IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 3 March 2025, accepted 21 March 2025, date of publication 26 March 2025, date of current version 7 April 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3555119

RESEARCH ARTICLE

Toward Greener Matrix Operations by Lossless Compressed Formats

FRANCESCO TOSONI[1], PHILIP BILLE[2], VALERIO BRUNACCI[3], (Graduate Student Member, IEEE), ALESSIO DE ANGELIS[3], (Member, IEEE), PAOLO FERRAGINA[1,4], AND GIOVANNI MANZINI[1]

[1]Computer Science Department, University of Pisa, 56127 Pisa, Italy
[2]DTU Compute, Technical University of Denmark, 2800 Kongens Lyngby, Denmark
[3]Engineering Department, University of Perugia, 06125 Perugia, Italy
[4]L'EMbeDS Department, Sant'Anna School of Advanced Studies, 56127 Pisa, Italy

Corresponding author: Francesco Tosoni (francesco.tosoni@di.unipi.it)

This work was supported in part by European Union-Horizon 2020 Program through the Scheme "INFRAIA-01-2018-2019—Integrating Activities for Advanced Communities, SoBigData++: European Integrated Infrastructure for Social Mining and Big Data Analytics" (http://www.sobigdata.eu) under Grant 871042; in part by the NextGenerationEU—National Recovery and Resilience Plan (Piano Nazionale di Ripresa e Resilienza—PNRR) through the Project "SoBigData.it-Strengthening the Italian RI for Social Mining and Big Data Analytics" under Grant IR0000013-Avviso (3264 del 28/12/2021); in part by the Spoke "Future HPC and BigData" of the ICSC-Centro Nazionale di Ricerca in High-Performance Computing, Big Data and Quantum Computing funded by European Union-NextGenerationEU-PNRR; and in part by the Independent Research Fund Denmark under Grant DFF-9131-00069B.

ABSTRACT Sparse matrix-vector multiplication (SpMV) is a fundamental operation in machine learning, scientific computing, and graph algorithms. In this paper, we investigate the space, time, and energy efficiency of SpMV using various compressed formats for large sparse matrices, focusing specifically on Boolean matrices and real-valued vectors. Through extensive analysis and experimentation on server and edge devices, we observed that different matrix compression formats entail distinct trade-offs among space efficiency, execution time, and energy consumption. Notably, selecting the appropriate compressed format can reduce energy consumption by an order of magnitude on both servers and single-board computers. Moreover, our experiments reveal that while data parallelism can improve execution speed and energy efficiency, optimizing both simultaneously poses interesting challenges. Specifically, we demonstrate that for certain compression schemes, the optimal degree of parallelism for minimizing execution time does not coincide with that for energy efficiency. This finding aligns with recent results in the literature, thus challenging the prevailing assumption of a straightforward linear correlation between execution time and energy usage. Overall, our findings advocate for software engineers to adopt compressed data structures as important instruments in the development of more energy-efficient software components, extending beyond SpMV.

INDEX TERMS Green computing, lossless compression techniques, compressed matrix formats, matrix-to-vector multiplications, computation-friendly compression, PageRank.

I. INTRODUCTION

The emergence of Machine Learning (ML) and, in particular, Deep Learning algorithms has brought about significant advancements in computer vision (e.g., image classification [1], [2]), natural language processing (e.g., text classification [3], [4], [5]), speech recognition, information retrieval, and more. These advancements have led to increased network complexity, more parameters, greater demands for training resources, and longer prediction latency.

Moreover, with the generation of vast amounts of data that exceed Moore's Law [6], [7], the storage and processing

The associate editor coordinating the review of this manuscript and approving it for publication was Zhengmao Li.

© 2025 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

of these data for data science applications have become critical concerns. This is particularly evident in the context of Graph Databases (GraphDBs) [8], [9], which are emerging as the primary software architecture for applications requiring interconnected data modeling (i.e., data represented as graphs).[1] Graphs are essential for network analytics and are increasingly vital in the context of LLMs due to the advent of Retrieval-Augmented Generation, which enhances LLMs through the effective use of knowledge graphs [11], [12], [13]. Efficiently storing and processing large graphs in terms of time and space remains a significant challenge.

For this reason, sustainability-oriented practices that leverage industrial big data to drive product innovation are gaining importance. Industrial big data plays a mediating role in the relationship between sustainability orientation and product innovation [14]. Additionally, Green Computing and Green Software Engineering have emerged in response to sustainability challenges, prompting a paradigm shift that both integrates and, in some aspects, diverges from traditional approaches to the design, use, and management of computer hardware and software.

Recent studies [15] estimate that the ICT sector currently contributes 2.1−3.9% of global greenhouse gas (GHG) emissions, with Machine Learning being a significant contributor [16]. Energy waste extends beyond data centers to small-scale edge and Internet of Things (IoT) devices, where battery life is a critical constraint [17], [18]. Consequently, growing awareness of energy consumption has fueled interest in the so-called *green computing*, particularly in AI-enabled IoT infrastructures (AIoT). By ''greening data structures and algorithms,'' software engineers aim to reduce resource usage, lower power consumption, and extend battery life through more efficient algorithm design. However, as noted in [19], programmers often lack experience with software-induced energy consumption, despite its increasing importance as a design constraint.

In this paper, we draw attention to space–time and energy-efficient representations of large matrices on a server machine and a resource-constrained single-board computer. Indeed, the problem of managing scalability challenges for storage, operation, and bandwidth, is relevant in server-client transmissions and local computations. For example, in ML applications although training costs can be one-time, or even free with pre-trained models, deploying and running their inference capabilities over an extended period often results in expensive matrix-based computations for client-side RAM and CPU of edge and IoT devices.

Compressed data structures provide a natural solution to mitigate storage occupancy while enabling efficient data transmission and offering opportunities for faster processing. Indeed, computation-friendly compressed representations [20] allow for indexing, querying, processing, and manipulating data directly in their compressed format without requiring any prior decompression, thereby saving memory usage and speeding up computations at run time.

However, despite the critical role of matrix-to-vector computations in scientific computing, efficient machine learning inference, graph analysis, and more, to our knowledge, the only study that examines the actual impact of compressed data structures on energy consumption is that of Fuentes-Sepúlveda and Ladra [18], which focuses on compact integer representations. Given the above discussion, we believe that an analysis of compressed matrix representations from a green viewpoint may provide software engineers with a practical guide that can allow them to select the most energy-efficient matrix format based on the time and space constraints of their problem. As a side outcome, with our paper, we aim to advance the knowledge of the relationship between lossless compression and energy consumption.

II. OUR CONTRIBUTION

In this paper, we focus specifically on the space, time, and energy efficiency of matrix-to-vector multiplications using well-known compressed formats for large sparse binary matrices. Sparse matrix-vector multiplication (SpMV) is central to scientific computing applications, as it underpins the solution of sparse linear systems and sparse eigenvalue problems through iterative methods. Additionally, SpMV is essential for graph-analysis algorithms such as PageRank and centrality algorithms and is relevant for inference in binary machine learning models.

The primary research questions we address in this paper are as follows:

- Q1 For matrix-vector multiplication, does compression lead to savings in space, time, and energy?
- Q2 Are time-optimized compressed matrix formats the most energy-efficient, and vice versa?
- Q3 In the context of compressed matrix-vector multiplications, which runtime metrics (such as CPU cycles, number of instructions, and completion time) most significantly impact energy efficiency?

