Improving Energy Saving of One-sided Matrix Decompositions on CPU-GPU Heterogeneous Systems

Jieyang Chen
jchen3@uab.edu
University of Alabama at Birmingham
Birmingham, Alabama, USA

Xin Liang
xliang@uky.edu
University of Kentucky
Lexington, Kentucky, USA

Kai Zhao
kzhao@uab.edu
University of Alabama at Birmingham
Birmingham, Alabama, USA

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

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

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

Abstract

One-sided dense matrix decompositions (e.g., Cholesky, LU, and QR) are the key components in scientific computing in many different fields. Although their design has been highly optimized for modern processors, they still consume a considerable amount of energy. As CPU-GPU heterogeneous systems are commonly used for matrix decompositions, in this work, we aim to further improve the energy saving of one-sided matrix decompositions on CPU-GPU heterogeneous systems. We first build an Algorithm-Based Fault Tolerance protected overclocking technique (`ABFT-OC`) to enable us to exploit reliable overclocking for key matrix decomposition operations. Then, we design an energy-saving matrix decomposition framework, Bi-directional Slack Reclamation (`BSR`), that can intelligently combine the capability provided by `ABFT-OC` and DVFS to maximize energy saving and maintain performance and reliability. Experiments show that `BSR` is able to save up to 11.7% more energy compared with the current best energy saving optimization approach with no performance degradation and up to 14.1% $Energy \times Delay^2$ reduction. Also, `BSR` enables the Pareto efficient performance-energy trade-off, which is able to provide up to 1.43× performance improvement without costing extra energy.

CCS Concepts: • **Hardware → Power and energy**; • **Computer systems organization → Dependable and fault-tolerant systems and networks**.

Keywords: GPU, matrix decomposition, energy saving, fault tolerance

PPoPP '23, February 25-March 1, 2023, Montreal, QC, Canada
© 2023 Association for Computing Machinery.
ACM ISBN 979-8-4007-0015-6/23/02...$15.00
https://doi.org/10.1145/3572848.3577496

1 Introduction

To meet performance requirements for current mission-critical scientific computing, millions of computing cores are equipped in modern High Performance Computing (HPC) systems consuming tens of megawatts of power [16]. With the increasing need for higher performance, it is anticipated that future HPC systems will consist of even more computing cores and consume more power. As HPC systems are achieving higher parallelism, how to achieve high performance and energy efficiency while ensuring computing reliability has become a critical challenge for scientific computing.

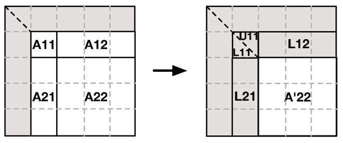
As the type of processor that contributes the most of the computing parallelism in many current and future HPC systems, Graphics Processing Units (GPUs), equipped with thousands of low-power cores, offer high computational power and energy efficiency. Many applications and libraries have been designed and optimized for GPU accelerators [1, 3, 8, 9, 13, 25, 34, 36, 42, 43]. Benefiting from the fact that GPUs are designed for highly parallelizable computations while CPUs are more efficient with serial computations, CPUs and GPUs that are linked through fast interconnections [30, 31] are usually used together to form heterogeneous systems that can efficiently handle a large spectrum of scientific computing workloads. Many scientific applications or software begin to have an optimized design for CPU-GPU heterogeneous systems such as the MAGMA linear algebra library [15].

One-sided dense matrix decomposition such as Cholesky, LU, and QR play a pivotal role in many scientific applications. Their state-of-the-art designs for CPU-GPU heterogeneous systems are proposed in [44, 45], and they have been highly optimized in the MAGMA library and used as key computational kernels by many applications across different fields [18, 21, 23, 24, 35, 46, 53].

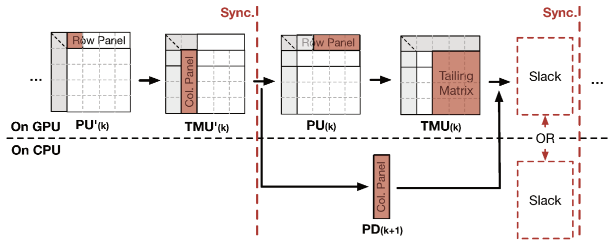
Many works have been done to improve the energy efficiency of linear algebra operations. [7] proposed to use a DVFS-based approach to optimize matrix decompositions. [2, 51, 52] seek to use reduced core supply voltage to reduce the energy consumption of matrix-matrix multiplication operations. Although much work has been done to improve the energy saving of matrix decomposition on CPU-GPU heterogeneous systems, it is still desirable to further improve their energy saving since matrix decompositions as they still consume a considerable amount of energy. Improving the energy saving of matrix decomposition can lead to more energy-efficient scientific computing. However, the major challenge, as pointed out in [27, 28, 38, 51, 52], is that aggressive energy-saving optimizations can weaken the reliability of the system and cause performance degradation, which is unacceptable for time-sensitive and mission-critical scientific applications.

In this work, we aim to further improve the energy saving of one-sided matrix decompositions on CPU-GPU heterogeneous systems while maintaining performance and reliability. We first build an Algorithm-Based Fault Tolerance protected overclocking technique (`ABFT-OC`) to enable us to exploit reliable overclocking for key matrix decomposition operations. Then, we design an energy-saving matrix decomposition framework, Bi-directional Slack Reclamation (`BSR`), that can intelligently combine the capability provided by `ABFT-OC` and Dynamic Voltage and Frequency Scaling (DVFS) to maximize energy saving and maintain performance and reliability. Also, `BSR` enables the Pareto efficient performance-energy trade-off. Specifically, our contributions are listed as follows:

- We propose the first adaptive algorithm-based fault tolerance protected overclocking technique (`ABFT-OC`) for matrix decompositions on CPU-GPU heterogeneous systems. Overclocking with an optimized voltage guardband can enable us to exploit higher clock frequencies with higher energy efficiency. However, aggressive overclocking can decrease system reliability, so we propose to couple ABFT with overclocking to enable trustable computation. To reduce fault tolerance overhead, we further propose a lightweight adaptive-ABFT technique that automatically adjusts its fault tolerance strength according to the error rate.
- Next, based on `ABFT-OC`, we propose a novel slack-based energy saving framework - Bi-directional Slack Reclamation (`BSR`), which aims to exploit *slack*, processor idle time, to save energy and enable flexible Pareto efficient performance-energy trade-off. Different from existing works, `BSR` reclaims slack in both directions using both `ABFT-OC` and DVFS to save more energy and enable performance improvement.
- We implement our `BSR` on three key one-sided matrix decompositions: Cholesky, LU, and QR. We evaluate our implementation on a modern CPU-GPU heterogeneous system with Nvidia GPU. Experiments show that `BSR` is able to save up to 11.7% more energy compared with the current best energy saving optimization approach with no performance degradation and up to 14.1% $Energy \times Delay^2$ reduction. Also, `BSR` enables the Pareto efficient performance-energy trade-off, which is able to provide up to 1.43× performance improvement without costing extra energy.

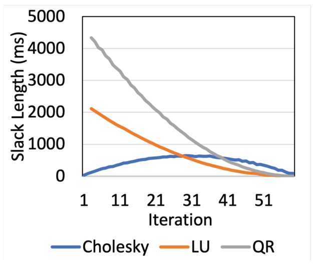


(**a**) Blocked algorithm

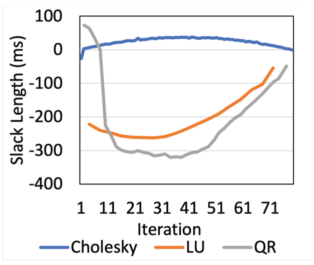


(**b**) State-of-the-art design on CPU-GPU heterogeneous systems

Figure 1. One iteration of LU decomposition



(**a**) Double precision



(**b**) Single precision

Figure 2. Slacks occur when decomposing a 30720 × 30720 matrix on our heterogeneous system. Block size is optimized for performance. Positive values represent slacks on the CPU side and negative values represent slacks on the GPU side.

2 Related Works and Problem Statement

In this section, we first introduce the design of state-of-the-art matrix decomposition on CPU-GPU heterogeneous systems and we focus on discussing their key computing characteristics. Then, we review how existing works leverage such computing characteristics to optimize for energy efficiency. Finally, we formulate our research problem and challenges.

2.1 State-of-the-art matrix decompositions

The state-of-the-art matrix decompositions for CPU-GPU heterogeneous systems use the blocked version matrix decomposition algorithms. Blocks, logically divided sub-matrices, form *Panel* and *Trailing Matrix*. The decomposition process begins from the up left corner of the matrix and moves towards the down right corner iteratively. An illustration of one iteration of the LU decomposition is shown in **Figure 1(a)**. Each iteration includes three major operations: 1 *Panel decomposition* (PD): $L \cdot 1 \times U11 \leftarrow A \cdot 1$; 2 *Panel update*

(PU): $U12 \leftarrow (L11)^{-1} \times A12$; and ③ *Trailing matrix update* (TMU): $A'22 \leftarrow A22 - L21 \times U12$. Cholesky, LU, and QR decomposition all share similar three operations. On CPU-GPU heterogeneous systems, the three operations are assigned to different processors based on their characteristics. PD is assigned to the CPUs since it is highly sequential. PU and TMU are assigned to the GPUs as they are high parallelizable. As illustrated in **Figure 1(b)**, to overlap the computation on CPUs and GPUs, a look-ahead optimization [26] is used that allows the partial PU and TMU to be done first (i.e., PU' and TMU'), so that the PD of the next iteration can be done with the rest of PU and TMU concurrently. Depending on the computational power of the CPU/GPU and the amount of workload assigned during decomposition, those concurrent tasks may finish at different times, which leads to idle computing cycles on the CPU or GPU. The idle is called *slack*. **Figure 2** show how slack length can change during Cholesky, LU, and QR decompositions on our test platform.

