Energy Balance between Voltage-Frequency Scaling and Resilience for Linear Algebra Routines on Low-Power Multicore Architectures

Sandra Catalán[a], José R. Herrero[b], Enrique S. Quintana-Ortí[a], Rafael Rodríguez-Sánchez[a],

[a]*Depto. Ingeniería y Ciencia de Computadores, Universidad Jaume I, Castellón, Spain.*
[b]*Dept. d'Arquitectura de Computadors, Universitat Politècnica de Catalunya, Spain.*

Abstract

Near Threshold Voltage (NTV) computing has been recently proposed as a technique to save energy, at the cost of incurring higher error rates including, among others, Silent Data Corruption (SDC). In this paper, we evaluate the energy efficiency of dense linear algebra routines using several low-power multicore processors and we analyze whether the potential energy reduction achieved when scaling the processor to operate at a low voltage compensates the cost of integrating a fault tolerance mechanism that tackles SDC. Our study targets algorithmic-based fault-tolerant versions of the dense matrix-vector and matrix(-matrix) multiplication kernels (GEMV and GEMM, respectively), using the BLIS framework, as well as an implementation of the LU factorization with partial pivoting built on top of GEMM. Furthermore, we tailor the study for a number of representative 32-bit and 64-bit multicore processors from ARM that were specifically designed for energy efficiency.

Keywords: Energy efficiency, voltage-frequency scaling, fault tolerance, dense linear algebra, high performance, multicore processors.

Email addresses: catalans@uji.es (Sandra Catalán), josepr@ac.upc.edu (José R. Herrero), quintana@uji.es (Enrique S. Quintana-Ortí), rarodrig@uji.es (Rafael Rodríguez-Sánchez)

1. Introduction

Aggressive technology scaling is steadily shrinking transistor size, at the pace dictated by Moore's Law [1], increasing the occurrence of faults in computing systems in the way. In the future, as the number of components integrated in CMOS circuits grows, the mean time between failures (MTBF) for the full system will be significantly reduced, promoting *resilience* into a crucial challenge on the road towards Exascale systems [2, 3].

Under the pressure of the *energy wall* [2, 3, 4], near threshold voltage (NTV) computing has been proposed as a means to reduce energy consumption [5]. However, scaling the operational voltage diminishes the critical charge required to flip a stored value [6, 7], so that particles of low energy due to atmospheric radiation can cause soft errors (SE). Consequently, error rates for low power modes are higher than those present in high power modes, as the error rate grows exponentially with the reduction on the supply voltage [8].

Already today, reliability and energy consumption are key criteria for system design. However, hardware protection mechanisms cannot be relied upon as the sole method for SE mitigation. Instead, they should be used in conjunction with other mitigation techniques for improved reliability at an energy-efficient cost. In this scenario, fault tolerance techniques such as checkpointing or redundancy (replication) may be too costly from the computational and energy perspectives, favoring alternative approaches based on application-specific algorithmic-based fault tolerance (ABFT) [3, 9, 10].

In this paper, we investigate the energy costs of tackling in software with SE that result in silent data corruption (SDC), using the dense matrix-vector multiplication (GEMV) and dense matrix multiplication (GEMM) as case studies. These two kernels are the cornerstone upon which the entire dense linear algebra (DLA) software stack, and implicitly many scientific and engineering codes, rely for high performance. One particular example is the LU factorization with partial pivoting (GETRF), which can be encoded to cast a major fraction of its computations in terms of GEMM, and is adopted in our work as an additional case study. As target architectures, we employ several general-purpose multicore processors from ARM that are specially designed to deliver reasonable performance with high energy efficiency. In more detail, our paper makes the following contributions:

- We evaluate the energy efficiency under different voltage-frequency scaling (VFS) configurations. For this purpose, we leverage efficient

multi-threaded implementations of GEMV, GEMM and GETRF based on the BLIS framework [11], especially tailored for the ARMv7 Cortex-A7/A15 and the ARMv8 Cortex-A53/A57.

- We review the theoretical costs of introducing simple software resilience techniques for GEMV, GEMM, and GETRF. For GEMV we discuss how to deal with the errors via redundancy, exploiting the memory-bound nature of this kernel to deliver an affordable resilience mechanism. For GEMM, we follow the ABFT described in [12]; and the same mechanism provides a reliable solution for GETRF, when built on top of a resilient GEMM.

- Finally, we introduce iso-energy models that capture the interplay between VFS, error detection costs, and error correction overhead/error rates, and how these factors combine to modify the energy efficiency for these three linear algebra operations and the target low-power architectures when operating in *low-voltage at extended margins* (LVEMs) configurations.

At this point, we note that the detection+correction strategies discussed for the target dense linear algebra operations can be regarded as theoretical proposals, and the details of their actual implementation do not impact our iso-energy model. Indeed, we believe that a detailed analysis of the reliability of the error detection (and correction) mechanism(s), which may indeed suffer from errors themselves, is beyond the scope for this work. We consider this as a clear target for a research that aims to produce practical and efficient resilient implementations of dense linear algebra kernels; see [12].

In Section 2 we review some related work. In Section 3 we briefly review the BLIS implementation of GEMV and GEMM, and we describe how to modify them in order to produce resilient versions, as well as the implications on the LU factorization in LAPACK. Section 4 outlines the experimental setup. In Section 5 we present an experimental evaluation of four ARM multicore processors from the points of view of performance and energy efficiency. Next, in Section 6 we analyze the trade-off between energy consumption and fault tolerance. Finally, in Section 7 we summarize our work with some concluding remarks.

2. Related work

Reducing the operating voltage of an electronic circuit is a well-known technique that can potentially diminish its power consumption. Oftentimes, this reduction in voltage comes together with a decrease of the operational frequency, an strategy known as VFS [13, 14, 15]. However, the supply voltage can also be reduced while maintaining the operating frequency, an approach known as undervolting, in an attempt to save power while preserving throughput. Undervolting and VFS outside of the nominal region can nevertheless introduce errors, which need to be corrected in case the voltage is dropped in excess. Hardware support for error detection/correction from operation at lower supply voltage was introduced in [16]. The impact on the memory system was specifically studied in [17, 18, 19]. In [20] the authors explore the potential energy benefits of reducing the chip's voltage to the safe limit (V_{min}) at a fixed frequency. (V_{min} is program dependent.) Exceeding such safe limits causes SDC to arise, with an avalanche error effect when the voltage is pushed below a certain threshold. Interestingly, an additional 4–5% undervolt below V_{min} causes the OS to crash. However, that work does not address resilience mechanisms. Instead, it focuses only on shifting the guard-band down for energy improvement without impacting the correctness level. They show that there is about a 20% voltage guard-band for the graphics processors tested in their work which can result in up to 25% energy savings.

While most previous research focuses on exploring energy-saving and resilience-enhancing opportunities separately, only very few selected publications study their interactions. Some works assess the energy costs of traditional resilience-enhancing methods such as checkpointing/restart or replication. The former is analyzed in [21], modeling its energy costs and introducing checkpoint compression to reduce the energy consumption. The latter is studied in [22], which examines the energy costs of coordinated checkpointing and replication, contributing mainly to make replication more time- and energy-efficient. In [10, 23] the authors study the interplay between energy efficiency and resilience (mainly the checkpoint/restart technique) in high performance computing with a focus on undervolting. More specifically, they develop analytical models to investigate the potential of achieving high energy efficiency in HPC by undervolting, with hardware/software-level resilience techniques applied on-the-fly to guarantee the correct execution of HPC runs.

