr>
<td>Memory</td>
<td>4 GB GDDR5</td>
</tr>
<tr>
<td>Base Clock</td>
<td>1126 MHz</td>
</tr>
<tr>
<td>Boost Clock</td>
<td>1216 MHz</td>
</tr>
<tr>
<td>Memory Clock</td>
<td>7 GHz</td>
</tr>
<tr>
<td>Default Voltage</td>
<td>1.075 V</td>
</tr>
</tbody>
</table>

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.

\[ N_{fp} = 2n^3 - n^2 \]  

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 GFLOPS/Watt 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 GFLOPS/Watt of the GPU by 9%. When the number of faults increases to 16, there will be 7% improvement in GFLOPS/Watt 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, 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 $V_{\text{min}}$ without introducing any errors; which was achieved by leveraging guard-band voltage of the GPU. However they did not go beyond $V_{\text{min}}$ because errors would occur with any further undervolting. In our work, we show that even beyond $V_{\text{min}}$ 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 ($V_{\text{safeMin}}$), 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 $V_{\text{min}}$. After this point, the most predominant error was SDC error, which can be corrected at the application level. Then, we designed an 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 $\text{GFLOPS/WATT}$ 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.