2.2 Existing slack-based energy saving

Matrix decompositions have been designed to maximize their usage on highly optimized BLAS-3 GPU kernels, so their energy efficiency is inherently high, which leaves limited room for further optimization. As for now, the most effective class of energy-saving optimizations for matrix decompositions on CPU-GPU heterogeneous systems is DVFS-based approaches, which aim to exploit different energy-saving techniques when there are slacks.

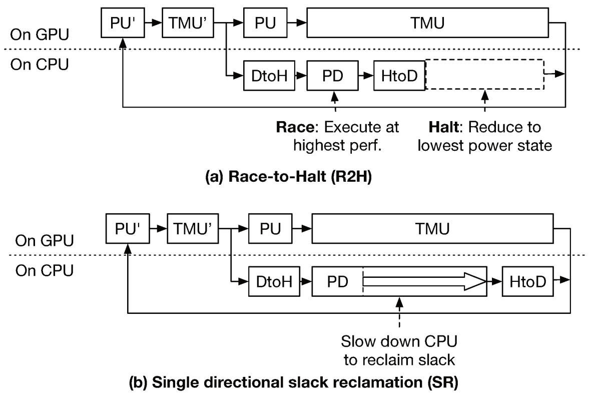

On GPU
On CPU
PU'
TMU'
PU
TMU
DtoH
PD
HtoD
Race: Execute at highest perf.
Halt: Reduce to lowest power state
(a) Race-to-Halt (R2H)
On GPU
On CPU
PU'
TMU'
PU
TMU
DtoH
PD
HtoD
Slow down CPU to reclaim slack
(b) Single directional slack reclamation (SR)

Figure 3. Existing slack-based energy saving

There are two main strategies for optimizing energy costs: *Race-to-Halt* (`R2H`) [22, 37, 39] and *Slack Reclamation* (`SR`) [7]. As shown in **Figure 3**, the main idea of R2H is to timely reduce clock frequency to the minimum as soon as the tasks on the non-critical path finish. The processor maintains its minimum clock frequency during the slack to save energy. This strategy is usually implemented by the hardware or the operating system leveraging their workload monitoring capabilities. `SR` saves energy by slowing down the tasks on the non-critical path. The reason this strategy can save energy is due to the relation between the dynamic power of the processor and its clock frequency $P_{dynamic} \propto f^{2.4}$ [17]. Theoretically, `SR` is able to save more energy compared with `R2H` [7]. Since the processor's clock frequency need to be adjusted before the execution of each task and the length of slack can change as shown in **Figure 2**, some form of computation pattern prediction is necessary. In the start-of-the-art `SR` [7], the authors propose to predict computation patterns leveraging algorithmic knowledge in matrix decompositions.

2.3 Motivation of further improving energy saving

Despite a lot of research efforts have been made to improve the energy saving of matrix decomposition on CPU-GPU heterogeneous systems, it is still desirable to further improve their energy saving since matrix decompositions are heavily used in many scientific applications. Thus improving the energy saving of matrix decomposition can lead to more energy-efficient scientific computing.

2.4 Challenges of further improving energy saving

2.4.1 Performance degradation. DVFS is designed to enable performance-energy trade-off while maintaining processor reliability. So, existing DVFS-based energy-saving techniques can only be applied to tasks on the non-critical path to avoid negatively impacting the overall performance. This has already been extensively exploited by existing works. To save even more energy, the only other choice is to apply DVFS-based energy-saving techniques to tasks on the critical path, however, this will inevitably lead to performance degradation to the overall decomposition since modern CPU and GPU processors tend to have better energy efficiency when running at lower clock frequencies.

2.4.2 Reliability degradation. Other approaches such as processor undervolting can also be used to reduce the energy cost of computation. Since it works by decreasing the core supply voltage while maintaining the same clock frequencies, it can save energy without performance degradation. However, they can decrease system reliability [2, 51, 52]. Such reliability degradation can be manifested as hard errors (e.g., process or system crash) or SDCs (e.g., incorrect calculation, bit-flips in memory cells), which can seriously decrease the reliability of matrix decomposition. Although ABFT has been used with undervolting in [2, 51] to improve the energy efficiency of matrix-matrix multiplications and ensure computing correctness, applying existing ABFT techniques can still bring considerable performance overhead. This overhead can be especially high for matrix decompositions since the iterative computing fashion is prone to error propagation, which needs the strongest variant of ABFT, *full checksum ABFT* [4], to provide sufficient protection.

2.5 Research questions

In this work, we try to answer the following research questions:

RQ:1 How to further improve energy saving of matrix decompositions on CPU-GPU heterogeneous system beyond the state-of-the-art works?
RQ:2 How to maximize energy saving for matrix decomposition while maintaining both performance and reliability at the same time?
RQ:3 How to enable performance-energy trade-off in matrix decomposition?

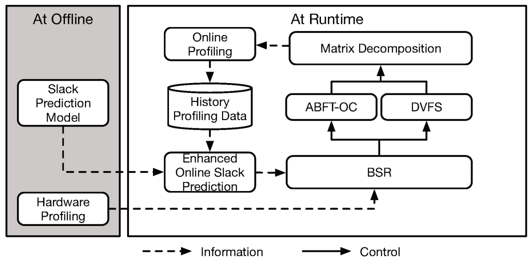


Figure 4. Overview of our energy-saving matrix decomposition framework

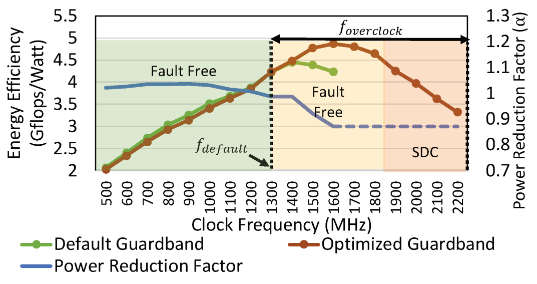
3 Design of Energy-Saving Matrix Decomposition

In this work, we propose to build a matrix decomposition framework that maximizes energy saving while maintaining both performance and reliability at the same time. **Figure 4** shows the overview of our framework. We first focus on enabling reliable computation when overclocking by coupling ABFT with overclocking - `ABFT-OC`. To reduce fault tolerance overhead, we further propose a lightweight adaptive-ABFT technique that automatically adjusts its fault tolerance strength according to the error rate. Next, based on `ABFT-OC`, we propose a novel slack-based energy saving optimization framework - `BSR`, which aims to exploit slack, to save energy and enable flexible Pareto efficient performance-energy trade-off. Different from existing works, `BSR` reclaims slack in both directions using both `ABFT-OC` and DVFS to save more energy and enable performance improvement.

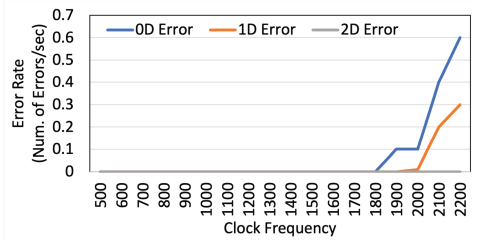
3.1 Adaptive Algorithm-Based Fault Tolerance Protected Overclocking (ABFT-OC)

To design a technique that maximize energy saving for matrix decompositions, we seek hardware energy optimization techniques beyond DVFS. DVFS has been extensively used for energy saving by both hardware and applications. It optimizes energy efficiency by lowering the core voltage (V_{dd}) with the decrease of clock frequency for reducing energy consumption. However, lowering frequency can inevitably cause performance degradation. Processor voltage guardband optimization largely mitigates this issue by allowing lowering of the core voltage without decreasing clock frequency or overclocking without violating the hardware power limit.

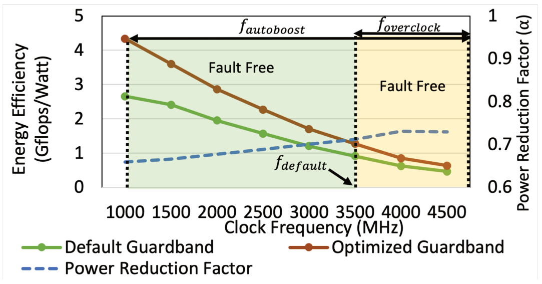
3.1.1 Voltage guardband optimization for overclocking. In this work, we define *overclocking* as the processor state



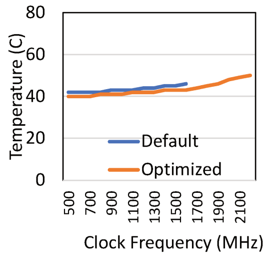
(a) GPU energy efficiency



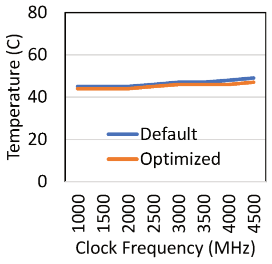
(b) GPU SDC error rate



(c) CPU energy efficiency



(d) Maximum sustained GPU core temperature



(e) Maximum sustained CPU core temperature

Figure 5. Profiling results of our testing CPU and GPU

where it sustains at a higher-than-default clock frequencies. **Figure 5** (a) shows the achievable overclocking frequency range and their energy efficiency of our test GPU at different clock frequencies after we apply voltage guardband optimization. Please note unlike previous works that were based on Windows-based GPU driver [27, 29, 51] where the core voltage can be directly adjusted and monitored, the Linux-based GPU driver does not allow us to directly control and monitor the GPU core voltage. Even though we find that optimizing the voltage guardband of GPU on Linux is still achievable through the clock offset command of the NVML API on

Linux-based GPU driver. We omit the details due to the page limit. CPU undervolting can be directly achieved on the Linux system. We set the offset of the CPU core voltage using a third-party tool `intel-undervolt`. **Figure 5** (c) shows the CPU energy efficiency before and after we set the optimized voltage guardband. Please note unlike our testing GPU, our testing CPU can achieve overclocking with the default guardband, but an optimized guardband can help us achieve higher energy efficiency.