In this work we consider low-voltage at extended margins (LVEM), i.e., configurations operating below nominal voltage which may incur soft errors, but always considering voltage values above some threshold where an avalanche error effect will appear and systematically cause catrastofic unrecoverable errors. Our paper separates from previous work in that we do not attempt to develop a model for the error type or error rates that may occur when applying VFS/undervolting at extended margins. Instead, we focus on the specific domain of dense linear algebra operations, and we analyze the trade-off between energy savings attained via LVEM computing and the detection+correction overheads introduced by a (generic) resilience mechanism.

3. Resilient Versions of the BLIS Kernels and the LU Factorization

In this section, we describe the implementation of the BLIS routines for GEMV and GEMM. In addition, we analyze the theoretical costs of error detection and correction for both kernels, as well as for the GETRF routine for the LU factorization, under a unified framework. We note that the multi-layered organization of the BLIS kernels renders that, inside each loop, a specific part of the result is updated using certain parts of the inputs. This in turn allows the integration of a fault tolerance mechanism within different loops, trading off workspace/error correction cost for error detection cost, as described next.

3.1. Matrix-vector multiplication

Consider the BLAS-2 GEMV kernel, $y := \alpha Mx + \beta y$, where M is an $m \times n$ matrix; y, x are vectors of m, n components, respectively; and α, β are scalars that, for simplicity, we consider equal to 1. Assuming the matrix is stored contiguously in memory (either by rows or by columns), this operation is implemented in BLIS as three nested loops around two packing routines and a micro-kernel; see Figure 1. The packing routines copy the contents of y, x into contiguous buffers, (but only if the vectors are not already stored with unit stride,) and the micro-kernel casts the operations in terms of a *fused vector-vector kernel* [11].

BLIS does not offer multi-threaded versions of BLAS-2 kernels such as GEMV. However, a straight-forward solution to parallelize this operation on the architectures targeted in our work can extract the parallelism from Loop 1 by statically distributing the iteration space (and, therefore, the workload)

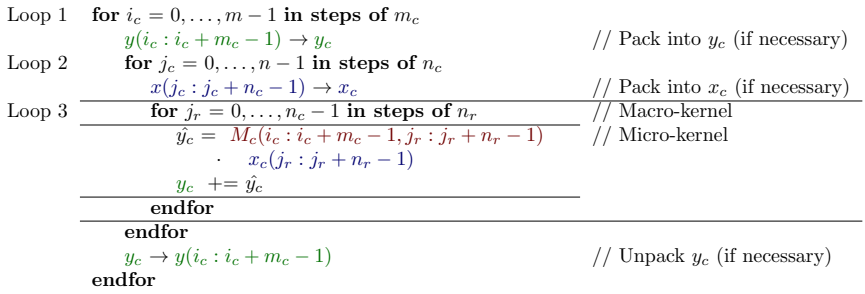

Figure 1: High performance implementation of GEMV in BLIS. In the code, $M_c \equiv M(:, j_c : j_c + n_c - 1)$ is a notation artifact, while y_c, x_c correspond to actual buffers that are involved in data copies.

of this loop among the threads. This can be easily achieved using, e.g., OpenMP.

The low ratio between the floating-point arithmetic operations (flops) performed in GEMV ($2mn$) and the number of memory accesses ($mn + m + n$ at best) turns this kernel into a memory-bound operation that, on current architectures, proceeds at the speed dictated by the bandwidth of the memory layer where M is stored. Let us assume that M is large so that it only fits into the system's main memory $\mathcal{M}$. A potential strategy to introduce a resilience mechanism into GEMV consists in computing each micro-kernel twice and checking for differences between the two results (redundancy or duplication). If the submatrix M_c fits into a certain level of the cache hierarchy that is considered reliable, say $\mathcal{C}$, we can then expect that the second execution of the micro-kernel proceeds at the speed of this cache level, which is hopefully much faster than the main memory, introducing a low detection overhead. We can denote this cost as $\mathcal{O}_d^{mv}$, and we can expect its value to reflect the ratio between the bandwidths of $\mathcal{C}$ and $\mathcal{M}$. If an error is detected, the correction can simply run the micro-kernel a third time, using a majority vote to select the correct value(s). Here it is possible to apply the correction selectively, at a finer granularity, yielding a much lower correction overhead. For example, if the i_r-th entry of $\hat{y}_c$ is corrupted, we only need to re-compute the dot product between the i_r-th row of M_c and x_c, for a cost of $2n_r$ flops. Compared with the $2m_c n_c$ flops required to compute the complete micro-kernel, this results in a correction overhead

$$\mathcal{O}_c^{mv} = \frac{n_r}{m_c n_c} \cdot E_{abs}^{mv}, \tag{1}$$

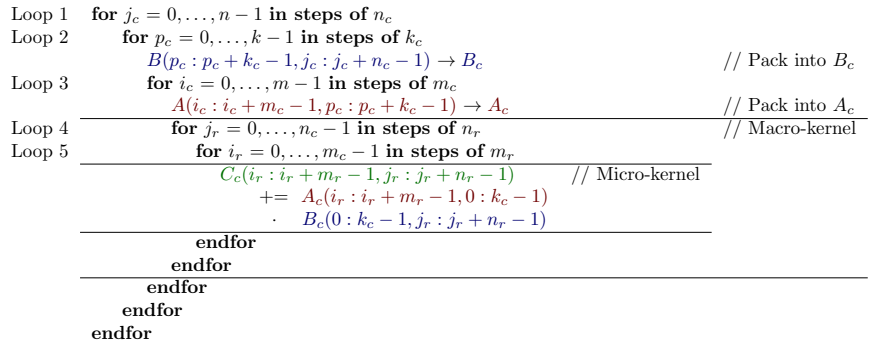

Figure 2: High performance implementation of GEMM in BLIS. In the code, $C_c \equiv C(i_c : i_c + m_c - 1, j_c : j_c + n_c - 1)$ is just a notation artifact, introduced to ease the presentation of the algorithm, while A_c, B_c correspond to actual buffers that are involved in data copies.

where E_{abs}^{mv} is the absolute (total) number of entries in y_c with errors.

3.2. Matrix multiplication

Consider next the BLAS-3 GEMM kernel, $C = \alpha AB + \beta C$, where C is $m \times n$, A is $m \times k$, B is $k \times n$, and α, β are scalars. Hereafter we will define the problem dimension with the triplet $\{m, n, k\}$ and, for simplicity, we will assume that $\alpha = \beta = 1$. BLIS [11] implements this operation as two *packing routines* embedded into three nested loops around a *macro-kernel*. In practice, the packing routines promote that certain blocks of A and B are copied into the lower-level cache(s), and enforce that the data in these copies are accessed with unit stride. In addition, BLIS further divides the macro-kernel update into a sequence of rank-k_c updates, or *micro-kernels*. These micro-operations perform the actual flops, transferring the data between the registers/L1 cache and the lower levels of the cache hierarchy; see Figure 2 and [11].