In this study, we focus on the problem of efficient matrix-to-vector multiplications, specifically when the matrix is sparse and Boolean, while the vector is real-valued. Computing the product of a binary adjacency matrix from a large graph with a real-valued vector is a crucial operation in various graph analytics tasks. Among these tasks, we consider the computation of the well-known PageRank score, which assesses the relevance of a Web page (see Section IV). This score has applications beyond the Web domain, and extensive datasets are available for evaluating performance in real-world scenarios.

We believe that the findings presented in this article will be of interest not only to the Network Analytics community but also to software engineers seeking efficient vector multiplications over binary and ternary matrices like the ones that occur in highly quantized LLMs. For instance, recent large language models (LLMs) such as BitNet [21],

[1]The global Graph Database market was valued at $2.33 billion in 2023 and is projected to reach $14.58 billion by 2032 [10].

which introduces a 1-bit Transformer architecture, present opportunities for leveraging compact binary formats to optimize sparse matrix-vector multiplication (SpMV) in terms of space, time, and energy efficiency. Similarly, ternary quantization techniques [22], [23], [24], which encode values as $(-1, 0, +1)$ and employ binary matrix representations, replace FP32 multiplications with more efficient additions and subtractions [25]. These approaches offer a favorable trade-off between space and accuracy and can benefit from compressed binary matrix formats by deploying two matrices to represent the two non-zero values of the quantization. This is particularly relevant in federated learning [26] because it enables utilizing resource-constrained nodes such as smartphones or IoT devices, where battery life is a critical concern [27].

Further, the inference and training stages of Graph Neural Networks (GNNs) are often dominated by the time required to compute lengthy sequences of matrix multiplications involving the sparse graph adjacency matrix and its embeddings. Motivated by this challenge, the authors of [28] recently published preliminary results showcasing efficient matrix multiplication kernels based on new compressed formats for binary matrices. However, their work remains largely in the early stages, and its scalability has yet to be validated, as the authors tested their results solely on adjacency matrices derived from networks of scientific co-authors and citations, which are relatively small in scale. The largest graph they evaluated contains only 540,486 nodes. In contrast, this paper considers matrix formats capable of processing datasets with millions of nodes, even on resource-constrained devices like Raspberry Pis (for instance, our dataset `indochina-2004` has 7,414,866 nodes).

From a technical perspective, we address the problem of compressing Boolean sparse matrices for subsequent fast matrix-to-vector multiplications and extend the results in [29] by experimentally analyzing three distinct matrix-compression formats: the grammar-based methods described in [30], the k^2-tree [31], and Zuckerli [32], which is an enhanced version of WebGraph [33]. The appeal of these formats stems from their ability to achieve significant space reduction on disk by exploiting data patterns beyond mere sparsity while ensuring that time performance scales with the size of the compressed matrix representation. It is important to note that dense formats requiring quadratic space would impose prohibitive space and time demands, even for matrices containing millions of nodes.

In this paper, we have adapted each compression format to support matrix-to-vector multiplication across multiple threads. For each dataset, we measured the running time and energy consumption of the matrix-to-vector multiplications by executing the PageRank algorithm [34]. We conducted our tests on two platforms: a multi-core Intel® Core™ i9-7960X desktop server utilizing up to 32 threads, with energy performance assessed via a power meter and the Intel's Running Average Power Limit (RAPL) [35], and a resource-constrained edge device (Raspberry Pi) using up to 8 threads, measuring energy consumption with a benchtop power meter. This way, we are deploying a more robust and reliable energy measurement setup for our compressed matrix formats that incorporates Intel's RAPL for comparison (as used in [18] for compressed integer representations) but also adopts power meters as ground truth for energy estimation, thus addressing RAPL's accuracy limitations highlighted in the recent literature (see Section III-B and Section VI-F).

We summarize the main contributions of our paper as follows (additional details and contributions are provided in the next sections):

- In contrast to [29], where the authors experimented with matrix multiplications using the WebGraph [33] format, we evaluate a broader and more recent set of matrix representations (see Section V). The tested formats are *computation-friendly*, as their compression strategies go beyond mere sparsity, enabling direct operations on the data without prior decompression and allowing computations in time proportional to the size of the compressed representation. Additionally, we assess the impact of compression on energy consumption, testing not only single-threaded versions as in [29] but also multi-thread implementations, via a robust and reliable energy measurement setup which deploys power meters besides Intel's RAPL.
- Our analysis involves collecting metrics and comparing results across two platforms: a multi-core Intel® Core™ i9-7960X desktop server and a resource-constrained Raspberry Pi 4 single-board computer. For the server, we employ the Intel RAPL software profiler and an external power meter to measure power consumption, while for the Raspberry Pi 4, we use a digital multimeter to obtain precise supply current measurements. This approach ensures that our findings remain robust across different platforms and are relevant to diverse communities, including those focused on data center optimization and energy-efficient edge computing. Our examination of the energy performance of the PageRank algorithm on both platforms reveals that a careful selection of the lossless compression format for the involved matrices can consistently yield energy savings of one to two orders of magnitude across all tested datasets and platforms; these findings thus carry significant positive implications for energy-critical applications relying on matrix-based computations.
- We present an implementation of a multithreaded matrix-to-vector multiplication algorithm based on the recently introduced Zuckerli compression format. This format enhances the WebGraph algorithm by improving compression ratios while maintaining resource usage for decompression comparable to WebGraph. Ours is the first attempt to utilize the Zuckerli format for matrix-vector calculations, thereby offering independent results that shed light on the potential of this format for applications involving (large) matrix computation kernels, such as PageRank.

- Regarding question Q1, we observe that while the Zuckerli format and the k^2-tree are more space-efficient, they are generally slower and tend to consume about an order of magnitude more energy than the grammar-based solutions proposed in [30]. Our results highlight a favorable trade-off between time and energy consumption for the k^2-tree and Zuckerli, thus extending recent studies [36], [37], [38] to these other resources.
- Regarding question Q2, the findings on the tested compressed matrix formats align with recent research on data compression that shed light on the non-energy-proportionality of multicore processor computing; see, e.g., [39] and [40]. Our results confirm, from a data compression perspective, that the oversimplified energy model —which assumes that energy consumption increases linearly with completion time— does not accurately represent energy consumption in modern computer architectures. By plotting running time and energy consumption as a function of the thread count, we found that the optimal degree of parallelism for energy efficiency is sometimes lower than that for time efficiency. This suggests that scaling energy consumption is more challenging than scaling execution time. Consequently, in certain situations, software engineers should consider accepting suboptimal time performance to achieve energy-optimal executions.
- In response to question Q3, our findings indicate that the number of L1 and L3 cache operations is crucial for both time and energy efficiency in data compression algorithms. Inefficient utilization of the cache hierarchy can degrade instruction throughput (instructions per clock cycle), leading to increased latencies and energy inefficiencies. This observation aligns with the findings in [41], where the authors argued that primary CPU activities for energy-related estimations (referred to as prime Performance Monitoring Counters, or PMCs, for short) are those related to cache, branch instructions, microoperations (μops), floating-point instructions, instruction decode queues and cycles.

The remainder of the paper is structured as follows. In Section III we discuss related works. In Section IV we describe the PageRank algorithm and how we implemented it in our experiments. In Section V we discuss compressed matrix representations and we justify the choice we made for our experiments. Section VI describes our experimental setup, and Section VII illustrates and discusses the results of our experiments. Section VIII summarizes the lessons learned from our analysis and outlines suggestions for future work.