Finding the optimized guardband is done by gradually lowering specific power settings of CPU/GPU to the point where energy efficiency is maximized without process or OS crash. The whole process can be done in less than 30 minutes and it only needs to be done once during software installation. As optimized guardband can be workload-dependent, we specifically use the workload in matrix decomposition i.e., TMU on GPU and PD on CPU to find optimized guardband. Also, as shown in **Figure 5** (b), we observe that setting to extreme high clock frequencies for the GPU can weaken the reliability of computation e.g., SDCs. So, we propose to incorporate fault tolerance with overclocking by designing `ABFT-OC`.

3.1.2 Design of `ABFT-OC`. Reliable computation is the foundation of our optimized matrix decomposition. As overclocking achieved through the use of optimized guardband can lead to SDCs, we propose to use ABFT [4–6, 10–12, 14, 19, 20, 32, 33, 40, 41, 47–50] to handle SDCs during matrix decompositions. Since the processor power state is under control and the corresponding SDC error rate is known, SDC error rate is predictable during matrix decompositions. So we propose the first ABFT that can adjust its fault tolerance strength and overhead at runtime based on the predicted error rate to minimize fault tolerance overhead and ensure correctness. SDC refers to the kind of error that only causes incorrect calculation results without process or system crash. When using our optimized guardband, the SDC is caused by insufficient core voltage supply when at high clock frequencies. The rate of SDC can increase as we increase the clock frequency when we apply a optimized guardband at the same time.

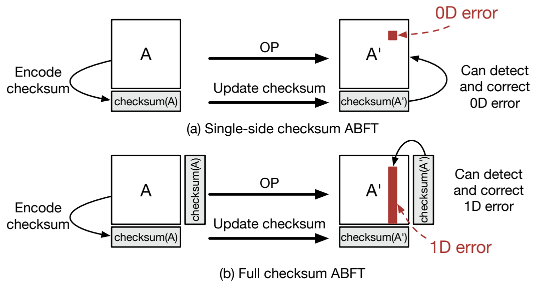


Figure 6. ABFT checksum for detecting and correcting SDCs in matrix operations

Depending on where the hardware fault occurs, it may be manifested as different kinds of SDC. For example, calculation error is usually caused by faults in the logic part of ALU or FPU. Memory storage error is usually caused by faults (e.g., bit flips) in the storage cells of DRAM, cache, or registers. For matrix operations, matrix elements can be repeatedly accessed to obtain final results. If an element whose value is corrupted gets repeatedly referenced, it may cause error propagation. Depending on the cause of the error and the computation pattern (i.e., how data is used/reused) of a matrix operation, the error pattern can be different. The degrees of error propagation [4] can be classified as: 0D, 1D, and 2D. **0D**: a single standalone error with no error propagation; **1D**: an error propagates to entire/part of one row/column; **2D**: an error propagates beyond one row/column. So, we distinguish different degrees of error propagation in **Figure 5**.

Table 1. Theoretical estimation on ABFT fault coverage (FC) on the TMU operation of the 5^{th}, 10^{th}, and 15^{th} iteration of LU decomposition if we apply different clock frequencies.

Iter.	ABFT	1800MHz	1900MHz	2000MHz	2100MHz	2200MHz
5^{th}	Single	Fault-free	Full Coverage	99.86%	97.51%	96.45%
	Full	Fault-free	Full Coverage	Full Coverage	Full Coverage	Full Coverage
10^{th}	Single	Fault-free	Full Coverage	99.94%	98.92%	98.46%
	Full	Fault-free	Full Coverage	Full Coverage	Full Coverage	Full Coverage
15^{th}	Single	Fault-free	Full Coverage	99.98%	99.76%	99.65%
	Full	Fault-free	Full Coverage	Full Coverage	Full Coverage	Full Coverage

Algorithm 1: Adaptive-ABFT strategy

```
Function ABFT-OC():
    In  : Desired ABFT fault coverage $FC_{desired}$
    In  : Desired GPU clock freq. $F^{GPU}_{desired}$
    In  : Default GPU clock freq. $F^{GPU}_{BASE}$
    In  : Predicted operation execution time $T'^{GPU}$
    $SingleABFTCheck \leftarrow FALSE$
    $FullABFTCheck \leftarrow FALSE$
    while ($\lambda_{F^{GPU}_{desired},0D} > 0 \,||\, \lambda_{F^{GPU}_{desired},1D} > 0 \,||\, \lambda_{F^{GPU}_{desired},2D} > 0$) && !$SingleABFTCheck$ && !$FullABFTCheck$ do
        $T^{GPU}_{projected} = T'^{GPU} \times \frac{F^{GPU}_{desired}}{F^{GPU}_{BASE}}$
        if $FC_{single}(F^{GPU}_{desired}, T^{GPU}_{projected}) \geqslant FC_{desired}$ then
            $SingleABFTCheck = TRUE$
        else if $FC_{full}(F^{GPU}_{desired}, T^{GPU}_{projected}) \geqslant FC_{desired}$ then
            $FullABFTCheck = TRUE$
        else
            $F^{GPU}_{desired} = F^{GPU}_{desired} - 100MHz$
        end
    end
return $F^{GPU}_{desired}$, $SingleABFTCheck$, $FullABFTCheck$
```

ABFT is based on the idea that if we encode a certain amount of matrix information in checksums before a matrix operation and apply the same matrix operation to checksums, the checksum relation would still hold for the resulting matrix. By verifying the checksum relations after the operation, we can detect and correct errors in the result matrix. Depending on how much information is encoded in checksums, the fault tolerance strength is different. As shown in **Figure 6**, there are two commonly schemes for checksum encoding: 1 Single side checksum encodes matrices along either rows or columns. Since it only encodes the matrix in one dimension, it brings relative lower overhead. However, it can only efficiently tolerate 0D error pattern. 2 Full checksum encodes matrices along both rows and column at the same time. Since it encodes matrices in both dimensions, it brings stronger protection i.e., both 0D and 1D error patterns. However, it also brings higher fault tolerance overhead.

Given that the fault tolerance strength is limited, we must determine suitable ABFT protection according to the error rate and limit the clock frequency range to ensure all errors can be detected and corrected with a high probability. Otherwise, undetected or uncorrected errors would cause serious error propagation later, which requires recovery with high overhead. In this work, we find that it is useful to estimate the probability that a certain kind of ABFT can detect and correct all errors given different error rates at different overclocking frequencies. In order to do that, we first define an error rate function R given clock frequency derived from our profiling results in **Figure 5**: $\lambda_{f,ErrType} = R(f, ErrType)$ where λ is the error rate of a certain error type ($ErrType$). The error type can be 0D, 1D, or 2D. f is the processor clock frequency. Assuming the rate is constant for a given clock frequency, we treat the distribution of probability errors that occur during a period of time as the Poisson distribution. So, the probability of having k errors in a certain type during a period of time T can be estimated using the Poisson distribution function: $p = \frac{e^{-\lambda_{f,ErrType}T}(\lambda_{f,ErrType}T)^i}{i!}$. Both single-side and full checksum encode the matrix for each matrix block individually. They cannot tolerate more than one fault strike to a matrix block during one error detection interval (i.e., one iteration of matrix decomposition). Assuming the matrix is of size n with matrix block size b, single-side checksum ABFT can tolerate up to $S = \frac{n}{b} \times \frac{n}{b}$ 0D errors, as long as two 0D errors do not strike the same matrix block within one iteration of matrix decomposition. Full checksum ABFT can tolerate up to S 0D and 1D errors combined, as long as two 0D/1D errors do not strike the same matrix block within one iteration of matrix decomposition. Assuming error occurs randomly and uniformly in time and space, we provide the theoretical estimation on the probability that ABFT can detect and correct all errors in one detection interval (i.e. *Fault Coverage (FC)*).

$$FC_{single}(f,T) =$$
$$\left(\sum_{k=0}^{S} \frac{e^{-\lambda_{f,0D}T}(\lambda_{f,0D}T)^k}{k!} \prod_{i=0}^{k} \frac{S-i}{S}\right) e^{-\lambda_{f,1D}T} e^{-\lambda_{f,2D}T}$$
$$FC_{full}(f,T) =$$
$$\left(\sum_{k=0}^{S} \sum_{j=0}^{S-k} \frac{e^{-\lambda_{f,0D}T}(\lambda_{f,0D}T)^k}{k!} \frac{e^{-\lambda_{f,1D}T}(\lambda_{f,1D}T)^j}{j!} \prod_{i=0}^{k+j} \frac{S-i}{S}\right) e^{-\lambda_{f,2D}T}$$

Table 1 show the example estimation results based on different GPU overclocking frequencies and the execution time of the TMU operation in three selected iterations of the LU decomposition. We define $FC > 99.9999\%$ as *Full Coverage*. Having the capability of fault coverage estimation, we propose an adaptive-ABFT scheme. Unlike existing ABFT works, which enable ABFT during the entire matrix decomposition process, our adaptive-ABFT only enables ABFT error detection and correction when the error rate is above 0. **Algorithm 1** shows the adaptive-ABFT strategy. We first check the error rate function in Line 4. If the rate of any kind of error is above zero, we check if applying ABFT can provide enough fault coverage (Line 5 - 9). We prioritize single-side ABFT over full ABFT to lower fault tolerance overhead. If none of the ABFT schemes can provide enough fault coverage, we progressively lower the GPU clock frequency (Line 11) until enough fault coverage is provided. Finally, we return the adjusted clock frequency together with flags indicating if we need to do a single or full ABFT check. Please note ABFT-OC would also work for CPU. We exclusively apply it to GPU in our algorithm since SDCs only occur to the GPU on our test system.