The BLIS parallelization of GEMM for multi-threaded multicore processors and modern many-threaded architectures was discussed in [24, 25]. The approach parallelizes the nested five–loop organization of GEMM at one or more levels (i.e., loops), taking into account the cache organization of the target platform, the granularity of the computations, and the risk of race conditions, among other factors. For the multicore processors targeted in this work, an efficient choice is to extract the parallelism from Loop 4 only [26] via, e.g., OpenMP.

The two-sided checksum-based detection mechanism for BLIS GEMM presented in [12] operates at the macro-kernel level. That is, within Loop 3 in

Figure 2, which computes the macro-operation $C_c \mathrel{+}= A_c B_c$ of dimension $\{m_c, n_c, k_c\}$. It requires a total of six matrix-vector products plus the calculation of two matrix norms (involving A_c, B_c or C_c), for a detection cost of $4m_c n_c + 5m_c k_c + 5k_c n_c$ flops. The overhead of the detection mechanism with respect to the cost of computing the macro-block C_c is thus:

$$\mathcal{O}_d^{mm} = \frac{4m_c n_c + 5m_c k_c + 5k_c n_c}{2m_c n_c k_c}. \tag{2}$$

The simplest method for correcting errors consists in re-calculating the whole macro-kernel (product). With this approach, the cost of correcting a macro-block C_c with errors becomes $2m_c n_c k_c$ flops, and the overhead ratio with respect to the total cost (in case C_c contains errors) is 1. However, by detecting the "location of the error", the correction can be performed at a lower granularity, considerably reducing its overhead [12]. For example, the correction can operate at the micro-kernel level (Loop 5 in Figure 2), on each $m_r \times n_r$ micro-block of C_c updated with a product of dimension $\{m_r, n_r, k_c\}$; see Figure 2. The cost of correction applied to the macro-kernel $C_c \mathrel{+}= A_c B_c$ then becomes $2m_r n_r k_c \, E_{abs}^{mm}$, where E_{abs}^{mm} is the number of micro-blocks in C_c with at least one error, and the overhead of correcting the errors within this macro-block is

$$\mathcal{O}_c^{mm} = \frac{2m_r n_r k_c}{2m_c n_c k_c} \cdot E_{abs}^{mm} = \frac{m_r n_r}{m_c n_c} \cdot E_{abs}^{mm}. \tag{3}$$

For many architectures (including the cores targeted in our work), the dimensions m_c, n_c involved in the detection are 2–3 orders of magnitude larger than m_r, n_r, and therefore the correction cost is low.

3.3. LU factorization

The legacy routine in LAPACK [27] for the LU factorization with partial pivoting encodes the right-looking blocked variant of this operation as a loop that processes the matrix, from the left/top corner to the right/bottom one, by column blocks (panels) of n_b columns, where n_b is often referred to as the algorithmic block size. In rough detail, for each iteration of the loop, the routine factorizes the "current" panel below the matrix diagonal, applies the pivots to the remaining blocks of the matrix, and then updates the blocks to the right of the current panel with respect to the factorization [28].

A key aspect to realize is that, in practice, n_b is chosen to be small with respect to the problem size, so that the algorithm casts most of its operations

in terms of rank-n_b updates, which are performed via the efficient GEMM. Therefore, a plain approach to obtain a parallel execution of GETRF on a multicore processor, can simply rely on a multi-threaded version of GEMM to leverage the hardware concurrency of the system.

In addition, the fact that GETRF casts most of its operations in terms of GEMM provides a natural detection+correction mechanism based on the ABFT version of GEMM described earlier. The remaining operations of the factorization, which given an optimal value of n_b represent a tiny fraction of the total, can be protected via a redundancy approach as that adopted for GEMV. Under these conditions, we can expect that the detection and correction overheads for GETRF are similar to those of GEMM; i.e., $\mathcal{O}_d^{lu} \approx \mathcal{O}_d^{mm}$ and $\mathcal{O}_c^{lu} \approx \mathcal{O}_c^{mm}$.

3.4. Detection vs correction

The design of efficient detection+correction mechanisms, even within a restricted domain such as dense linear algebra operations, introduces some complex questions. Among these, we can mention the target error rate, the possibility of errors in the detection and/or correction mechanisms themselves, the fault model, the type of errors (faults that manisfest in the exact same value when repeating the calculation, single event upsets), etc. We remark that, while these questions are relevant when proposing a practical implementation of the resilience mechanisms, they lie beyond the main focus of this paper.

At this point, we note the difference between the two sources of overhead: while the detection is performed independently of the error probability, the correction only occurs when an error is detected [12]. Therefore, the cost for the detection mechanism is fixed and the correction overhead is directly proportional to the error rate.

Consider now GEMM, and let us define

$$E_{rel}^{mm} = \frac{E_{abs}^{mm}}{m_c n_c/(m_r n_r)} \tag{4}$$

as the *rate* of micro-blocks with some error(s) per macro-block. Thus, given that C_c consists of $m_c n_c/(m_r n_r)$ micro-blocks, we obtain that $\mathcal{O}_c^{mm} = E_{rel}^{mm}$. Generalizing this result, $\mathcal{O}_c^{mm}$ equals the average *rate* of micro-blocks per macro-block of C that contain at least one error. An analogous derivation yields that, for GEMV, $\mathcal{O}_c^{mv}$ is equivalent to the average rate of micro-blocks of y that contain at least one error.

4. Experimental Setup

For the experimental evaluation, we employed the multicore architectures from the following two systems in the experimentation:

ODROID-XU3. This board is furnished with a Samsung Exynos 5422 system-on-chip (SoC). This processor comprises an ARM Cortex-A15 quad-core cluster plus an ARM Cortex-A7 quad-core cluster, both implementing the ARMv7a microarchitecture. Each Cortex core has its own private 32-Kbyte L1 (data) cache. The four ARM Cortex-A15 cores share a 2-Mbyte L2 cache and the four ARM Cortex-A7 cores share a smaller 512-Kbyte L2 cache. In addition, the two clusters access a 2-Gbyte DDR3 RAM. The frequency can be varied in the range 200 MHz–1.4 GHz for the Cortex-A7 cluster and 200 MHz–2.0 GHz for the Cortex-A15 cluster, with a 100 MHz-step in both cases. However, in order to reduce the number of experiments, we will perform our experiments with frequencies separated by 200 MHz. This then fixes the corresponding (supply) voltage as shown in the corresponding columns of Table 1. Note that the voltage remains constant in the frequency ranges [200,500] MHz for the Cortex-A7 and [200,700] MHz for the Cortex-A15 [29]. All cores in the same cluster must operate at the same frequency.

In the ODROID-XU3 board the `pmlib` monitoring tool [30] collects power consumption corresponding to instantaneous power readings from four independent sensors/power domains in the board (Cortex-A7 cluster, Cortex-A15 cluster, DRAM and GPU), with a sampling rate of 250 ms. (To compensate for this low sampling rate, our calibration experiments repeat the execution of the kernels for a period that is sufficiently long to obtain enough power measurements.) When evaluating the energy of one of the clusters, we only consider the sensor corresponding to that component. Given that we do not employ the GPU in our experiments, and that the four Cortex-A15 cores and up to three Cortex-A7 cores can be disabled when idle, we can expect a negligible power consumption for these components when inactive [29].