III. RELATED WORK

A. GREEN SOFTWARE ENGINEERING

As energy consumption becomes a critical design constraint for computing devices, hardware engineers and system designers continue to explore innovative approaches to mitigate the energy demands of matrix-based computations. Integrating energy management principles into application development and algorithm design demonstrated that energy-aware hardware and software can significantly reduce the energy consumption of legacy applications [42]. Nevertheless, though significant progress has been achieved in energy optimization through advancements in hardware design and modeling [43], [44], recent studies, such as the case study on energy consumption in compact integer vectors by [18], highlight that green software engineering remains in its infancy and that limited attention has been devoted to sustainable software practices, particularly in core areas of computer science such as algorithm design. This underscores the need for further research to establish best practices in this setting, and for energy-efficient compressed data structures in particular.

The success of ChatGPT has significantly popularized machine learning algorithms, while also drawing attention to the environmental and economic costs associated with their use [45]. For instance, training the GPT-3 model, which boasts 175 billion parameters, incurs millions of dollars per iteration [3], not to mention the substantial computational expenses involved in experimenting with various hyperparameters. A 2023 estimate from OpenAI's CEO [46] placed the training cost for GPT-4 at approximately 100 million dollars, highlighting the urgent need for innovative and sustainable approaches to AI development. Furthermore, a 2025 estimate [47] indicates that even DeepSeek presents challenges for energy consumption: in response to a prompt asking whether it is acceptable to lie, DeepSeek generated a 1,000-word answer that consumed 17 800 joules of energy—about 41% more than a comparable model from Meta for the same prompt. Overall, tests on 40 prompts revealed that DeepSeek utilized 87% more energy than the Meta model.

This paper explores energy conservation through green algorithm design/engineering and green software development —i.e., software with measurable energy performance that reduces resource usage compared to unoptimized legacy code. Green software employs energy-consumption models [48] to predict and measure power usage, helping developers minimize energy consumption. Early models, such as the simplistic energy complexity model proposed by [49], expressed energy consumption asymptotically as $T + (PB) \cdot I$, where T is completion time, I represents the number of parallel I/Os, P denotes the number of memory banks, and B signifies block size. However, such models fail to capture the complexities of modern architectures, particularly features like contention among multiple threads or energy-expensive data translation look-aside buffer (dTLB) accesses [41]. Contemporary research reveals that energy consumption in multiprocessor environments is not proportional to time complexity [39], [40], paving the way for time-energy bicriteria optimizations. For instance, [50] showed that imbalanced partitioning strategies in parallel applications can effectively reduce completion times; yet, the authors don't discuss the power consumption of their approach. In this paper, we aim to contribute to this

research area by demonstrating that bicriteria energy-time optimizations can be achieved through a careful selection of the compressed matrix formats that are most suitable for the problem at hand.

An essential component of green software design is the establishment of accurate frameworks to evaluate and compare the energy costs of algorithmic solutions. The Software Energy and Resource Efficiency Analysis (SERENA) [51], the Green Software Measurement Process (GSMP) [52], the Sustainability Assessment Framework (SAF) Toolkit,[2] the Green Metrics Tool,[3] the Cloud Energy Usage Estimation Model [53], the Software Footprint,[4] the Emission Estimation Framework by the Sustainable Digital Infrastructure Alliance (SDIA) [54] and the Container Overhead Measurement Methodology by [55] are tools that enable researchers to measure and mitigate the environmental impact of software. For further discussion on these frameworks, refer to [48].

Despite progress in this field, a lack of consensus and awareness of energy-related best practices persists. As [56] noted, business motivations and short-term thinking often hinder sustainability efforts. Additionally, power-oriented source code optimizations, as discussed by [57], may be impractical for large-scale software, underscoring the need for guaranteed energy-efficient solutions that are reusable and straightforward to integrate. [48] has hence recently introduced the Green Software Measurement Model (GSMM), accompanied by a glossary based on the ontology presented by [58]. The model, reflecting a decade of research by teams from four countries, stressed the importance and invited further contributions from researchers and green software engineers, motivating us to provide additional insights on the relationship between compact data structure and energy efficiency.

B. ENERGY PROFILING

Accurate energy profiling during application execution is essential for developing effective strategies that minimize energy consumption at the software level. Research [59] indicates that relying on imprecise energy measurement tools in optimization methods can lead to significant inefficiencies. The aforementioned GSMM model [48] underscores the critical role of reliable measurement systems in obtaining accurate energy profiling estimates. It also emphasizes the importance of selecting appropriate metrics —such as CPU and GPU usage, execution duration, mean power draw, energy consumption, and network traffic— to construct a comprehensive model of an application's energy usage.

According to the literature [41], [59], the primary approaches for energy measurement include system-level physical power measurements using external power meters, on-chip power sensors, and energy predictive models. We will now briefly discuss each of these methods.

a: SYSTEM-LEVEL PHYSICAL POWER MEASUREMENTS USING EXTERNAL POWER METERS

Methods based on power meters are widely regarded as the most accurate [41], [59] and serve as the ground truth for energy profiling. However, they are limited to system-level measurements and cannot decompose energy consumption at the component level, a significant limitation for hybrid applications leveraging multiple devices.

Yet, recent research [60] introduces methodologies for measuring component-level energy consumption in hybrid applications on heterogeneous platforms. Further, custom instrumentation systems, such as PowerMon [61], PowerPack [62], and PowerInsight [63], offer fine-grained measurements, but face challenges related to temporal resolution, topological granularity, and the need for specialized hardware expertise [59].

b: ON-CHIP POWER SENSORS

On-chip power sensors, accessible via vendor-specific libraries, provide another dominant approach, offering fine-grained, component-level energy breakdowns. Such sensors are standard in many processors, including Intel and AMD multicore CPUs, Nvidia GPUs, and Intel Xeon Phis. For instance, Intel devices offer Intel's RAPL [64], [65], while AMD CPUs utilize Application Power Management (APM) [66]. RAPL estimates energy consumption across domains like core, package, and DRAM using Model Specific Registers (MSRs).

In earlier processor generations, such as Sandy Bridge and Ivy Bridge E5, RAPL relied on performance monitoring counters (PMCs) to estimate energy consumption [67]. By contrast, newer processors like Haswell and Skylake use independent voltage regulators (VR IMON) for power estimation in both CPU and DRAM domains [68, §2].

RAPL has also been utilized in research such as [18], which evaluates the energy-saving impact of integer compression and has served as a cornerstone of numerous power-measurement frameworks over the past decade. These frameworks include Scaphandre,[5] CodeCarbon,[6] and the Green Metrics Tool (GMT), developed by Green Coding Berlin, as referenced in [48].

Nonetheless, while some studies [35], [69] assert that RAPL provides reasonably accurate measurements, others [59], [70], [71] report significant prediction errors, with inaccuracies reaching up to 150%. According to [41], these discrepancies arise from the complexity of modern multicore CPU architectures, which contend with challenges such as severe resource contention, NUMA-induced memory latency variations, and energy dissipation across multiple

[2]https://github.com/S2-group/SAF-Toolkit
[3]https://github.com/green-coding-solutions/green-metrics-tool
[4]https://www.oeko.de/blog/energieverbrauch-von-software-eine-anleitung-zum-selbermessen/
[5]https://web.archive.org/web/20230623133912/https://01.org/blogs/2014/running-average-power-limit-%E2%80%93-rapl
[6]https://github.com/mlco2/codecarbon

power domains (e.g., CPU sockets and DRAM) under dynamic power management. Consequently, the RAPL-based approach adopted by [18] may raise concerns about measurement accuracy. Therefore, in our study, we complement RAPL estimates with a more reliable system-level methodology based on power meter measurements; see Section VI-F.

c: ENERGY PREDICTIVE MODELS

Alternative power estimation methods rely on PMCs and linear predictive models. PMCs are specialized hardware registers in modern processors that record counts of software events (e.g., page faults, context switches) and hardware events (e.g., CPU cycles, branch instructions, cache misses). However, as [66] notes, a major limitation of these models is their inability to capture all PMCs simultaneously. Since collecting exhaustive PMC data during a single application run is often impractical, researchers typically employ heuristic techniques such as principal component analysis, expert judgment, or energy conservation laws to identify the most relevant PMC subsets. Such approaches facilitate the construction of simpler, yet reliable energy estimation models [41]. In this paper, we resolved not to experiment with this class of energy profilers, as our experimental setup already incorporates system-level physical measurements through power meters, which are widely regarded as the most accurate method for energy profiling to date [41], [59].