3.2 Bi-directional slack reclamation (BSR)

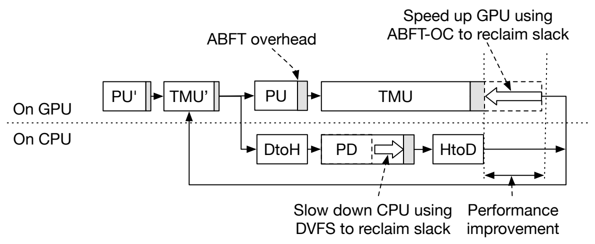


Figure 7. Bi-directional slack reclamation (BSR)

The current best energy-saving approach, single directional slack reclamation (SR) [7], saves energy by slowing down tasks on the non-critical paths via DVFS. This work proposes a novel Bi-directional slack reclamation (BSR) energy-saving technique that reclaims slacks in two directions at the same time using both ABFT-OC and DVFS. Specifically, BSR reclaims slacks by simultaneously slowing down tasks on the non-critical path using DVFS and speeding up tasks on the critical path using ABFT-OC. An illustration of BSR is shown in **Figure 7**. Compared with SR, BSR brings three major advantages: 1 potential higher energy saving through both DVFS and ABFT-OC at the same time; 2 performance improvement in addition to energy saving optimization; 3 enabling performance-energy consumption trade-off.

3.2.1 Enhanced Algorithmic-based Slack Prediction.

Slack prediction is critical for making correct power status adjustments so that energy saving can be maximized. As BSR enables more opportunities for slack reclamation, it is more critical for it to make accurate slack predictions. The state-of-the-art algorithmic slack prediction was first proposed by [7].

Table 2. Ratios of time complexity of PD, PU, TMU, transfer size, and ABFT-related operations between k^{th} and $k+1^{th}$ iteration. n and b are the total size and the block size of the input matrix respectively. PU of Cholesky and QR are omitted since they do not affect the slack

Operation	Computation & Checksum Update	Data Transfer	Checksum Verification
PD-Cho.	1	1	1
TMU-Cho.	$(1+k)(1-\frac{b}{n-kb-b})$	N/A	$1-\frac{b}{n-kb-b}$
PD-LU	$1-\frac{6b}{3n-(3k-1)b}$	$1-\frac{1}{n-kb}$	$1-\frac{1}{n-kb}$
PU-LU	$1-\frac{b}{n-kb-b}$	N/A	$1-\frac{b}{n-kb-b}$
TMU-LU	$1-\frac{2b}{n-kb}$	N/A	$1-\frac{2b}{n-kb}$
PD-QR	$1-\frac{b}{6n-(6k+1)b}$	$1-\frac{b}{n-kb-b}$	$1-\frac{b}{n-kb-b}$
TMU-QR	$1-\frac{b}{n-kb-b}-\frac{b}{n-kb+b}+\frac{b^2}{(n-kb-b)(n-kb+b)}$	N/A	$1-\frac{b}{n-kb-b}-\frac{b}{n-kb+b}+\frac{b^2}{(n-kb-b)(n-kb+b)}$

It mainly works by profiling the tasks in the 1st iteration of decomposition and using the profiled time together with ratios of computational time complexity between k^{th} iteration and the 1st to predict the execution time of tasks in the k^{th} iteration of decomposition. By leveraging algorithmic knowledge and profiling results, algorithmic slack prediction can achieve much higher prediction accuracy compared with statistical-learning-based approaches and hardware-based approaches.

However, we find that the accuracy of current algorithmic slack prediction highly relies on the profiling accuracy of the 1st iteration and the assumption that computational efficiency stays constant across different iterations on a given processor. As the measurement of the 1st iteration can be inaccurate (e.g., when it is short) and the computational efficiency of tasks can also change considerably throughout the decomposition process, all these inaccuracies can accumulate and cause large prediction errors in the latter part of the decomposition process, which lead to wrong slack reclamation decisions.

In BSR, we propose an enhanced algorithmic-based slack prediction that greatly improves slack prediction accuracy. The enhanced algorithmic-based slack prediction rely on the profiled execution time of the p last neighbor iterations to predict the execution time of the current iteration to reduce the negative impacts bring by inaccurate profiling and changes in computational efficiency since tasks in neighbor iterations tend to have similar input sizes and thus similar computational efficiencies. Since a closer neighbor has a more accurate estimation of computational efficiency, we apply different weights to different profiling results in our enhanced algorithmic-based slack prediction. Specifically, the execution time of a task in k^{th} iteration ($T_k^{'OP}$) is predicted as:

$$T_k^{'OP} = w_1 r_{k-1,k}^{OP} T_{k-1}^{OP} + w_2 r_{k-2,k}^{OP} T_{k-21}^{OP} + ... + w_p r_{k-p,k}^{OP} T_{k-p}^{OP}$$

where $r_{j,k}^{OP}$ is the ratio of theoretical time complexity of OP between j^{th} and k^{th} iteration, which can be calculated based on the algorithm time complexity and relative change of the input sizes of OP. **Table 2** shows the ratios of key components of matrix decompositions. We omit the calculation process due to the page limit. T_{k-i}^{OP} is the actual profiled execution time of OP of the i^{th} last neighbor. w_1 is the weight we applied to the i^{th} last neighbor. Through empirical study, we find that $p=4$ and $w_1=\frac{1}{2}, w_2=\frac{1}{4}, w_3=\frac{1}{8}, w_4=\frac{1}{8}$ can help provide enough prediction accuracy for energy saving. When ABFT is applied, the slack of the k^{th} iteration is predicted as:

$$\begin{aligned} slack_k = {} & T_k^{'TMU} + T_k^{'TMU\ checksum\ update} + T_k^{'TMU\ checksum\ verf} \\ & T_k^{'PU} + T_k^{'PU\ checksum\ update} + T_k^{'PU\ checksum\ verf} \\ & - T_k^{'PD} - T_k^{'PD\ checksum\ update} - T_k^{'PD\ checksum\ verf} \\ & - T_k^{'Data\ Transfer} - T_k^{'Transfer\ checksum} \end{aligned}$$

3.2.2 Bi-directional slack reclamation strategies.

Compared with SR, BSR offers more flexibility by reclaiming slacks from both directions, so the fractions of slacks that are reclaimed by the two tasks are adjustable, which in turn controls the performance-energy efficiency trade-off. So, we define *reclamation ratio* (r) to be the fraction of the slack we try to reclaim by speeding up the task on the critical path and $1-r$ to be the fraction we try to reclaim by slowing down the task on the non-critical path. **Algorithm 2** shows our BSR algorithm that makes decisions at the beginning of each matrix decomposition iteration. The execution time of tasks and slack are predicted in Line 3 - 4 using our enhanced algorithmic-based slack prediction. Given reclamation ratio r, we calculate the desired execution time of tasks on CPU and GPU in Line 5 - 11. We also consider the overhead of DVFS operations in our calculation to minimize the impact on performance. Line 12 - 15 calculate the desired CPU/GPU clock frequencies and limit them within the available frequency range. Line 16 - 17 calculates the projected execution time if we apply the desired frequencies. Note that the projected time may be different from the desired time since desired frequencies could be out of the available range. Finally, we make decisions on whether or not we adjust CPU/GPU clock frequencies in Line 18 - 22. If the projected time suggests that it can make a negative impact on the performance, it will skip frequency adjustment for this iteration i.e., setting AdjustCPU/GPU to FALSE. Note that this does not mean we do not reclaim slack of this iteration. Since we still keep the adjusted CPU/GPU frequencies from the last iteration, the partial of slack can still be reclaimed. This strategy ensures we reclaim most of the slacks while minimizing performance impact. Line 23 invokes our adaptive-ABFT strategy for overclocking. Finally, we return the final decisions regarding CPU/GPU clock frequency adjustments and ABFT protection strength for the current iteration.