Juno (r0). This development platform features an ARM Cortex-A57 dual-core cluster plus an ARM Cortex-A53 quad-core cluster, both implementing the ARMv8 microarchitecture. Each core has its own private 32-Kbyte L1 (data) cache. The two ARM Cortex-A57 cores share a 2-Mbyte L2 cache and the four ARM Cortex-A53 cores share a smaller 1-Mbyte L2 cache. Both clusters are connected to a DDR3 RAM with a capacity of 8 Gbytes. The fre-

quency and voltage can be varied as displayed in the corresponding columns of Table 1. All cores in the same cluster must operate at the same frequency.

In the Juno board `pmlib` collects power consumption data corresponding to instantaneous power readings using a data acquisition device from National Instruments connected to the internal shunt resistors available in the board (Cortex-A53 cluster, Cortex-A57 cluster, system, and GPU), with a sampling frequency of 100 Hz. When evaluating the energy of one of the clusters, we only consider the line corresponding to that component, for the same reasons exposed above.

	ARM Cortex-A7		ARM Cortex-A15		ARM Cortex-A53		ARM Cortex-A57	
Conf.	Freq.	Voltage	Freq.	Voltage	Freq.	Voltage	Freq.	Voltage
C_1	0.200	0.913	0.200	0.912	0.450	0.820	0.450	0.810
C_2	0.400	0.913	0.400	0.912	0.575	0.860	0.625	0.850
C_3	0.600	0.951	0.600	0.912	0.700	0.910	0.800	0.900
C_4	0.800	1.026	0.800	0.925	0.775	0.960	0.950	0.950
C_5	1.000	1.101	1.000	0.973	0.850	1.010	1.100	1.000
C_6	1.200	1.176	1.200	1.023	–	–	–	–
C_7	1.400	1.273	1.400	1.062	–	–	–	–
C_8	–	–	1.600	1.115	–	–	–	–
C_9	–	–	1.800	1.191	–	–	–	–
C_{10}	–	–	2.000	1.318	–	–	–	–

Table 1: VFS configurations (voltage-frequency pairs, in V and GHz, respectively) available in the Samsung Exynos 5422 and Juno (r0) SoCs.

All the experiments were performed using IEEE 754 single-precision and a multi-threaded implementation of BLIS that spawns threads that run on either the Cortex-A7, the Cortex-A15, the Cortex-A53, or the Cortex-A57 cluster. Hereafter we report performance in terms of GFLOPS (G), power dissipation (P) in W(atts), and energy efficiency (EE) as the ratio GFLOPS/W.

5. Energy Efficiency of the BLIS Kernels and the LU Factorization

We next analyze the performance and energy efficiency of the GEMV, GEMM kernels from BLIS, and the GETRF routine for the LU factorization built on top of GEMM, on the four types of cores available on the target systems.

Table 2 reports the performance, power dissipation and energy efficiency metrics for the three linear algebra routines, executed using four cores of the Cortex-A7, the Cortex-A15 or the Cortex-A53 clusters, or two cores of the Cortex-A57 cluster. That corresponds to the maximum number of each

Architecture	Configuration	GEMV			GEMM			GETRF		
		G	P	EE	G	P	EE	G	P	EE
ARM Cortex-A7	C_1	0.271	0.064	4.238	0.758	0.072	10.473	0.554	0.060	9.259
	C_2	0.501	0.115	4.360	1.573	0.140	11.204	1.193	0.119	9.984
	C_3	0.677	0.166	4.086	2.355	0.217	10.834	1.809	0.186	9.733
	C_4	0.802	0.233	3.440	3.258	0.328	9.937	2.398	0.279	8.603
	C_5	0.911	0.319	2.853	4.070	0.483	8.426	2.998	0.409	7.323
	C_6	0.999	0.417	2.395	4.893	0.672	7.278	3.536	0.561	6.303
	C_7	1.000	0.541	1.848	5.630	0.943	5.968	3.980	0.784	5.078
ARM Cortex-A15	C_1	0.381	0.188	2.028	3.502	0.496	7.067	2.205	0.388	5.679
	C_2	0.718	0.335	2.141	7.109	0.970	7.328	4.619	0.761	6.068
	C_3	0.997	0.471	2.115	10.652	1.436	7.418	7.067	1.132	6.245
	C_4	1.227	0.582	2.108	14.165	1.926	7.356	9.232	1.506	6.130
	C_5	1.396	0.768	1.817	17.757	2.686	6.611	11.702	2.120	5.519
	C_6	1.539	0.981	1.568	21.145	3.632	5.822	13.700	2.793	4.905
	C_7	1.648	1.182	1.394	24.344	4.562	5.336	15.856	3.532	4.489
	C_8	1.756	1.489	1.179	27.710	5.978	4.635	17.152	4.539	3.779
	C_9	1.728	1.855	0.931	–	–	–	–	–	–
	C_{10}	1.744	2.569	0.679	–	–	–	–	–	–
ARM Cortex-A53	C_1	0.877	0.198	4.425	7.461	0.359	20.787	4.901	0.272	18.148
	C_2	1.008	0.259	3.887	9.488	0.510	18.594	6.221	0.374	16.818
	C_3	1.106	0.327	3.387	11.271	0.685	16.447	7.453	0.494	15.204
	C_4	1.148	0.399	2.880	12.533	0.855	14.658	8.164	0.620	13.161
	C_5	1.191	0.470	2.536	13.629	1.045	13.040	8.883	0.727	12.333
ARM Cortex-A57	C_1	0.733	0.270	2.715	6.159	0.536	11.482	4.189	0.509	8.222
	C_2	0.972	0.404	2.408	8.491	0.843	10.072	5.801	0.781	7.429
	C_3	1.163	0.560	2.077	10.812	1.214	8.903	7.375	1.157	6.373
	C_4	1.286	0.709	1.814	12.698	1.586	8.006	8.702	1.490	5.839
	C_5	1.375	0.858	1.603	14.538	2.059	7.059	9.982	1.952	5.115

Table 2: Performance and energy efficiency for GEMV, GEMM and GETRF executed in the Exynos 5422 and Juno (r0) SoCs. The red color identifies the fastest configuration while the green color indicates the most energy-efficient one. When executing GEMM/GETRF, results could not collected in the Cortex-A15 cores for the two highest frequencies, because the chip temperature raised too high and the operating system reacted by automatically lowering the frequency.

core type available on each cluster. The only exception is the execution of GEMV on the Cortex-A15/Cortex-A57, for which we only employed a single thread/core as the use of a higher number of threads did not deliver any performance gain, due to the limited memory bandwidth of these architectures and the memory-bound nature of this operation. The sizes for each problem/architectures were selected to be large enough to achieve a large fraction of the asympthotic GFLOPS rate for each combination.

From the point of view of raw performance, GEMM and GETRF deliver considerably higher GFLOPS rates for the Cortex-A15 than for the Cortex-A7. However, the differences are not so significant for these two compute-bound kernels when executed on the Cortex-A57 vs the Cortex-A53 as the former integrates only half the number of cores of the latter. On the other hand, for the memory-bound GEMV we can observe narrower differences between the two ARMv7 microarchitectures and the same holds for the two ARMv8 processors.