Hybrid models, such as those in [66] and [72], integrate utilization data with PMCs to more effectively capture energy-intensive activities during application execution. These hybrid approaches significantly improve prediction accuracy compared to models based on a single variable type. Nonetheless, recent studies [60] highlight persistent limitations; in particular, it has been argued that such models cannot comprehensively capture complex architectures [70], [71] and that they fail at accounting for the energy costs of disk and network I/O operations [66].

Software-based energy predictive models have gained widespread adoption as an alternative for estimating energy consumption. Most of these models are linear, as recommended by [59], due to the absence of rigorous theoretical and experimental validation for non-linear models and typically deploy PMCs as their primary predictor variables.

Summarizing the discussion of the previous subsections, we can state that our study aligns with the recommendations of [59] by using power meters as the benchmark for energy estimations; yet, we also report and comment on RAPL estimates, obtained via the Linux Perf tool [73], a comprehensive profiling suite (aka, `perf_events`) that facilitates the collection of various PMCs. This will allow us to adopt a robust and reliable energy measurement setup for our compressed matrix formats, that offers additional insights into energy consumption patterns (cf. Section VI-F).

C. STATIC AND DYNAMIC ENERGY

Energy consumption can be divided into two primary components: static and dynamic. *Static energy* is computed by multiplying the platform's idle power (assumed constant over time) by the application's execution time. In contrast, *dynamic energy* varies temporally and is obtained by subtracting the static energy from the total energy consumed. Formally, let P_{idle} represent the idle (or static) power consumption, E_T the total energy consumed during execution, and T the application's execution time in seconds. As outlined in [41] and [59], the dynamic energy E_D is expressed as

$$E_D = E_T - P_{\text{idle}} T \tag{1}$$

Now let $p(t)$ denote the instantaneous power supplied to the device at time t; and let $i(t)$ denote the instantaneous current. Assuming that the supply voltage V remains constant over time, let I_{idle} denote the idle current; then, we can express E_D as

$$E_D = \int (p(t) - P_{\text{idle}})\ dt\ = V \int (i(t) - I_{\text{idle}})\ dt \tag{2}$$

In practical scenarios, stochastic noise, such as Gaussian noise $\epsilon_N \sim \mathcal{N}(0, \sigma^2)$, may introduce additive errors. Consequently, the measured dynamic energy can be represented as $\tilde{E}_D = E_D + \epsilon_N$. To reduce the impact of such fluctuations, in Section VI-F we repeated each experiment multiple times, averaging the resulting power data.

Unlike the approach in [18], which focuses on estimating the total energy consumption of the tested platform, this study emphasizes dynamic energy consumption. Our experimental results confirm that the platforms and applications evaluated in this work exhibit dynamic energy consumption that dominates the static component which is a platform-intrinsic and time-invariant property.

D. PARALLEL PROGRAMMING

Parallel programming is widely recognized for its potential to mitigate power consumption. Most contemporary computer systems are based on CMOS (Complementary Metal-Oxide Semiconductor) technology, wherein dynamic power consumption scales proportionally to the cube of the operating voltage. Consider a scenario where a software application achieves a 1.5× speedup using two processing cores. This speedup can be leveraged by software engineers to either reduce latency or conserve power [74, §2.5.3]. Assuming latency requirements are already satisfied, reducing the clock rate by 1.5× can result in substantial energy savings. Denoting by $\mathcal{P}_1$ the energy consumption of the single-threaded application and by $\mathcal{P}_2$ that of the data-parallel version, the following relationship holds:

$$\mathcal{P}_2 = 2\left(\frac{1}{1.5}\right)^3 \approx 0.6\,\mathcal{P}_1 \tag{3}$$

where the factor of 2 accounts for the use of two cores. Another common energy-saving strategy in parallel computing is to complete computations as quickly as possible, enabling all threads to transition into a low-power *sleep mode*, which consumes significantly less energy [74, §2.5.3]. The

authors of [57] demonstrate that idle or sleep states can achieve up to 19% energy savings and can be seamlessly integrated with other energy-saving techniques. They further suggest that increasing parallelism from 1 to 16 threads can reduce energy consumption for many applications, even those with poor scalability. This finding is particularly relevant for mobile devices, where parallelism can decrease the time the device remains active (e.g., with the display and other components powered on), enhancing user experience and reducing energy consumption.

The findings of [42] further support the notion that increased parallelism not only reduces execution time but also lowers energy consumption. Their study evaluates algorithmic patterns for energy-efficient design, highlighting that some patterns yield greater energy savings than others. For instance, the authors emphasize prioritizing parallel reads over writes. However, while data parallelism can enhance both performance and energy efficiency, achieving an optimal balance between the two presents intriguing challenges, as our experimental results will confirm.

To capitalize on the energy-saving potential of parallel computing in green algorithm engineering, we have implemented and tested data-parallel versions of all code; see Section VI-E.

IV. THE PAGERANK ALGORITHM

We selected the PageRank algorithm [75, §21.2] as an example of an important algorithm involving a sparse and usually highly compressible matrix. The PageRank algorithm was first used by Google's search engine to evaluate the relevance of Web pages, modeled as vertices in a Web graph.

Given a directed graph $G = (V, E)$ modeling the Web with $n = |V|$ vertices (pages) and $m = |E|$ edges (links), let A be its adjacency matrix, where $A_{uv} = 1$ if and only if there is a link from vertex u to vertex v; $A_{uv} = 0$ otherwise. The normalized adjacency matrix M is defined as $M = D^{-1} \cdot A$, where D is a diagonal matrix whose diagonal elements $d_u = \sum_v A_{uv}$ contain the out-degree of each vertex u. M is the so-called random-walk matrix, where a random walker at vertex u jumps to a neighbor v of u with probability $1/d_u$.

The graph G may have dangling vertices, namely vertices having no outward edges. To avoid a random walker getting stuck in such vertices, PageRank adds all possible outward edges to all graph vertices. This is equivalent to substituting all empty rows of M with the vector $\vec{1}/n$. This modification ensures that M is a *stochastic matrix*, meaning that for each matrix row i, it holds that $\sum_{j=1}^{n} M_{ij} = 1$. Moreover, for convergence reasons, PageRank introduces the *teleport* step: When reaching a vertex, the random walker selects with some fixed probability (say $0 < \alpha < 1$) whether it has to continue by traversing a graph edge or jump to a random vertex in the graph. In their original paper [34] Brin and Page suggest setting $\alpha = 0.15$. With the above notation, the PageRank vector giving the relevance of each page is computed by the unique limit of the sequence [76] of the probability distribution $\vec{\pi}_t$

$$\vec{\pi}_t = \alpha \cdot \vec{\pi}_0 + (1 - \alpha) \cdot \vec{\pi}_{t-1} \cdot M \tag{4}$$

In [34], the initial probability distribution $\vec{\pi}_0$ is set to $\vec{1}/n$; however, it can be any non-zero vector and still guarantee convergence to the same PageRank vector.