Algorithm 2: `BSR` strategy

```
1  Function BSR():
       In  : reclamation ratio $r$
       In  : iteration $k$
       In  : GPU DVFS latency $L^{GPU}$
       In  : CPU DVFS latency $L^{CPU}$
       In  : Desired ABFT fault coverage $FC_{desired}$
2      Apply optimized guardband for both CPU and GPU
3      $T'^{CPU}, T'^{GPU}, T'^{DataTransfer} \leftarrow$ EnhancedAlgorithmicPrediction($k$)
4      $slack_k \leftarrow = T'^{GPU} - T'^{CPU} - T'^{DataTransfer}$
5      if $slack_k > 0$ then
6          $T^{GPU}_{desired} \leftarrow T'^{GPU} - (slack_k \times r) - L^{GPU}$
7          $T^{CPU}_{desired} \leftarrow T^{GPU}_{desired} - L^{CPU} - T'^{DataTransfer}$
8      else
9          $T^{CPU}_{desired} \leftarrow T'^{CPU} - (slack_k \times r) - L^{CPU}$
10         $T^{GPU}_{desired} \leftarrow T^{CPU}_{desired} - L^{GPU} + T'^{DataTransfer}$
11     end
12     $F^{GPU}_{desired} \leftarrow Roundup(F^{GPU}_{BASE} \times \frac{T'^{GPU}}{T^{GPU}_{desired}}, 100Mhz)$
13     $F^{CPU}_{desired} \leftarrow Roundup(F^{CPU}_{BASE} \times \frac{T'^{CPU}}{T^{CPU}_{desired}}, 100Mhz)$
14     $F^{GPU}_{desired} = LimitToRange(F^{GPU}_{min}, F^{GPU}_{max})$
15     $F^{CPU}_{desired} = LimitToRange(F^{CPU}_{min}, F^{CPU}_{max})$
16     $T^{GPU}_{projected} = T'^{GPU} \times \frac{F^{GPU}_{desired}}{F^{GPU}_{BASE}}$
17     $T^{CPU}_{projected} = T'^{CPU} \times \frac{F^{CPU}_{desired}}{F^{CPU}_{BASE}}$
18     $T_{max} = max(T'^{GPU}, T'^{CPU} + T'^{DataTransfer})$
19     if $T^{GPU}_{projected} > T_{max}$ then $AdjustGPU \leftarrow FALSE$;
20     else $AdjustGPU \leftarrow TRUE$;
21     if $T^{CPU}_{projected} > T_{max}$ then $AdjustCPU \leftarrow FALSE$;
22     else $AdjustCPU \leftarrow TRUE$;
23     $F^{GPU}_{desired}, SingleABFTCheck, FullABFTCheck \leftarrow$
           ABFT-OC($FC_{desired}, F^{GPU}_{desired}, F^{GPU}_{BASE}, T'^{GPU}$)
24 return $AdjustCPU, AdjustGPU, F^{CPU}_{desired}, F^{GPU}_{desired},$
       $SingleABFTCheck, FullABFTCheck$
```

3.2.3 Theoretical performance improvement and energy saving analysis.

Next, we provide a theoretical analysis of performance improvement and energy saving. With losing generality, we assume that the slack on the CPU in the following discussion for simplification. The performance improvement mainly comes from speeding up the tasks on the critical path. So, the performance improvement of iteration k can be simply calculated as: $\Delta T = T^{old}_k - T^{new}_k = T^{GPU}_k - (T^{GPU}_k - slack_k \times r) = slack_k \times r$. This suggests that higher r leads to higher performance. As for energy consumption, the theoretical amount of energy saving on the CPU when adopting `BSR` with reclamation ratio r in the iteration k can be estimated as:

$$\Delta E^{CPU}_k = \Delta E^{CPU\_dynamic}_k + \Delta E^{CPU\_static}_k$$

$$\Delta E^{CPU\_dynamic}_k = E^{CPU\_dynamic\_old}_k - E^{CPU\_dynamic\_new}_k =$$
$$d^{CPU} P^{CPU}_{total} T^{CPU}_k -$$
$$\alpha^{CPU} \left(\frac{f^{CPU\_new}}{f^{CPU\_old}} \right)^{2.4} d^{CPU} P^{CPU}_{total} (T^{CPU}_k + slack_k (1-r)) =$$
$$d^{CPU} P^{CPU}_{total} T^{CPU}_k -$$
$$\alpha^{CPU} \left(\frac{T^{CPU}_k}{T^{CPU}_k + slack_k (1-r)} \right)^{2.4} d^{CPU} P^{CPU}_{total}$$

$$(T^{CPU}_k + slack_k (1-r)) =$$
$$\left(1 - \alpha^{CPU} \frac{(T^{CPU}_k)^{1.4}}{(T^{CPU}_k + slack_k \times (1-r))^{1.4}} \right) d^{CPU} P^{CPU} T^{CPU}_k$$
$$\Delta E^{CPU\_static}_k = (T^{CPU}_k - \alpha^{CPU} (T^{CPU}_k + slack_k (1-r)))$$
$$(1 - d^{CPU}) P^{CPU}_{total}$$

Similarly, we can estimate the energy saving on GPUs as follows:

$$\Delta E^{GPU}_k = \left(1 - \alpha^{GPU} \frac{(T^{GPU}_k)^{1.4}}{(T^{GPU}_k - slack \times r)^{1.4}} \right) d^{GPU} P^{GPU}_{total} T^{GPU}_k +$$
$$(T^{GPU}_k - \alpha^{GPU} (T^{GPU}_k - slack_k \times r))(1 - d^{GPU}) P^{GPU}_{total}$$

Where $\alpha^{CPU/GPU}$ are total power reduction factors when we use optimized guardband of CPU/GPU. We measure that in our hardware profiling work **Figure 5**. For clock frequencies out of the default range, we use constant values of the last measured value to estimate (dashed line). $T^{CPU/GPU}_k$ are the original task execution time of CPU/GPU. $P^{CPU/GPU}_{total}$ are the total power of CPU/GPU at the default guardband and clock frequencies. $d^{CPU/GPU}$ are the ratios of the CPU/GPU dynamic power in the total power consumption. The change of CPU/GPU dynamic power is estimated using: $P_{dynamic} \propto f^{2.4}$ [17]. When the critical path is on the GPU, it is for sure we can save energy on the CPU. However, whether or not we can save energy on the GPU depends on α^{GPU} and r. Assuming power reduction factor α^{GPU} is fixed and minimized by applying optimized processor guardband, then the reclamation ratio r controls the trade-off between performance improvement and energy consumption. Higher r leads to higher performance but less energy saving, and vice versa. The highest energy saving can be achieved with $r_{max\_energy} = 0$ without performance improvement. The max r that achieves maximum without impacting energy efficiency is hard to be solved directly. So, we use a numerical approach to solve for r. By solving $\Delta E^{CPU}_k + \Delta E^{GPU}_k = 0$ using Newton's method, we are able to get estimated solutions. For example, for decomposition with input 30730×30720, the averaged reclamation ratios across all iterations are 0.28 for Cholesky, 0.26 for LU, and

0.31 for QR, which approximately matches our experimental results in **Figure** 11.

4 Experimental Evaluation

4.1 Evaluation Methodology

We compare BSR with two state-of-the-art energy-saving approaches R2H and SR together with the original design in the MAGMA library.

- Original: The original matrix decompositions in the state-of-the-art MAGMA library. We keep the CPU/GPU clock frequency fixed at the default (autoboost disabled).
- R2H: The original matrix decompositions in the state-of-the-art MAGMA library with CPU/GPU autoboost feature enabled. The processor clock frequency is dynamically set according to the workload.
- SR: The state-of-the-art energy efficient matrix decompositions using single directional slack reclamation [7].
- BSR: Our proposed matrix decomposition with BSR energy efficiency optimization and ABFT-OC. Clock frequencies can reach greater ranch where SDCs can occur but are correctable by ABFT.

All the above versions are implemented for Cholesky, LU, and QR decomposition for double precision inputs with block size tuned for performance.

Table 3. Hardware/System Configuration for Experiments.

Processor	Intel Core i7-9700K	NVIDIA RTX 2080 Ti
Base Clock	3.5(↑by0.1)GHz	1.3(↑by 0.1)GHz
Overclocking	3.6-4.5(↑by0.1)GHz	1.4-2.2(↑by 0.1)GHz
Memory	32 GB RAM	12 GB RAM
Default guardband	Vcore offset: 0mV	Graphics clock offset: 0
Optimized guardband	Vcore offset: -150mV	Graphics clock offset: +200

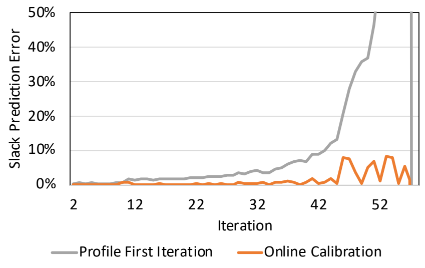


Figure 8. Slack prediction error of the LU decomposition using different approaches

4.2 Experimental Environment

All experiments are performed on a power-aware CPU-GPU server. **Table** 3 lists hardware configuration of the experimental platform and system tools used for adjusting CPU/GPU guardband/clock frequencies and for measuring the energy consumption of CPU and GPU. Limited to the capability of our test platform, we only measure the energy consumption

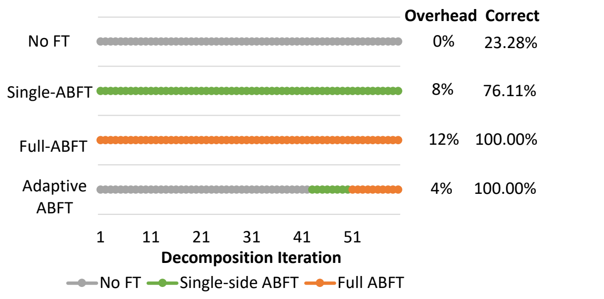


Figure 9. Comparing overhead and correctness when different ABFT scheme is applied in double precision LU decomposition with reclamation ratio $r = 0.25$

of the CPU package and GPU device. For accurate measurement of energy consumption and stable SDCs error rate at reduced guardband, we adjust the external cooling system to stabilize the CPU/GPU temperature at 45°C and 55°C respectively. From the software perspective, all matrix decomposition versions are built with GCC 7.4.0 and CUDA 11.6 with the highest optimization flags turned on. NVIDIA cuBLAS 11.1 and Intel MKL 2020 are used as linear algebra computing kernels. MKL is configured to use all CPU cores. The operating system is Ubuntu 18.04.