Let us analyze the effect of the frequency next. Considering GEMM, GETRF and the ARMv7a microarchitectures, the increase of the GFLOPS rate is slightly superlinear for the Cortex-A7 and a bit sublinear for the Cortex-A15. For example, in the latter, when the operation frequency raises from 200 MHz to 1.6 GHz (a factor of 8×), the GFLOPS rate for GEMM grows by 7.91×. Unsurprisingly, the behaviour for GETRF is similar, with a growth of 7.78×. On the other hand, GEMV shows a less appealing scenario, with a performance improvement of only 4.6× under the same conditions. The behaviour for the ARMv8 microarchitectures is similar to that of the Cortex-A15. For instance, in the Cortex-A57, changing the frequency from 450 MHz to 1.1 GHz (a factor of 2.4×) results in speedups of 2.36× for GEMM and 2.38× for GETRF, but only 1.87× for GEMV.

In general, the energy efficiency (expressed in terms of GFLOPS/W), benefits from operating at low voltages. In case several frequencies operate under the same voltage, then using the highest available frequency is normally desirable in order to obtain higher performance, and consequently, better energy efficiency.

To conclude this analysis, we point out the differences between a memory-bound operation such as GEMV vs the compute-bound GEMM, GETRF. This is reflected in the much lower performance and scalability (with respect to the number of cores and frequency) of the former but, interestingly, also in the considerably lower dissipation rates attained by GEMV.

6. Resilience versus Energy for the BLIS Kernels and the LU factorization

The general assumption by the sytem's programmer/user is that the "hardware" offers an error-free execution model, but LVEM partially removes this condition in the search for higher energy efficiency by operating below the nominal voltage. The question to answer then is whether the energy gains attained with the reduction of voltage compensate the energy costs of tackling the errors in the software.

In order to investigate this question in subsections 6.2–6.4, in subsection 6.1 we first formulate a model that can be leveraged to predict the power dissipation as a function of the voltage-frequency variations.

Architecture		GEMV	GEMM	GETRF
ARM Cortex-A7	α_1	0.2048	0.4085	0.3415
	α_2	0.0528	0.0000	0.0000
	$\mathcal{R}$	0.0254	0.0209	0.0144
ARM Cortex-A15	α_1	0.6674	2.9285	2.2532
	α_2	0.1307	0.0000	0.0000
	$\mathcal{R}$	0.0165	0.0228	0.0119
ARM Cortex-A53	α_1	0.4146	1.1972	0.7767
	α_2	0.1088	0.0000	0.0540
	$\mathcal{R}$	0.0063	0.0072	0.0158
ARM Cortex-A57	α_1	0.6738	1.8638	1.7610
	α_2	0.1326	0.0000	0.0000
	$\mathcal{R}$	0.0225	0.0071	0.0120

Table 3: Parameters for modeling power dissipation as a function of the voltage and frequency, and relative residual $\mathcal{R}$.

6.1. Power models

Our power model is grounded in the hypothesis that the CPU power is given by

$$P_{CPU} = P_{static} + P_{dynamic},$$

where the dynamic power

$$P_{dynamic} \ \propto \ V_{cc}^2 \times f,$$

with V_{cc} and f standing for the CPU voltage and frequency, respectively [31]. In addition, in principle we have

$$P_{static} \ \propto \ V_{cc};$$

see also [31]. However, after an experimental analysis of different models, for the static power we decided to depart from this assumption to use

$$P_{static} \ \propto \ V_{cc}^2,$$

yielding the model

$$P_M(V, f) \ \ = \ \ \alpha_1 V^2 f + \alpha_2 V^2, \tag{5}$$

where the coefficients α_1 and α_2 are respectively connected with the dynamic power (due to CMOS switching) and the static power (due to leakage) of the system [31].

To callibrate the coefficients of the power models, we solved a linear least squares problem with nonnegativity constraints, using the actual power consumption in Table 2 and the voltage-frequency pairs in Table 1. The

parameters for the power models are presented in Table 3 for each operation–microarchitecture pair. The relative residual defined by

$$\mathcal{R} = \|(P - P_M(V, f))\|_2 / \|P\|_2, \tag{6}$$

also included in Table 3, quantifies the deviation from the model estimation $P_M(V, f)$ and the actual power consumption values P in Table 2, reflecting the accuracy of the model. Concretely, for these models/parameters, the largest relative error is 2.54%, but it is also below 1% in several cases. As illustrated in the following section, having an accurate model paves the way to elaborate an iso-energy model that relates energy consumption and error rates when operating with varied VFS configurations.

Considering the coefficients of the power models in Table 3, we observe two distinct cases, corresponding to the compute-bound GEMM and GETRF vs the memory-bound GEMV. For the former operations, the parameter values show that the power consumption is solely dictated by the dynamic component ($\alpha_2 = 0$). Conversely, the static component plays a nonnegligible role for the latter. An exception to this is the execution of GETRF on the Cortex-A53, for which there is a contribution from both the dynamic and static parts to the total power. This is related to the existence of a serial bottleneck in the execution of this factorization on the Cortex-A53.

6.2. Trading off hardware reliability for energy efficiency

The determination of the optimal configuration from the point of view of energy efficiency in the previous section sparks the investigation of an scenario where the clusters operate below the nominal voltage. (We note that we cannot enforce this configuration in our hardware.) In this situation, we could expect a reduction of power dissipation but also an increase of the SDC rate, due to the operation in the LVEM region. The question which arises then is *how much detection+correction overhead we can afford in a fault-tolerant implementation of DLA operations while matching (or, hopefully, improving) the energy efficiency of the error-free execution.*

In order to derive a model that relates the correction and detection overheads to the energy efficiency, let us consider an error-free *reference* configuration (V_R, f_R), with V_R above the safety threshold margin, delivering a performance rate G_R for a power dissipation P_R. On the other hand, assume an error-prone *arbitrary* configuration (V_A, f_A), with V_A below the lowest nominal margin, offering a performance G_A, and a power dissipation P_A which can be approximated using our models as $P_M(V_A, f_A)$.

In the error-prone execution, we can expect to pay a total overhead determined by $\mathcal{O}_d$ and $\mathcal{O}_c$. Concretely, the effect of both sources of overhead can be modeled in the same manner, by assuming they yield a decrease in the "effective" GFLOPS rate to $G_A^{Eff} = G_A(1 + \mathcal{O}_d + \mathcal{O}_c)^{-1}$, with overhead factors $0 \leq \mathcal{O}_d, \mathcal{O}_c$. Note that this consideration turns the analysis of the energy trade-offs between VFS and resilience independent of the specific fault tolerance tecnique introduced in the algorithms.

In order to match the energy efficiency (iso-energy) of the original routine, executed with the reference configuration and no errors, with that of the fault-tolerant version, executed with the arbitrary configuration with potential errors causing additional overheads $\mathcal{O}_d$ and $\mathcal{O}_c$, the following must hold:

$$\frac{G_R}{P_R} = \frac{G_A^{Eff}}{P_A} \quad = \quad \frac{G_A}{(1+\mathcal{O}_d+\mathcal{O}_c)P_A}. \tag{7}$$

From this expression, we can then obtain the sought-after iso-energy formula that relates the correction overhead, and therefore the error rate (as $\mathcal{O}_c = E_{rel}$ for GEMV, GEMM and GETRF), to the voltage-frequency configuration (for a given detection overhead) under iso-energy conditions:

$$\mathcal{O}_c^{iso} \quad = \quad E_{rel}^{iso} \quad = \quad \frac{G_A}{G_R} \cdot \frac{P_R}{P_A} - (1 + \mathcal{O}_d). \tag{8}$$

We will proceed next to specialize this model depending on the voltage-frequency relation and the type of DLA operation (compute-bound vs memory-bound).