A. IMPLEMENTATION DETAILS

In PageRank implementations, the iteration (4) is repeated until the Euclidean norm of the difference between two consecutive estimates $\|\vec{\pi}_t - \vec{\pi}_{t-1}\|$ goes below an assigned threshold. The convergence of the method is guaranteed by the mathematical properties of the iteration matrix. Note that we do not explicitly build the matrix $M = D^{-1}A$ but, instead, we operate with the adjacency matrix A of the input graph. The diagonal matrix D is represented by an array $O[1, n]$, where the entry $O[u]$ represents the out-degree d_u of vertex u.

Formula (4) involves left products, but since most compressed matrix representations are row-oriented, meaning they represent matrices in row-major order, it is a standard practice [29] to transpose (4) so that the left matrix-vector product becomes a right product involving $M^T = A^T D^{-1}$. Note that $\left(D^{-1}\right)^T = D^{-1}$, since D is a diagonal matrix.

V. COMPRESSED MATRIX REPRESENTATIONS

A. WEBGRAPH AND ZUCKERLI

Many adjacency matrices of Web graphs contain redundancies and dependencies that are exploited by specialized compression tools to reduce their storage requirements. The first dedicated compression method for Web-related graphs, WebGraph [33], was introduced over two decades ago. It compresses adjacency lists by leveraging similarities among the lists of proximate vertices within a graph. For instance, Web pages within the same domain often share outgoing links to a common set of destination pages. These shared links can be represented once in a reference list, which subsequent lists can copy. WebGraph takes advantage of this copying property by compressing each adjacency list relative to a reference list. Initially implemented in Java, WebGraph has since been reimplemented in C++ [77], and more recently, its original authors released a Rust implementation [78].

Google's Zuckerli [32] is a more recent compression method for Web graphs that builds upon WebGraph by incorporating advanced compression techniques and novel heuristics, effectively enhancing compression efficiency. Zuckerli offers two types of compressed representations: one using Asymmetrical Numeral Systems (ANS) [79] encoding for optimized storage, and another using Huffman encoding [80, §12.1], [81, §16.3] for rapid, direct access to individual adjacency lists without requiring full decompression of the graph.

The authors of [29] noted that these compressed graph representations provide a computation-friendly framework for efficient right matrix-to-vector multiplications. The core idea is to exploit the copy property between the adjacency

lists of a vertex v_i and its reference vertex v_{r_i}. This approach allows the computation of $v_i \cdot x$ by reusing the result of $v_{r_i} \cdot x$ and adding the contributions of the differences between the adjacency lists of v_i and v_{r_i}. Building on this idea, we implemented matrix-to-vector multiplications directly on Zuckerli-compressed matrices.

Experimental results from [82], which evaluated Zuckerli with its default compression settings on standard Web graph datasets (cf. Section VI-B) demonstrate that it either outperforms or closely matches in space the most compressed parametrization of WebGraph while delivering one to three times faster access query execution. Consequently, we utilized the code available in Google's GitHub repository (https://github.com/google/zuckerli) and selected Zuckerli's default compression setting as a representative Web matrix compression technique for our experiments.

B. K2-TREE

The k^2-tree [31] and [20, §9.2.1] is a compact data structure designed to compress adjacency matrices by exploiting the sparsity and clustering of 1's. The data structure underneath the k^2-tree is a k^2-ary tree where each node represents a submatrix of the adjacency matrix. For simplicity, the size is zero-padded to the next power of k^2. The tree root represents the whole matrix of size n^2 and each of its children represents a submatrix of size n^2/k^2. If a submatrix is empty (i.e., it contains only 0's), the corresponding node is represented by a bit set to 0. In all other cases, the corresponding node is represented by a bit set to 1. Once the first level is built, the procedure continues recursively only into the submatrices with 1's, by splitting each of them into k^2 smaller submatrices. Thus, the height of a k^2-tree is always $1+\log_k n$, independently of the specific distribution of 1's and 0's within the input matrix; the access to a single cell costs $\mathcal{O}\left(\log_k n\right)$. Note that a submatrix is not split further when full of 0's.

Figure 1 depicts an a 8×8 binary matrix (left) and the corresponding k^2-tree for $k = 2$ (right). Empty submatrices are highlighted in color, with the same color used to mark the zeros representing these submatrices in the k^2-tree. Note that pointers are shown for illustration purposes only, as the actual k^2-tree is pointerless, with tree navigation operations relying on `rank` queries over the bit array encoding the internal nodes. Each tree level stores the nonempty submatrices of a given size in row-major order. Higher levels correspond to larger submatrices, with the root representing the original matrix and the leaves corresponding to individual entries.

The k^2-tree supports many operations matrix-to-matrix additions and multiplications directly in the compressed domain, yielding the resultant matrix in its compressed k^2-tree representation [36], [37]. In this paper, we focus on the PageRank computation for which we necessitate only the right matrix-to-vector multiplications. Note that such operations do not necessitate `rank` support for the bit vectors representing the k^2-tree since a simple traversal of the tree nodes suffices to perform the matrix-vector multiplication.

The k^2-tree has recently found application to solve queries over labeled graphs and as a foundational building block for applications that process discrete bi-dimensional data. It is often advantageous to represent these data with full precision while leveraging data redundancy and dependencies for space compaction and computational efficiency [83], [84]. Recently, the k^2-tree has been utilized for implementing the Two-dimensional Block Trees [82], which allows for a compressed representation of discrete, repetitive bi-dimensional data while enabling direct access to any compressed submatrix.

In [36], the k^2-tree was utilized to address two-way Regular Path Queries (2RPQs), a widely used feature in labeled graph databases, including those compliant with the SPARQL 1.1 standard. The study demonstrated that the k^2-tree provides a self-contained, dependency-free solution that is orders of magnitude faster to construct; its query space–time performance for 2RPQs, though slower, remains competitive with highly optimized and resource-intensive alternatives.

For these reasons, we included the k^2-tree as a baseline in our experiments, utilizing the original C++ implementation provided by the University of A Coruña [31].[7] Other implementations are available in the SDSL library [85][8] and more recently in the codebase of [86].[9] We evaluated all these implementations and found that the latter two did *not* outperform the University of A Coruña's version in terms of space, time, or energy efficiency on our test datasets. Consequently, we chose to use the A Coruña implementation for our experiments. However, we modified this version by removing all `rank`-related data structures, as they are unnecessary for our multiplications, thereby slightly reducing the space usage compared to the original implementation [31].

C. REPAIR COMPRESSED MATRICES

The technique of matrix multiplications on RePair compressed matrices (mm-repair, for short) has been introduced in [30]; it compactly represents the non-zero entries of a matrix using a grammar that captures regularities between rows, reducing the representation size *and* speeding-up matrix operations.