4.3 Evaluation Results

4.3.1 Online slack prediction accuracy comparison. Figure 8 shows the relative online prediction error using only the first iteration to predict [7] vs. our enhanced slack prediction approach proposed in this work. We can see both approaches can give less than 10% relative error for the first 2/3 of the iterations. However, since [7] only depends on the profiling result of the first iteration, the error caused by profiling and prediction will accumulate and become significant (about 11.4% on average) as the decomposition progresses. Our enhanced algorithmic slack prediction uses an online calibration approach to effectively avoid error from accumulating and reducing relative prediction error to around 4% on average.

4.3.2 ABFT overhead and correctness comparison. Figure 9 shows the computational overhead and probability of computing correctness when different ABFT schemes are applied. We use double precision LU decomposition with BSR reclamation ratio $r = 0.25$ as an example. The correctness is estimated by repeating the decomposition 100,000 times and comparing the results. We observe similar results on other types of decompositions. Due to relative short slack in the later part of decomposition, higher GPU clock frequencies are needed, which reach degrees of overclocking that can have SDC errors. If we do not apply any fault tolerance, only 23.28% of the overall matrix decomposition tests output correct results. If we apply single-side checksum ABFT, it improves the percentage of tests with correct output to 76.11% since 0D errors can be effectively detected and corrected. However, 1D error cannot be handled by single-side

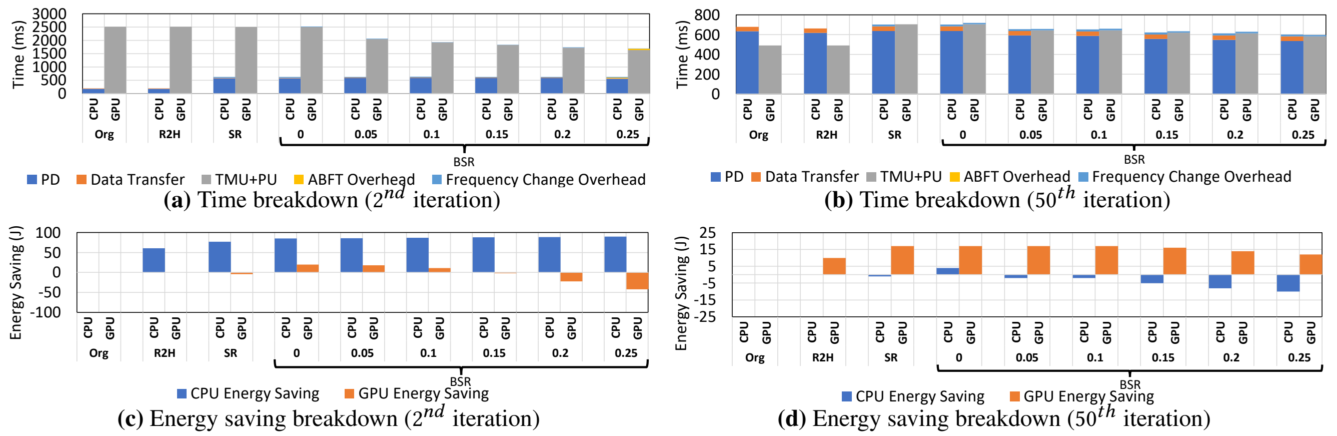


Figure 10. Time and energy saving breakdown of the 2nd and 50th iteration of the LU decomposition (Input size: 30720×30720). Energy saving is compared with the original design. Positive values represent energy saving and negative values represent extra energy costs.

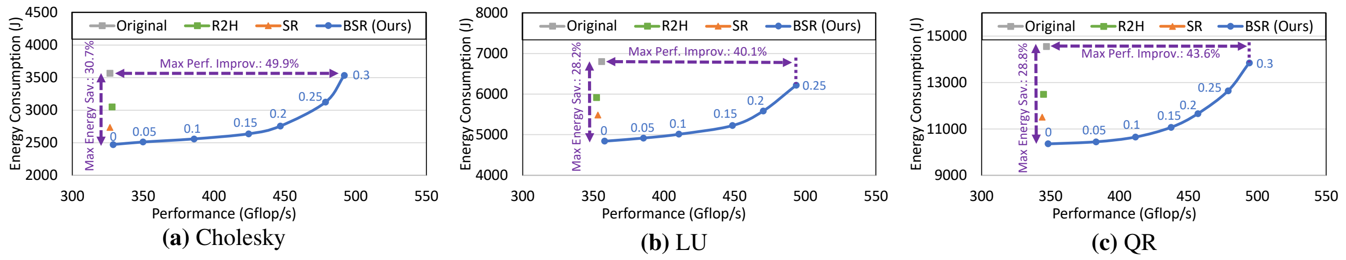


Figure 11. Pareto efficient performance-energy consumption trade-off enabled by adjusting the reclamation ratio. Input size: 30720×30720 double precision

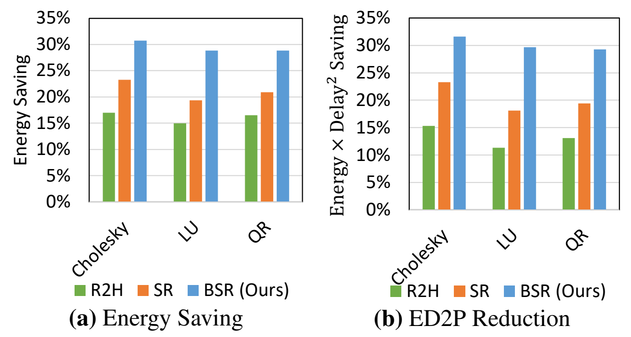


Figure 12. Overall energy saving and ED2P Reduction compared with the original design. Input size: 30720×30720.

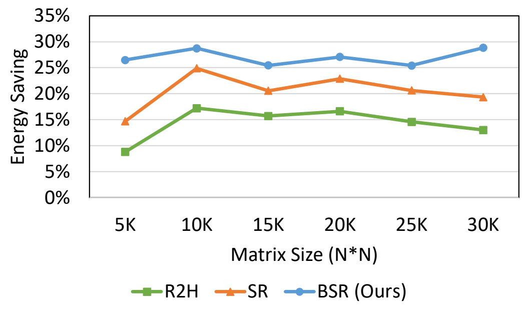


Figure 13. Overall energy saving of LU compared with the original design with different input matrix sizes

checksum ABFT. When full checksum ABFT is applied, it can ensure all decomposition tests are correct, but it also brings 12% overhead. Our adaptive-ABFT can adaptively apply necessary levels of fault tolerance to ensure high reliability and low overhead. For example, when we set the reclamation ratio $r = 0.25$, the first 41 iterations are running at fault-free clock frequencies (1700Mhz), so adaptive-ABFT completely disables ABFT for eliminating unnecessary fault tolerance overhead. For $42^{th} - 49^{th}$ iteration, the slacks need to be reduced by `BSR` using more aggressive overclocking (up to 1900Mhz), so it applies single-side checksum ABFT. Finally, it applies full checksum ABFT after 50^{th} iteration since higher clock frequencies are used (up to 2200Mhz). So, with adaptive-ABFT, we can still ensure all decomposition tests are correct with only 4% fault tolerance overhead.

4.3.3 Per iteration performance and energy comparison.

To understand how each of the different approaches affects the performance and energy efficiency of matrix decompositions, we show the profiling results of 2^{nd} and 50^{th} iteration of the LU decomposition in terms of time and energy costs breakdown in **Figure** 10. For the original version, we can see the slack occurs on the CPU side for the 2^{nd} iteration and GPU side for the 50^{th} iteration. For clarity, we refer to the case that slack is on the CPU side as C and the case that slack is on the GPU side as G in our following discussion. For `R2H`, we observe noticeable energy saving in both C and G due to reduced energy consumption on the CPU side and GPU side respectively. For SR, we see slack is fully reclaimed in G, but not fully reclaimed in C due to the limited clock frequency range on the CPU and longer slack length. For BSR, we test different reclamation ratios r and mark their values under the bars. We set r from 0 to a certain value that leads to maximum achievable performance. This maximum r is higher for C than G since GPU has greater overclocking capabilities than CPU in our system when we apply optimized guardband. We can see maximum energy saving is achieved when $r = 0$, which is consistent with our previous theoretical analysis. Maximum performance $r = 0.25$ for C and G, which are close to our theoretical estimation. When we increase r, we see an increase in energy consumption for the processor on the critical path due to the increase in clock frequency. For C, we observe a slight increase in energy-saving since the slack is long enough for the CPU to always run at the lowest clock frequency, and reducing the total execution time can save more CPU static energy. We also observe a slight decrease in energy saving in G, mainly due to the slight increases in clock frequencies. Even though it can still save energy since 1) the clock frequencies are low; 2) power reduction brings by optimized guardband. Finally, Thanks to ABFT-OC, we can exploit higher overclocking frequencies where we can achieve higher performance and energy efficiency in C.

4.3.4 Overall energy saving and energy efficiency comparison.

Next, we show the overall energy-saving capability of different approaches in **Figure 12(a)**. We evaluate all three matrix decompositions with an input size of 30720×30720. All four versions of each type of matrix decomposition produce a similar performance. To maximize energy saving the reclamation ratio of BSR is set to 0. We can see that compared with the state-of-the-art MAGMA library, our BSR is able to save energy by 30.7% for Cholesky, 28.2% for LU, and 28.8% for QR. That is $1.31\times - 1.49\times$ more energy saving compared with the current state-of-the-art SR energy saving approach and $2.03\times - 2.20\times$ more energy saving compared with R2H. In addition, we use $Energy \times Delay^2$ (ED2P) to measure the energy efficiency of matrix decompositions. As shown in **Figure 12(b)**, compared with the original design, our BSR is able to reduce ED2P by 29.3%-31.6%. Compared with R2H, BSR is able to reduce ED2P by 18.6%-20.7%. Finally, compared with SR, BSR is able to reduce ED2P by 10.8%-14.1%.