6.3. Undervolting

Let us consider first an scenario where the voltage is reduced ($V_R \geq V_A \rightarrow 0$) while the frequency is maintained ($f_R = f_A$). Here, because of the constant frequency, we have $G_R = G_A$, and equation (8) can be simplified to

$$\mathcal{O}_c^{iso} \quad = \quad \frac{P_R}{P_A} - (1 + \mathcal{O}_d). \tag{9}$$

Then, applying the model in (5), and taking into account again that $f_R = f_A$,

$$\begin{array}{rclcl} \mathcal{O}_c^{iso} & = & \frac{P_R}{P_A} - (1 + \mathcal{O}_d) & \approx & \frac{P_M(V_R,f_R)}{P_M(V_A,f_A)} - (1 + \mathcal{O}_d) \\ & = & \frac{\alpha_1 V_R^2 f_R + \alpha_2 V_R^2}{\alpha_1 V_A^2 f_A + \alpha_2 V_A^2} - (1 + \mathcal{O}_d) & = & \left(\frac{V_R}{V_A}\right)^2 \frac{\alpha_1 f_R + \alpha_2}{\alpha_1 f_A + \alpha_2} - (1 + \mathcal{O}_d) \\ & = & \left(\frac{V_R}{V_A}\right)^2 - (1 + \mathcal{O}_d). & & \end{array} \tag{10}$$

This formula informs us that, for any architecture and DLA operation, the correction overhead/error rate that the execution can accommodate simply depends on the balance between the energy savings due to the voltage reduction, given by $(V_R/V_A)^2$*, and the fixed detection overhead to be paid for operating with an error-prone hardware, that is* $(1+\mathcal{O}_d)$. When these factors are equal, they cancel each other in (10) and the execution cannot incur any error without suffering an increase in the energy consumption. When the energy savings due to the voltage reduction exceed the energy costs of error detection, $(V_R/V_A)^2 > (1+\mathcal{O}_d)$, the surplus can be used to accommodate the correction overhead. If the error rate is low, this may even yield certain energy gains. In the opposite situation, i.e. $(V_R/V_A)^2 < (1+\mathcal{O}_d)$, even without any errors, there is an energy cost to be paid for the detection overhead that was not compensated by the reduction of voltage, and the correction overhead will add on top of this excess.

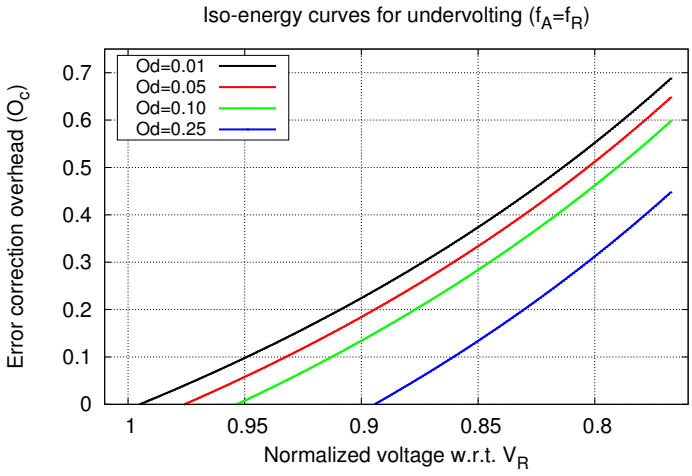

Figure 3: Configurations for the fault-tolerant versions of DLA operations, operating in undervolting conditions in the Cortex-A15, that attain the same energy efficiency as the original routines executed in the reference configuration C_3 without errors.

Figure 3 graphically depicts the iso-energy curves with respect to voltage scaling, for four different detection overhead rates: $\mathcal{O}_d = 0.01$, 0.05, 0.10 and 0.25. There we consider the reference voltage to be $V_R = 0.912$ (corresponding to the lowest nominal voltage for the Cortex-A15), and we vary $V_A \in [0.7, V_R]$, normalizing the reduction with respect to V_R. The plot shows

some relevant data points to be discussed. First, even when there are no errors, and the correction overhead is therefore nonexistent, the voltage still needs to be scaled down to a certain extent to compensate for the (fixed) cost due to the detection. From that point, as the voltage is further diminished, the fault-tolerant versions of the routines can accommodate a certain correction overhead while still delivering the same energy efficiency as the reference error-free execution. For example, when $\mathcal{O}_d = 0.10$ (green curve), the voltage has to be decreased to around 95% of V_R (about 0.870 V) to compensate for the detection overhead, even if there are no errors and, therefore, the execution incurs no correction overhead. For the same detection overhead, if the voltage is further scaled down to 85% of V_R (0.775 V), a correction overhead $\mathcal{O}_c^{iso}$ of about 28% basically matches the energy efficiency of the reference configuration. However, if the number of errors per macro-block in the DLA operations enforces a lower correction overhead, such voltage scaling will deliver a higher energy efficiency than the reference case.

6.4. General VFS

Let us consider next the more general case where the voltage is reduced, $V_R \geq V_A \rightarrow 0$, while the frequency is simultaneously scaled down from the reference value: $f_R \geq f_A \rightarrow 0$.

The first aspect to discuss is what is the effect of reducing the frequency on the performance. For this purpose, we remind that the analysis of the results in Section 5 exposed that, for GEMM and GETRF the performance basically scales linearly with the frequency, but the same does not hold for GEMV. However, in this subsection we are interested in the behaviour of these routines when both the frequency and the voltage are reduced from low reference values (V_R is among the lowest safe operation voltages and the corresponding f_R is also low). As the performance scalability problems of the memory-bound GEMV across the frequency were present only for the higher frequencies, we can also assume the linear behaviour of this operation for the cases we are interested in.

Imposing the linear relation on the frequency-performance interplay, we obtain $G_A = G_R(f_A/f_R)$, and (8) can be simplified as

$$\mathcal{O}_c^{iso} \;=\; \frac{G_A}{G_R} \cdot \frac{P_R}{P_A} - (1 + \mathcal{O}_d) \;=\; \frac{f_A}{f_R} \cdot \frac{P_R}{P_A} - (1 + \mathcal{O}_d), \tag{11}$$

and applying the model in (5),

$$\begin{aligned} \mathcal{O}_c^{iso} &= \frac{f_A}{f_R} \cdot \frac{P_R}{P_A} - (1 + \mathcal{O}_d) && \approx \frac{f_A}{f_R} \cdot \frac{P_M(V_R, f_R)}{P_M(V_A, f_A)} - (1 + \mathcal{O}_d) \\ &= \frac{f_A}{f_R} \cdot \frac{\alpha_1 V_R^2 f_R + \alpha_2 V_R^2}{\alpha_1 V_A^2 f_A + \alpha_2 V_A^2} - (1 + \mathcal{O}_d) &= \left(\frac{V_R}{V_A}\right)^2 \cdot \frac{f_A}{f_R} \cdot \frac{\alpha_1 f_R + \alpha_2}{\alpha_1 f_A + \alpha_2} - (1 + \mathcal{O}_d). \end{aligned} \tag{12}$$

From this point, we will distinguish two different behaviours, corresponding to the compute-bound operations GEMM and GETRF vs the memory-bound GEMV. For the former, the power model states that $\alpha_2 = 0$ (see Table 3), and thus (12) boils down to (10). *Therefore, the same analysis of the undervolting case applies to the compute-bound operations, for any architecture, when the frequency is reduced simultaneously with the voltage.*

Unfortunately, the analysis of GEMV is not so simple, as it will depend on the actual rate f_R/f_A (and the architecture). To study the behaviour in this operation, we consider four voltage-frequency relations (R_F^V):

1. Constant R_F^V: $f_A = \delta \cdot f_R$, with $0 \leq \delta \leq 1$, constant and independent of V_R/V_A. This is an "aggressive" approach as we choose to maintain f_A as V_A is scaled down. Note that if $\delta = 1$, this corresponds to undervolting.