Given a matrix M, the scheme firstly produces a representation consisting of a dictionary V of distinct values and a sequence S containing column-value pairs, as shown in Figure 2. S is obtained by scanning the matrix M row-by-row: each non-zero entry m_{ij} in M is represented in S by the pair $\langle \ell, j \rangle$, where ℓ is the index of the value m_{ij} in V. At the end of each row, a unique delimiter \$ is appended to S. Note that the above scheme works for any real-valued matrix M,

[7] https://lbd.udc.es/research/k2tree/

[8] https://github.com/simongog/sdsl-lite/blob/master/include/sdsl/k2_tree.hpp

[9] https://github.com/adriangbrandon/rpq-matrix/tree/main/lib/matrix_gn

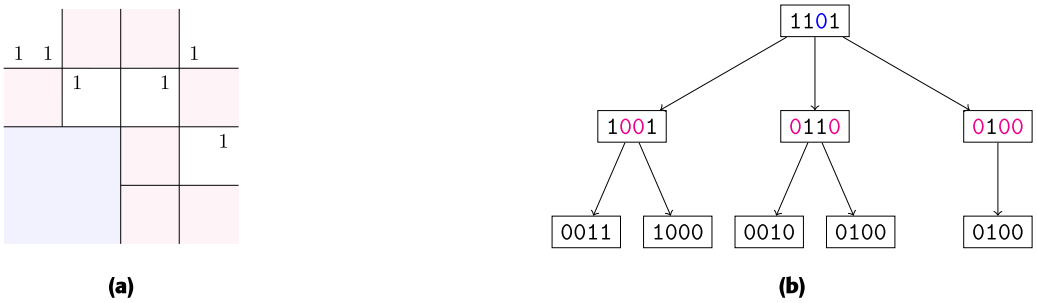


FIGURE 1. An 8 × 8 binary matrix with only the nonzero elements shown (a) and the corresponding k^2-tree (b). The different colors show the correspondence between zero submatrices and zero bits in the internal nodes of the k^2-tree.

though in this paper the input matrices are binary; thus, the dictionary V only contains the value 1.

The sequence S is then compressed using RePair [87] that computes a grammar representing S. RePair's output consists of the rules set R and a final sequence C of non-terminals interleaved with the row delimiter \$. For our example matrix M, the set R (eight rules of the form $N_i \rightarrow A_iB_i$) and the sequence C are shown in Figure 3 with a green background. The representation of M thus consists of the triplet (R, C, V). We now quickly review how this representation supports right and left matrix-to-vector multiplications without the need for decompressing the entire matrix, taking time $O(|R| + |C|)$ and using $O(|R|)$ additional space. We refer the reader to the original article [30] for the underlying theory and a thorough explanation of this scheme.

For right multiplications $y = Mx$, we perform a single scan over the rules R of the grammar from the first grammar rule up to the last one and evaluate the contribution of each rule given the values of the vector x. We perform the product $\mathsf{eval}_x(\langle \ell, j \rangle) = V[\ell] \cdot x[j]$ to evaluate a terminal symbol $\langle \ell, j \rangle$. For each nonterminal symbol N_i appearing on the left-hand side of rule $N_i \rightarrow A_iB_i$, we evaluate the sum $\mathsf{eval}_x(N_i) = \mathsf{eval}_x(A_i) + \mathsf{eval}_x(B_i)$. We store the results of the evaluated nonterminals in an auxiliary array W (dashed in Figure 3) whose size is $\Theta(|R|)$. The components of the resulting vector y are eventually determined by evaluating the nonterminals appearing in C. Figure 3 illustrates the procedure to right-multiply M by the vector $x = (2.0, 0.0, 2.0, 4.0)$, with the values W in the blue arrow corresponding to the evaluations of terminals and nonterminals of the grammar.

For left multiplications $y^T = z^T M$, mm-repair follows a dual procedure scanning the rules backward. The left-hand side of Figure 3 schematizes the left multiplication $z^T \cdot M$, with $z^T = (1.0, 0.0, 1.0, 2.0, 1.0)$. To begin with, if a non-terminal N_i appears in position j of C, we initialize $W[i] = z[j]$. Then we scan the rules bottom-up and for each rule $N_i \rightarrow A_iB_i$ we increase the entries corresponding to A_i and B_i by the value of $W[i]$. Eventually, for each non-terminal symbol $\langle \ell, j \rangle$ appearing on the right-hand side of the ith rule, we increase the zero-initialized component $y[\ell]$ of the product vector y by the quantity $V[j] \cdot W[i]$.

Experiments in [30] have demonstrated that R and C can be further compressed, resulting in more compact representations that require the on-the-fly decompression of R and C. This yields a family of compressors offering various space–time and energy trade-offs:

- Re32: R and C are not further compressed and are represented as 32-bit integers. This is the fastest variant.
- Reiv: R and C are represented as packed arrays, with entries requiring $1 + \log_2 N_{\max}$ bits ($N_{\max}$ being the largest value in R or C). In our implementation, we utilized the `int_vector` class from the SDSL-lite library [85].
- Reans: R is represented as a packed array, while C is encoded using the `ans-fold` entropy encoder from [79]. This variant achieves maximum compression.

Experimental results in [30] highlight that this family of mm-repair compressors significantly outperforms `gzip` in terms of compression while achieving compression ratios within 20% of `xz`. Unlike `gzip` and `xz`, however, mm-repair enables direct matrix-to-vector multiplications on the compressed file. In terms of matrix multiplication performance, the peak memory usage (PMU) of the multi-threaded implementation ranged from 6% to 50% of the uncompressed size for most inputs, with a PMU of $\leq$ 16% of the uncompressed size for highly compressible matrices [30].

The experiments in [30] also revealed that, compared to the well-established Compressed Linear Algebra (CLA) system [88] —considered state-of-the-art for matrix-to-vector multiplications over machine learning matrices— mm-repair achieved up to 10% better compression on 7 out of 8 datasets. The improvement in space efficiency was even more pronounced when considering runtime PMU, which was between 3.14 and 19.12 times smaller. Additionally, CLA was consistently at least twice as slow as mm-repair, despite CLA utilizing all available threads (80) whereas the authors of [30] used only 16 threads for mm-repair.

For these reasons, we have included the mm-repair method as a baseline in this paper for SpMV computations, experimenting with all three variants.

$$M = \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 4.1 & 5.3 & 0 & 5.3 \\ 4.1 & 0 & 2.0 & 4.1 \\ 0 & 5.3 & 2.0 & 5.3 \\ 4.1 & 5.3 & 2.0 & 4.1 \\ 4.1 & 5.3 & 2.0 & 5.3 \end{pmatrix} \end{array}$$

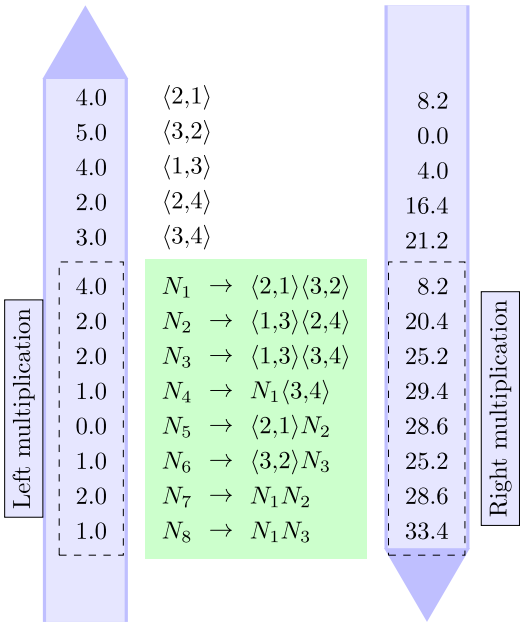
(a)

$$V = \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ (\; 2.0 \quad 4.1 \quad 5.3 \;) \end{array}$$

$$S = \langle 2,1\rangle\langle 3,2\rangle\langle 3,4\rangle\$\langle 2,1\rangle\langle 1,3\rangle\langle 2,4\rangle\$ \\ \langle 3,2\rangle\langle 1,3\rangle\langle 3,4\rangle\$\langle 2,1\rangle\langle 3,2\rangle\langle 1,3\rangle\langle 2,4\rangle\$ \\ \langle 2,1\rangle\langle 3,2\rangle\langle 1,3\rangle\langle 3,4\rangle\$$$

(b)

FIGURE 2. A matrix M (a) and the corresponding (V, S) representation (b).



$$C = N_4\$N_5\$N_6\$N_7\$N_8\$$$

FIGURE 3. The MMR-compressed M of Figure 2.