4.3.5 Overall energy saving on different input sizes.

In **Figure 13**, we show the results of applying energy-saving approaches on LU decomposition with different input sizes. Limited by the page space, we only show the results for LU decomposition. Other matrix decompositions behave similarly. We can see our BSR is able to stably save energy consumption across different input matrix sizes ranging from 5120×5120 and above. Note that it is hard to save energy on smaller matrices since they either lead to high fault tolerance overhead or small slacks that are hard to be reclaimed.

4.3.6 Overall Pareto efficient performance-energy consumption trade-off.

Finally, we show the overall Pareto efficient performance-energy consumption trade-off enabled by adjusting the reclamation ratio in BSR. As shown in **Figure 11**, by adjusting the reclamation ratio to a minimum 0, we achieve max energy saving with similar performance to the original design. In this case, compared with the original design, BSR is able to save energy by 28.2%-30.7%. Compared with R2H, BSR is able to save energy by 17.1%-18.9%. Compared with SR, BSR is able to save energy by 9.6%-11.7%. By increasing the reclamation ratio, we are able to adjust the performance or energy consumption of matrix decompositions. For example, with equal or less energy consumption, compared with the original design BSR is enable to improve the performance by 1.38×-1.51×. Also, compared with R2H, BSR is enable to improve the performance by 1.33×-1.43×. In addition, compared with SR, BSR is enable to improve the performance by 1.36×-1.43×. Finally, we see the results of BSR with different reclamation ratios form a Pareto set such that we cannot improve energy saving and performance at the same time without reliability degradation.

5 Conclusion

In this work, we focused on further improving the energy saving of matrix decompositions on CPU-GPU heterogeneous systems beyond existing state-of-the-art works. To achieve our goal, we first proposed ABFT-OC, a novel overclocking technique that is protected by ABFT to enable reliable computation for key operations in matrix decompositions when overclocking. Next, based on ABFT-OC, we proposed BSR, a novel matrix decomposition framework, that aims to maximize energy saving while maintaining performance and reliability. We evaluated BSR on three key matrix decomposition algorithms - Cholesky, LU, and QR. Experiments show that BSR is able to save up to 11.7% more energy compared with the current best energy saving optimization approach with no performance degradation and up to 14.1% ED2P reduction. Also, BSR enables the Pareto efficient performance-energy trade-off, which is able to provide up to 1.43× performance improvement without costing extra energy.

6 Acknowledgement

The fault tolerance developments in this work by Z. C. were supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through the Advanced Computing (SciDAC) program under Award Number DE-SC0022209. The energy-saving techniques in this work by L. B. was supported by the National Science Foundation under Grant Number 1907401.

References

[1] Heiko Burau, Renée Widera, Wolfgang Hönig, Guido Juckeland, Alexander Debus, Thomas Kluge, Ulrich Schramm, Tomas E Cowan, Roland Sauerbrey, and Michael Bussmann. 2010. PIConGPU: a fully relativistic particle-in-cell code for a GPU cluster. *IEEE Transactions on Plasma Science* 38, 10 (2010), 2831–2839.

[2] Aurélien Cavelan, Yves Robert, Hongyang Sun, and Frédéric Vivien. 2015. Voltage overscaling algorithms for energy-efficient workflow computations with timing errors. In *Proceedings of the 5th Workshop on Fault Tolerance for HPC at eXtreme Scale*. 27–34.

[3] Choong-Seock Chang, Seunghoe Ku, and H Weitzner. 2004. Numerical study of neoclassical plasma pedestal in a tokamak geometry. *Physics of Plasmas* 11, 5 (2004).

[4] Jieyang Chen, Hongbo Li, Sihuan Li, Xin Liang, Panruo Wu, Dingwen Tao, Kaiming Ouyang, Yuanlai Liu, Kai Zhao, Qiang Guan, et al. 2018. Fault tolerant one-sided matrix decompositions on heterogeneous systems with GPUs. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 854–865.

[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 International Conference on*.

[6] Jieyang Chen, Xin Liang, and Zizhong Chen. 2016. Online Algorithm-Based Fault Tolerance for Cholesky Decomposition on Heterogeneous Systems with GPUs. In *2016 International Parallel and Distributed Processing Symposium (IPDPS)*.

[7] Jieyang Chen, Li Tan, Panruo Wu, Dingwen Tao, Hongbo Li, Xin Liang, Sihuan Li, Rong Ge, Laxmi Bhuyan, and Zizhong Chen. 2016. GreenLA: green linear algebra software for GPU-accelerated heterogeneous computing. In *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 667–677.

[8] Jieyang Chen, Lipeng Wan, Xin Liang, Ben Whitney, Qing Liu, David Pugmire, Nicholas Thompson, Jong Youl Choi, Matthew Wolf, Todd Munson, et al. 2021. Accelerating multigrid-based hierarchical scientific data refactoring on gpus. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 859–868.

[9] Jieyang Chen, Nan Xiong, Xin Liang, Dingwen Tao, Sihuan Li, Kaiming Ouyang, Kai Zhao, Nathan DeBardeleben, Qiang Guan, and Zizhong Chen. 2019. TSM2: optimizing tall-and-skinny matrix-matrix multiplication on GPUs. In *Proceedings of the ACM International Conference on Supercomputing*. 106–116.

[10] Longxiang Chen, Dingwen Tao, Panruo Wu, and Zizhong Chen. [n.d.]. Extending Checksum-Based ABFT to Tolerate Soft Errors Online in Iterative Methods. ([n. d.]).

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

[12] Zizhong Chen. 2009. Optimal real number codes for fault tolerant matrix operations. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. 1–10.

[13] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759* (2014).

[14] Teresa Davies and Zizhong Chen. 2013. Correcting soft errors online in LU factorization. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*.

[15] Jack Dongarra, Mark Gates, Azzam Haidar, Jakub Kurzak, Piotr Luszczek, Stanimire Tomov, and Ichitaro Yamazaki. 2014. Accelerating Numerical Dense Linear Algebra Calculations with GPUs. *Numerical Computations with GPUs* (2014), 1–26.

[16] Jack J Dongarra, Hans W Meuer, Erich Strohmaier, et al. 1997. TOP500 supercomputer sites. *Supercomputer* 13 (1997), 89–111.

[17] R. Efraim, R. Ginosar, C. Weiser, and A. Mendelson. 2014. Energy aware race to halt: A down to EARtH approach for platform energy management. *IEEE Computer Architecture Letters* 13, 1 (Jan. 2014), 25–28.

[18] Olivier Guyon and Jared Males. 2017. Adaptive optics predictive control with empirical orthogonal functions (EOFs). *arXiv preprint arXiv:1707.00570* (2017).

[19] Doug Hakkarinen and Zizhong Chen. 2012. Multilevel diskless checkpointing. *IEEE Trans. Comput.* 62, 4 (2012), 772–783.

[20] Doug Hakkarinen, Panruo Wu, and Zizhong Chen. 2014. Fail-stop failure algorithm-based fault tolerance for cholesky decomposition. *IEEE Transactions on Parallel and Distributed Systems* 26, 5 (2014), 1323–1335.

[21] Keliang He, Elizabeth Martin, and Matt Zucker. 2013. Multigrid CHOMP with local smoothing. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE, 315–322.

[22] Tohru Ishihara and Hiroto Yasuura. 1998. Voltage scheduling problem for dynamically variable voltage processors. In *Proceedings of the 1998 international symposium on Low power electronics and design*. ACM, 197–202.

[23] Vahid Jalili-Marandi, Zhiyin Zhou, and Venkata Dinavahi. 2012. Large-scale transient stability simulation of electrical power systems on parallel GPUs. In *2012 IEEE Power and Energy Society General Meeting*. IEEE, 1–11.

[24] Wayne Joubert, Deborah Weighill, David Kainer, Sharlee Climer, Amy Justice, Kjiersten Fagnan, and Daniel Jacobson. 2018. Attacking the opioid epidemic: Determining the epistatic and pleiotropic genetic architectures for chronic pain and opioid addiction. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 717–730.

[25] S Ku, Choong-Seock Chang, and PH Diamond. 2009. Full-f gyrokinetic particle simulation of centrally heated global ITG turbulence from magnetic axis to edge pedestal top in a realistic tokamak geometry. *Nuclear Fusion* 49, 11 (2009).

[26] Jakub Kurzak and Jack Dongarra. 2006. Implementing linear algebra routines on multi-core processors with pipelining and a look ahead. In *International Workshop on Applied Parallel Computing*. Springer, 147–156.

[27] Jingwen Leng et al. 2016. *Guardband management in heterogeneous architectures*. Ph.D. Dissertation.

[28] Jingwen Leng, Alper Buyuktosunoglu, Ramon Bertran, Pradip Bose, and Vijay Janapa Reddi. 2015. Safe limits on voltage reduction efficiency in GPUs: a direct measurement approach. In *Microarchitecture (MICRO), 2015 48th Annual IEEE/ACM International Symposium on*. IEEE, 294–307.

[29] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M Aamodt, and Vijay Janapa Reddi. 2013. GPUWattch: Enabling energy optimizations in GPGPUs. *ACM SIGARCH Computer Architecture News* 41, 3 (2013), 487–498.

[30] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, Xu Liu, Nathan R Tallent, and Kevin J Barker. 2019. Evaluating modern gpu interconnect: Pcie, nvlink, nv-sli, nvswitch and gpudirect. *IEEE Transactions on Parallel and Distributed Systems* 31, 1 (2019), 94–110.