2. Linear R_F^V: $f_R/f_A = V_R/V_A$.

3. Quadratic R_F^V: $f_R/f_A = (V_R/V_A)^2$.

4. Modeled R_F^V: $f_R/f_A = (\beta_1 V_R^2 + \beta_2 V_R)/(\beta_1 V_A^2 + \beta_2 V_A)$. Here we model the frequency as a function of the voltage by applying linear regression to identify the trend in the data in Table 2. (In practice, in order to obtain the models for the Cortex-A7 and Cortex-A15 architectures, we discard the lowest and two lowest configurations, respectively, as in those cases the voltage is maintained while the frequency is reduced.) This is a "conservative" approach as we can consider that the data in Table 2 follows a VFS pattern that is above the threshold safety margin.

Let us consider the execution of GEMV on the Cortex-A15 to illustrate the behaviour of the iso-energy curves. As the reference configuration, we adopt the most energy-efficient one, corresponding to C_2 ($V_R = 0.912$ V and $f_R = 400$ MHz). The plots in Figure 4 for the constant, linear and quadratic relations in R_F^V (top row and bottom-left corner) are scaled versions of that

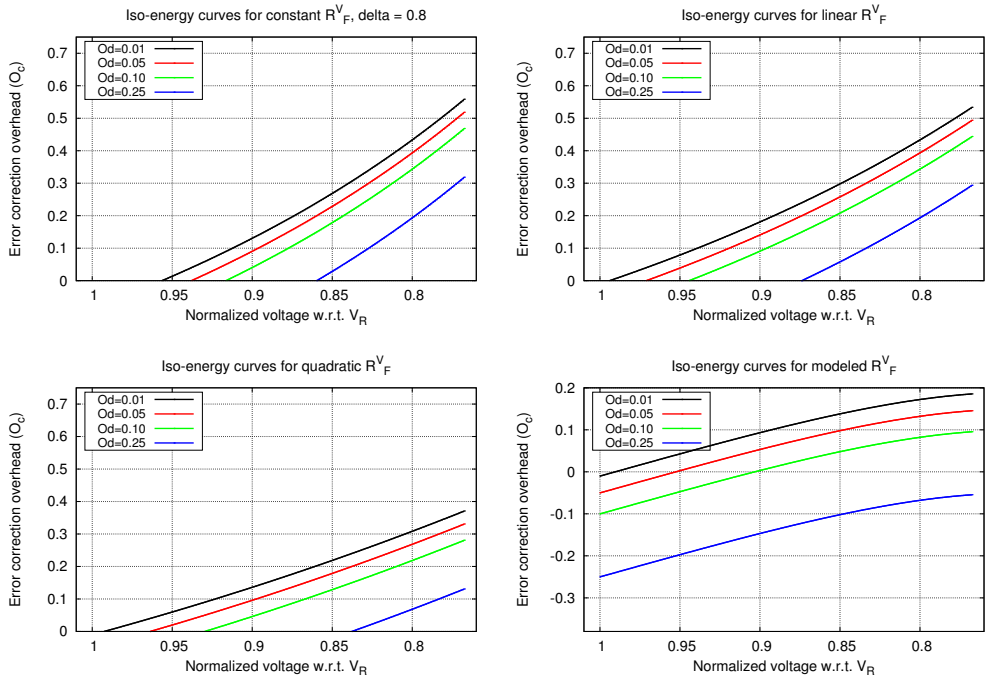

Figure 4: Configurations for the fault-tolerant version of GEMV, operating in VFS conditions in the Cortex-A15, that attain the same energy efficiency as the original routine executed in the reference configuration C_2 without errors. The coefficients for the modeled R_F^V are $\beta_1 = 1.8403$, $\beta 2 = -0.7665$, offering a relative error $\mathcal{R} = 0.1005$.

reported for the undervolting scenario, demonstrating that, under such conditions imposed on the relation between frequency and voltage, there is still hope for a strategy that tackles the energy costs of error detection+correction via software. On the other hand, the modeled relation (bottom-right corner) illustrates a more restrictive scenario, requiring a larger voltage reduction factor to compensate the detection overhead but also hinting an asymptotic limit on the iso-energy curves.

7. Concluding Remarks

Computing below the nominal voltage promises higher energy efficiency at the cost of exposing hardware unreliability to the user. In this paper, we have used three relevant kernels from dense linear algebra to investigate the

trade-offs between the energy gains due to lower power dissipation unleashed when operating at extended low-voltage margins versus the costs required to deal with soft errors (in the data) via a software resilience mechanism. We approach this question by distinguishing between the error detection overhead, which is constant and independent of the number of errors, and the error correction overhead, which is proportional to the error rate. Concretely, we rely on a simple power model that decomposes the consumption into its dynamic and static components, to then determine the error rate that an execution can accommodate while matching the energy efficiency (iso-energy) of an error-free configuration that operates above the voltage safety threshold. We recognize that a weak point of our study is the use of extrapolation to estimate the power consumption when operating at low voltage scale. To compensate for this, our extrapolation examples correspond to values that are within 85% and 95% of the lowest nominal voltage.

Our iso-energy models show that fault tolerance techniques for dense linear algebra operations, implemented in software, can complement hardware reliability in LVEM computing scenarios. Furthermore, (i) under undervolting conditions, the iso-energy models are independent of the DLA operation (algorithm); (ii) when the frequency is varied with the voltage, the same applies to the execution of a compute-bound DLA operations; and (iii) for memory-bound operations, when the frequency scaled proportionally to the voltage, it is tougher to compensate the energy costs of error detection+correction via VFS. Finally, we note that our methodology for the cases (i) and (ii) carries over to other architectures such as those from Intel and AMD designed for high performance computing.

Acknowledgments

The researchers from Universidad Jaume I were supported by project CICYT TIN2014-53495-R of MINECO and FEDER, and the FPU program of MECD. The researcher from Universitat Politècnica de Catalunya was supported by projects TIN2015-65316-P from the Spanish Ministry of Education and 2014 SGR 1051 from the Generalitat de Catalunya, Dep. d'Innovació, Universitats i Empresa.

We thank Francisco D. Igual, from Universidad Complutense de Madrid, for his help with the configuration and experimentation with the Juno (r0) system.

References

[1] G. Moore, Cramming more components onto integrated circuits, Electronics 38 (8) (1965) 114–117.

[2] M. Duranton, K. De Bosschere, A. Cohen, J. Maebe, H. Munk, HiPEAC vision 2015, `https://www.hipeac.org/publications/vision/` (2015).

[3] R. Lucas *et al*, Top ten Exascale research challenges, `http://science.energy.gov/~/media/ascr/ascac/pdf/meetings/20140210/Top10reportFEB14.pdf` (2014).