D. OTHER COMPRESSED MATRIX FORMATS

RePair-compressed matrices can be regarded as a generalization of the CLA system [88] which uses simpler compression schemes. We did not test CLA since the experiments in [30] have shown that matrix multiplication with RePair-based algorithms is faster and uses much less memory than CLA, especially for highly compressible matrices.

Another matrix compression scheme that supports matrix-vector multiplication in time proportional to the compressed graph size is the bicliques representation [89]. A biclique is a pair of vertex sets $\langle S, T\rangle$ such that every vertex in S is connected to every vertex in T. Thus a list of $|S| + |T|$ vertex id's is sufficient to encode all $|S| \cdot |T|$ edges. The edges not belonging to a biclique are compressed separately, for example employing a k^2-tree. With a proper algorithm for extracting bicliques, the resulting representation can be very space efficient for Web and social graphs, and it can be adapted to support matrix-vector multiplication [29]. We did not test this approach since only a proof-of-concept implementation is available.

Another recently introduced compressed representation of binary matrices is the Two-Dimensional Block Tree (2DBT) [82] that combines the idea of recursive subdivisions of the k^2-tree with copying redundant input portions. However, the problem of matrix-vector multiplication with this matrix representation has not been considered to date and, since it appears to be a non-trivial one, we resolved not to test 2DBT in our experiments.

VI. EXPERIMENTAL SETUP

A. TRANSPARENCY AND REPRODUCIBILITY

We made available the entire codebase to reproduce the experiments of our paper at https://github.com/acubeLab/green-lossless-spmv.

TABLE 1. Web graph datasets.

Dataset	#vertices	#edges	Web graph?
eu-2005	862 664	19 235 140	✓
hollywood-2009	1 139 905	57 515 616	✗
in-2004	1 382 908	16 917 053	✓
ljournal-2008	5 363 260	79 023 142	✗
indochina-2004	7 414 866	194 109 311	✓
uk-2002	18 520 486	298 113 762	✓
arabic-2005	22 744 080	639 999 458	✓
uk-2005	39 459 925	936 364 282	✓
it-2004	41 291 594	1 150 725 436	✓

B. DATASETS

We used different real datasets from WebGraph framework [33], [90], available from https://sparse.tamu.edu/LAW; Table 1 reports their specifications. As shown in the last column, most graphs are Web graphs, in that they derive from crawlings of one or more Web domains. The vertices in these graphs follow the lexicographical order of their reversed URL. It has been empirically observed that this helps compression since adjacent rows usually have very similar 0/1 patterns. Though the PageRank algorithm has been designed for Web graphs, we have also included two social

network graphs to measure the effectiveness of the compression algorithms in different settings. `hollywood-2009` is a graph of movie actors. Vertices are actors, and two actors are joined by an edge whenever they appear in a movie together: this is the only symmetric graph in the collection. `ljournal-2008` is a directed graph described in [91] and representing friendships in the social network LiveJournal. In this social network, the notion of friendship is asymmetric; thus the graph is directed.

C. COMPUTER ARCHITECTURE

We compiled and executed our code on a high-performance desktop server featuring a single-socket Intel® Core™ i9-7960X CPU running at 2.80 GHz, with 16 physical cores and 32 logical cores enabled by two-way Intel® Hyper-Threading. The system, equipped with 64 GB of DDR4 RAM, comprising four 16 GB Corsair memory modules (CMK64GX4M4A2400C14), operates at 2400 MHz. It runs Ubuntu 24.04.1 Long-Term Support (LTS) in 64-bit mode. The L1 data (L1d) and instruction (L1i) caches have a total capacity of 512 KiB, while the L2 caches collectively provide 16 MiB. A shared L3 cache offers 22 MiB. Given that cache associativity has become a significant factor in energy consumption [92], our L1d and L1i caches are 8-way set associative, the L2 caches are 16-way set associative, and the L3 cache is 11-way set associative.

We repeated the experiment on a resource-constrained single-board computer: a Raspberry Pi 4 model B, equipped with four ARM Cortex-A72 CPUs clocked at 1.5GHz. Each of the four cores has its own L1d and L1i cache memories (128+192 KiB); the single shared L2 cache has a size of 1 MB. L1d, L1i, and L2 cache memories are 2-, 3- and 16-way set associative. The device has 4 GB of Low Power Double Data Rate 4 Synchronous Dynamic Random-Access Memory (LPDDR4 SDRAM). The machine runs a 64-bit Ubuntu Desktop 24.04 LTS. For the experiment on this device, we used the datasets with less than 10^7 vertices (cf. Table 1), namely `eu-2005`, `hollywood-2009`, `in-2004`, `ljournal-2008`, and `indochina-2004`.

D. CODE SETUP

As a preprocessing step for our experiments, we extracted the adjacency matrix A from each dataset, transposed it, and compressed it via Zuckerli, the k^2-tree and mm-repair (cf. Section V). We also precomputed and stored on disk the array $O[1, n]$ of the out-degree for each vertex. For each compression format and input matrix, we executed 100 iterations of PageRank.

All tested codes were written in C/C++ and compiled via the flag `-O3`, which enables the highest level of optimization that the compiler can perform, including function inlining, loop unrolling, and Single Instruction Multiple Data (SIMD) instructions.

The authors of [57] demonstrated that GCC's `-O3` option provides significant energy savings, with `-O3` optimized software consuming less than 43% of the energy used by `-O0` optimized software. The potential of the `-O2` and `-O3` flags to achieve energy savings has also been recently confirmed for code compiled on Raspberry Pis [93].

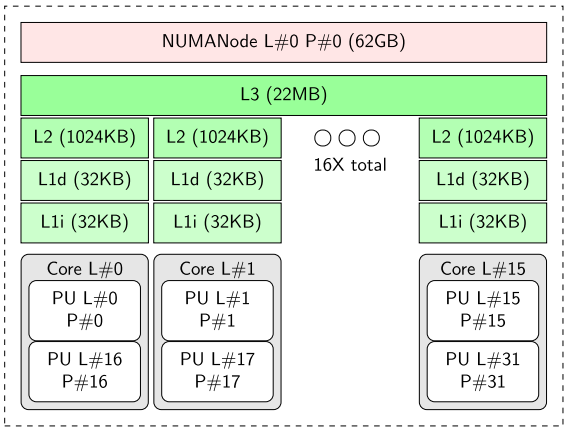


FIGURE 4. Server architecture as reported by `lstopo`.

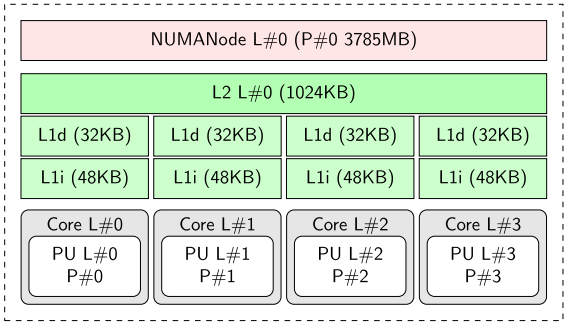


FIGURE 5. Raspberry Pi architecture as reported by `lstopo`.