[31] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Xu Liu, Nathan Tallent, and Kevin Barker. 2018. Tartan: evaluating modern GPU interconnect via a multi-GPU benchmark suite. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 191–202.

[32] Sihuan Li, Hongbo Li, Xin Liang, Jieyang Chen, Elisabeth Giem, Kaiming Ouyang, Kai Zhao, Sheng Di, Franck Cappello, and Zizhong Chen. 2019. Ft-isort: Efficient fault tolerance for introsort. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–17.

[33] 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.

[34] Kenneth Moreland, Christopher Sewell, William Usher, Li-ta Lo, Jeremy Meredith, David Pugmire, James Kress, Hendrik Schroots, Kwan-Liu Ma, Hank Childs, et al. 2016. Vtk-m: Accelerating the visualization toolkit for massively threaded architectures. *IEEE computer graphics and applications* 36, 3 (2016), 48–58.

[35] Lawrence M Murray, EM Jones, and J Parslow. 2012. On collapsed state-space models and the particle marginal Metropolis-Hastings sampler. *arXiv. org* (2012).

[36] Cody Rivera, Jieyang Chen, Nan Xiong, Jing Zhang, Shuaiwen Leon Song, and Dingwen Tao. 2021. Tsm2x: High-performance tall-and-skinny matrix–matrix multiplication on gpus. *J. Parallel and Distrib. Comput.* 151 (2021), 70–85.

[37] Nikzad Babaii Rizvandi, Javid Taheri, and Albert Y Zomaya. 2011. Some observations on optimal frequency selection in DVFS-based energy consumption minimization. *J. Parallel and Distrib. Comput.* 71, 8 (2011), 1154–1164.

[38] Jingweijia Tan, Nilanjan Goswami, Tao Li, and Xin Fu. 2011. Analyzing soft-error vulnerability on GPGPU microarchitecture. In *Workload Characterization (IISWC), 2011 IEEE International Symposium on*.

[39] Li Tan and Zizhong Chen. 2015. Slow down or halt: Saving the optimal energy for scalable HPC systems. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. ACM, 241–244.

[40] Dingwen Tao, Sheng Di, Xin Liang, Zizhong Chen, and Franck Cappello. 2018. Improving performance of iterative methods by lossy checkponting. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 52–65.

[41] Dingwen Tao, Shuaiwen Leon Song, Sriram Krishnamoorthy, Panruo Wu, Xin Liang, Eddy Z. Zhang, Darren Kerbyson, and Zizhong Chen. 2016. New-Sum: A Novel Online ABFT Scheme For General Iterative Methods. In *Proceedings of the 25th International Symposium on High-Performance Parallel and Distributed Computing*.

[42] Jiannan Tian, Sheng Di, Kai Zhao, Cody Rivera, Megan Hickman Fulp, Robert Underwood, Sian Jin, Xin Liang, Jon Calhoun, Dingwen Tao, et al. 2020. Cusz: An efficient gpu-based error-bounded lossy compression framework for scientific data. *arXiv preprint arXiv:2007.09625* (2020).

[43] Jiannan Tian, Cody Rivera, Sheng Di, Jieyang Chen, Xin Liang, Dingwen Tao, and Franck Cappello. 2021. Revisiting huffman coding: Toward extreme performance on modern gpu architectures. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 881–891.

[44] Stanimire Tomov, Jack Dongarra, and Marc Baboulin. 2010. Towards dense linear algebra for hybrid GPU accelerated manycore systems. *Parallel Comput.* 36, 5-6 (June 2010), 232–240. https://doi.org/10.1016/j.parco.2009.12.005

[45] Stanimire Tomov, Rajib Nath, Hatem Ltaief, and Jack Dongarra. 2010. Dense Linear Algebra Solvers for Multicore with GPU Accelerators. In *Proc. of the IEEE IPDPS'10*. IEEE Computer Society, Atlanta, GA, 1–8. DOI: 10.1109/IPDPSW.2010.5470941.

[46] Yash Ukidave, Xiangyu Li, and David Kaeli. 2016. Mystic: Predictive scheduling for gpu based cloud servers using machine learning. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 353–362.

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

[48] Panruo Wu, Nathan DeBardeleben, Qiang Guan, Sean Blanchard, Jieyang Chen, Dingwen Tao, Xin Liang, Kaiming Ouyang, and Zizhong Chen. 2017. Silent Data Corruption Resilient Two-sided Matrix Factorizations. In *Proceedings of the 22nd Principles and Practice of Parallel Programming*.

[49] Panruo Wu, Qiang Guan, Nathan DeBardeleben, Sean Blanchard, Dingwen Tao, Xin Liang, Jieyang Chen, and Zizhong Chen. 2016. Towards Practical Algorithm Based Fault Tolerance in Dense Linear Algebra. In *Proceedings of the 25th International Symposium on High-Performance Parallel and Distributed Computing*.

[50] Erlin Yao, Jiutian Zhang, Mingyu Chen, Guangming Tan, and Ninghui Sun. 2015. Detection of soft errors in LU decomposition with partial pivoting using algorithm-based fault tolerance. *The International Journal of High Performance Computing Applications* 29, 4 (2015), 422–436.

[51] Hadi Zamani, Yuanlai Liu, Devashree Tripathy, Laxmi Bhuyan, and Zizhong Chen. 2019. GreenMM: energy efficient GPU matrix multiplication through undervolting. In *Proceedings of the ACM International Conference on Supercomputing*. 308–318.

[52] Hadi Zamani, Devashree Tripathy, Laxmi Bhuyan, and Zizhong Chen. 2020. SAOU: safe adaptive overclocking and undervolting for energy-efficient GPU computing. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. 205–210.

[53] Tomás Zegard and Glaucio H Paulino. 2013. Toward GPU accelerated topology optimization on unstructured meshes. *Structural and multidisciplinary optimization* 48, 3 (2013), 473–485.

A Artifact Appendix

A.1 Abstract

This artifact contains the software framework (PowerLA) for energy saving matrix decomposition evaluation. It includes the energy-saving implementations of Cholesky, LU, and QR decomposition using H2R, SR, and BSR approaches. The framework is built based on the hybrid matrix decomposition algorithms in the MAGMA library. This artifact is available at: https://doi.org/10.5281/zenodo.7317070

A.2 Hardware requirements

- x86 CPU and NVIDIA GPUs (tested on a server with Intel Core i7-9700K with NVIDIA RTX 2080 Ti)

A.3 OS requirements

- Linux operating system (tested on Ubuntu 18.04)

A.4 Software dependencies/configurations

- For measuring CPU power: cpu-energy-meter
- For adjusting CPU clock frequency: cpupower
- For adjusting CPU core voltage: intel-undervolt
- For running GPU code: CUDA 11.4+.
- For measuring GPU power, control GPU clock offset: NVIDIA GPU driver 450.80.02+.
- For enabling GPU overclocking, set Coolbits to the maximum allowed. The Coolbits on the tested system was set to `28`.
- For compilation: GCC 7.5.0+ and NVCC 11.4+.
- For configuring the project: CMake 2.8+.

A.5 Building our PowerLA framework

- The PowerLA framework was built based on the MAGMA library v 2.5.4, so it uses the same build system as the MAGMA library. Please follow the `README.md` in the root directory to build PowerLA.

A.6 Running optimized matrix decompositions

1. The major three one-sided matrix decomposition algorithms (Cholesky, LU, and QR) are optimized. They are implemented in:

- Cholesky: ./src/dportf_gpu.cpp; ./src/sportf_gpu.cpp
- LU: ./src/dgetrf_gpu.cpp; ./src/sgetrf_gpu.cpp
- QR: ./src/dgeqrf_gpu.cpp; ./src/sgeqrf_gpu.cpp

In each source code file, we added the following variables to control the energy-saving and fault-tolerance behavior of each matrix decomposition.

- `int tmu_curr_freq` and `int tmu_base_freq`: set the current and based clock frequency of GPU. They should be the same.
- `int tmu_base_offset`: set the base clock offset of GPU.
- `int tmu_opt_offset`: set the optimized clock offset of GPU.
- `adj_gpu(device, tmu_base_freq, 338000)`: set the power limit of GPU.
- `int pd_curr_freq` and `int pd_base_freq`: set the current and based clock frequency of CPU. They should be the same.
- `bool reclaim_slack`: control if we want to enable Slack Reclamation (BSR or SR).
- `double reclamation_ratio`: control how much of the slack is reclaimed by the task on the critical path.
- `bool overclock`: control if we want to overclock with undervolting.
- `bool autoboost`: control if we want to enable hardware R2H.
- `bool COL_FT` and `bool ROW_FT` control if we want to enable ABFT (single-side or full checksum)

2. Once the PowerLA framework is built, the MAGMA testing binary executables can be used to run each matrix decomposition with a specified input matrix size. The executables can be run with:

```
<build dir>/testing/testing_*_gpu -N <size>
```

3. Configuring the variables for different modes

	Original	R2H	SR	BSR
`reclaim_slack`	false	false	true	true
`reclaimnation_ratio`	N/A	N/A	0	0-1
`overclock`	false	false	false	true
`autoboost`	false	true	false	false
`COL_FT/ROW_FT`	false	false	false	true

4. When each test finishes execution it will output:

- Energy consumption of CPU and GPU (total)
- Time cost (per operation & total)
- Predicted time cost (per iteration)
- The slack prediction error (total average)
- Clock frequency of CPU and GPU (per iteration)
- Decisions on slack reclamation (per iteration)