[4] J. F. Lavignon *et al*, ETP4HPC strategic research agenda achieving HPC leadership in Europe, `http://www.etp4hpc.eu/` (2013).

[5] U. Karpuzcu, N. S. Kim, J. Torrellas, Coping with parametric variation at near-threshold voltages, Micro, IEEE 33 (4) (2013) 6–14.

[6] E. L. Petersen, P. Shapiro, J. H. Adams, E. A. Burke, Calculation of cosmic-ray induced soft upsets and scaling in VLSI devices, IEEE Transactions on Nuclear Science 29 (6) (1982) 2055–2063. doi:10.1109/TNS.1982.4336495.

[7] A. H. Johnston, Scaling and technology issues for soft error rates, in: 4th Annual Research Conference on Reliability, Stanford University, 2000, pp. 1–9.

[8] V. Degalahal, N. Vijaykrishnan, M. J. Irwin, Analyzing soft errors in leakage optimized SRAM design, in: VLSI Design, 2003. Proceedings. 16th International Conference on, 2003, pp. 227–233. doi:10.1109/ICVD.2003.1183141.

[9] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, M. Snir, Toward exascale resilience: 2014 update, Supercomputing Frontiers and Innovations 1 (1) (2014) 5–28.

[10] L. Tan, S. L. Song, P. Wu, Z. Chen, R. Ge, D. J. Kerbyson, Investigating the interplay between energy efficiency and resilience in high performance computing, in: Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International, 2015, pp. 786–796. doi:10.1109/IPDPS.2015.108.

[11] F. G. Van Zee, R. A. van de Geijn, BLIS: A framework for rapidly instantiating BLAS functionality, ACM Trans. Math. Softw. 41 (3) (2015) 14:1–14:33.

[12] T. M. Smith, R. A. van de Geijn, M. Smelyanskiy, E. S. Quintana-Ortí, Toward ABFT for BLIS GEMM, Tech. Rep. TR-15-05, The University of Texas at Austin, Department of Computer Science, FLAME Working Note #76 (June 2015).

[13] A. P. Chandrakasan, S. Sheng, R. W. Brodersen, Low-power CMOS digital design, IEEE Journal of Solid-State Circuits 27 (4) (1992) 473–484. doi:10.1109/4.126534.

[14] M. Weiser, B. Welch, A. Demers, S. Shenker, Scheduling for reduced CPU energy, in: Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation, OSDI '94, USENIX Association, Berkeley, CA, USA, 1994.
URL http://dl.acm.org/citation.cfm?id=1267638.1267640

[15] G.-Y. Wei, M. Horowitz, A low power switching power supply for self-clocked systems, in: Low Power Electronics and Design, 1996., International Symposium on, 1996, pp. 313–317. doi:10.1109/LPE.1996.547531.

[16] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, T. Mudge, Razor: a low-power pipeline based on circuit-level timing speculation, in: Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on, 2003, pp. 7–18. doi:10.1109/MICRO.2003.1253179.

[17] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, S. L. Lu, Trading off cache capacity for reliability to enable low voltage operation, in: Computer Architecture, 2008. ISCA '08. 35th International Symposium on, 2008, pp. 203–214. doi:10.1109/ISCA.2008.22.

[18] A. R. Alameldeen, I. Wagner, Z. Chishti, W. Wu, C. Wilkerson, S.-L. Lu, Energy-efficient cache design using variable-strength error-correcting codes, in: Proceedings of the 38th Annual International Symposium on Computer Architecture, ISCA '11, ACM, New York, NY, USA, 2011, pp. 461–472. doi:10.1145/2000064.2000118.
URL http://doi.acm.org/10.1145/2000064.2000118

[19] A. Bacha, R. Teodorescu, Dynamic reduction of voltage margins by leveraging on-chip ECC in Itanium II processors, in: Proceedings of the 40th Annual International Symposium on Computer Architecture, ISCA '13, ACM, New York, NY, USA, 2013, pp. 297–307. doi:10.1145/2485922.2485948.
URL http://doi.acm.org/10.1145/2485922.2485948

[20] J. Leng, A. Buyuktosunoglu, R. Bertran, P. Bose, V. J. Reddi, Safe limits on voltage reduction efficiency in GPUs: A direct measurement approach, in: Proceedings of the 48th International Symposium on Microarchitecture, MICRO-48, ACM, New York, NY, USA, 2015, pp. 294–307. doi:10.1145/2830772.2830811.
URL http://doi.acm.org/10.1145/2830772.2830811

[21] D. Ibtesham, D. DeBonis, D. Arnold, K. B. Ferreira, Coarse-grained energy modeling of rollback/recovery mechanisms, in: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2014, pp. 708–713. doi:10.1109/DSN.2014.71.

[22] B. Mills, T. Znati, R. Melhem, K. B. Ferreira, R. E. Grant, Energy consumption of resilience mechanisms in large scale systems, in: 2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2014, pp. 528–535. doi:10.1109/PDP.2014.111.

[23] L. Tan, Z. Chen, S. L. Song, Scalable energy efficiency with resilience for high performance computing systems: A quantitative methodology, ACM Trans. Archit. Code Optim. 12 (4) (2015) 35:1–35:27. doi:10.1145/2822893.
URL http://doi.acm.org/10.1145/2822893

[24] F. G. V. Zee, T. M. Smith, B. Marker, T. M. Low, R. A. V. D. Geijn, F. D. Igual, M. Smelyanskiy, X. Zhang, M. Kistler, V. Austel, J. A. Gunnels, L. Killough, The BLIS framework: Experiments in portability, ACM Trans. Math. Soft. 42 (2) (2016) 12:1–12:19.

[25] T. M. Smith, R. van de Geijn, M. Smelyanskiy, J. R. Hammond, F. G. Van Zee, Anatomy of high-performance many-threaded matrix multiplication, in: Proc. IEEE 28th Int. Parallel and Distributed Processing Symp., IPDPS'14, 2014, pp. 1049–1059.

[26] S. Catalán, F. D. Igual, R. Mayo, R. Rodríguez-Sánchez, E. S. Quintana-Ortí, Architecture-aware configuration and scheduling of matrix multiplication on asymmetric multicore processors, Cluster Computing 19 (3) (2016) 1037–1051. doi:10.1007/s10586-016-0611-8.
URL http://dx.doi.org/10.1007/s10586-016-0611-8

[27] E. Anderson et al, LAPACK Users' guide, 3rd Edition, SIAM, 1999.

[28] G. H. Golub, C. F. V. Loan, Matrix Computations, 3rd Edition, The Johns Hopkins University Press, Baltimore, 1996.

[29] R. Gensh, A. Aalsaud, A. Rafiev, F. Xia, A. Iliasov, A. Romanovsky, A. Yakovlev, Experiments with the Odroid-XU3 board, Tech. Rep. CS-TR-1471, Newcastle University (May 2015).

[30] P. Alonso, R. M. Badia, J. Labarta, M. Barreda, M. F. Dolz, R. Mayo, E. S. Quintana-Ortí, R. Reyes, Tools for power-energy modelling and analysis of parallel scientific applications, 41st International Conference on Parallel Processing – ICPP (2012) 420–429.

[31] J. L. Hennessy, D. A. Patterson, Computer Architecture: A Quantitative Approach, 5th Edition, Morgan Kaufmann Pub., 2012.