Based on the methodology employed in [59], we adopt the following setup for our energy measurements:

1) We conduct experiments on machines exclusively reserved and fully dedicated to this purpose.
2) We ensure that the problem size utilized in the execution of the application does not exceed the main memory capacity, thereby preventing any swapping (paging) from occurring. For this reason, we use smaller datasets on the Raspberry Pi than in the server (see Section VI).
3) We set the CPU affinity mask for the application kernel using the `pthread_setaffinity_np` function from the POSIX threads (pthreads) library, which allows us to bind threads to specific cores. Based on the topology illustrated in Figure 4 (see in particular `P#0`, `P#1`, ... `P#31` context IDs), we incrementally assign each thread to a logical core ID, starting from zero. This method initially maximizes parallelism by distributing threads across different physical cores. Thus, after assigning one thread to each of the 16 cores, we leverage Intel®'s two-way Hyper-Threading to utilize the remaining contexts. Figure 5 illustrates the

TABLE 2. Disk occupancy.

dataset	re_32	re_iv	re_ans	k^2-tree	Zuckerli	gzip	xz
eu-2005	9.56	7.58	6.76	4.07	2.26	2.05	0.46
hollywood-2009	14.17	11.26	10.52	7.22	4.22	2.18	0.87
in-2004	11.38	8.98	7.26	2.92	1.31	1.93	0.39
ljournal-2008	26.99	21.34	19.42	14.27	9.69	2.71	1.52
indochina-2004	5.86	4.93	4.09	2.41	0.79	1.87	0.27
uk-2002	9.34	8.23	6.67	3.12	1.29	1.94	0.40
arabic-2005	6.20	5.49	4.64	2.75	0.96	1.89	0.32
uk-2005	6.79	6.12	5.03	2.72	0.96	1.90	0.34
it-2004	6.16	5.56	4.64	2.76	0.97	1.89	0.32

architecture of the Raspberry Pi. For this system, we adopt the same thread-binding approach, assigning threads to cores in a round-robin manner when running experiments with more than four threads.

4) On the server machine, fans significantly contribute to energy consumption. Given that their speed varies with temperature at all levels except full speed —resulting in corresponding changes in energy consumption—, we resolved to set the fans to full speed on the server. This configuration ensures that the fans consume a consistent amount of power over time, allowing us to include this consumption within the static power of the platform.
5) Modern processors allow for core clock scaling to balance performance and power consumption or to save power when performance is not required. This capability is known as Dynamic Voltage and Frequency Scaling (DVFS). Our server supports a `performance` DVFS mode, which statically sets the frequency to the highest available CPU frequency, and a `powersave` DVFS mode, which sets the frequency to the lowest available. To ensure reproducible performances and since our paper focuses on low power consumption, we opted for a DVFS configuration that sets the frequency statically at `powersave` for our experiments, both in the server and in the Raspberry Pi.

Additionally, we ensure that the PageRank results are consistent across different matrix formats, thereby guaranteeing the correctness of our tests. With this setup, we are confident that the dynamic energy consumption measurements obtained using multimeters and RAPL are reproducible and reflect solely the contribution of our PageRank application.

E. DATA PARALLELISM

In light of the motivations presented in Section III-D, we implemented and tested a data-parallel solution for all the tested matrix formats. We divided each matrix into as many row blocks as the number of threads. We compressed each block using the tested matrix formats and then ran a data-parallel PageRank on top of the compressed blocks. We followed this approach since right matrix-to-vector multiplications are *embarrassingly parallel*, meaning they do not require any specific synchronization data structure beyond a simple barrier to ensure each thread has completed its assigned job. For each algorithm, we forked and joined different threads in C/C++ using standard mechanisms based on POSIX Threads (Pthreads).

F. POWER MEASUREMENT

The study in [18] relied exclusively on Intel RAPL for energy measurements. However, as discussed in Section III-B, RAPL can exhibit significant inaccuracies [59], [70], [71] and its implementation varies across different architectural models. Consequently, conclusions drawn solely from RAPL readings may be unreliable. For this reason, although for the Intel® Core™ i9-7960X we also measured power consumption during PageRank computations using the RAPL interface (via the Linux profiler `perf`, version 5.15.149), we validated our RAPL measurements using a PZEM-016 power meter [94]. Within the power range relevant to our experimental setup, the power measurement uncertainty of this meter is given by $\epsilon = 0.5\% \cdot p$, where p represents the recorded power. Using the notation of Section III-C, let T denote the elapsed time and $P_D = E_D/T$ the average dynamic component of the supplied power, computed as the average of the meter's readings after subtracting the constant idle power. We then assess the accuracy $\mathcal{A}$ of our estimation for the dynamic energy E_D in (2) as

$$\mathcal{A} = \frac{P_D - \epsilon}{P_D}; \tag{5}$$

for our experiments, the average accuracy $\mathcal{A}$ was 97.65%. We sampled the instantaneous power supplied to the device at a frequency of approximately $6s^{-1}$, which we deem sufficient for accurately reconstructing the power signal. We then compute the dynamic energy consumed by each application run by approximating the integral in (2) via the trapezoidal method. For each test, we fix the number of PageRank iterations to 100, which we consider sufficient to ensure that each computation runs long enough. Sampling voltage over longer runs helps mitigate potential fluctuations in the sampled data that are beyond our control. Additionally, to further reduce variability and enhance the reproducibility of our results, we repeat each experiment four times and compute the average over these runs. The average coefficient of variation (CV) across the four energy estimations relative to the four runs is CV = 3.97%, which

we consider sufficiently low, indicating that the results of the four runs are consistent with each other and thus that four repetitions are adequate for obtaining reliable energy measurements.

As for the single-board Raspberry Pi, whose cache hierarchy and architecture are schematized in Figure 5, we pin each thread to each of the four available contexts/CPUs with increasing context IDs, proceeding in a round-robin fashion to assign multiple threads to the same core when experimenting with eight threads. We measured the current draw during PageRank computations by connecting a Fluke 8845A benchtop multimeter in series with the board's USB-C power cable. The multimeter was set to a 10 A range, with a resolution of $4\frac{1}{2}$ digits and a sampling frequency of $3s^{-1}$. Given the frequency spectrum of the current signal, this sampling rate is sufficient for an accurate reconstruction of the signal. We then compute the dynamic energy by applying the trapezoidal integration method and subtracting the idle energy.

According to the multimeter's manual [95], the measurement uncertainty ϵ for current measurement is expressed as

$$\epsilon = k_1 \cdot i + k_2 \cdot R \tag{6}$$

where $k_1 = 0.15\%$, $k_2 = 0.0080\%$, and i represents the sampled current value, and $R = 10\text{A}$ is the selected range. Since we are concerned with dynamic energy consumption, for each experiment we subtract from i the idle draw current I_{idle}, measured over a time window during which the device is inactive. Letting T denote the elapsed time, we define the average dynamic component of the current supplied to the device during the application's execution as $I_D = \frac{E_D}{TV}$, where I_D is computed as the mean of the meter's readings after subtracting the static idle current. The accuracy $\mathcal{A}$ of our dynamic energy estimation for E_D in (2) is then given by

$$\mathcal{A} = \frac{I_D - \epsilon}{I_D} \tag{7}$$

Note that, unlike in eq. (5), here we report the accuracy for the supply current, as this is the quantity measured by the Fluke multimeter. Additionally, recall that the supply voltage V remains constant, meaning the supply current is proportional to the supply power.

For our experiments, we found an average accuracy of $\mathcal{A} = 99.42\%$. To minimize measurement fluctuations, we conducted 100 iterations of PageRank, ensuring a sufficiently large sample size. Additionally, each measurement was repeated three times, and we computed the average of the resulting energy figures. The average CV among the dynamic energy estimations across the three runs was CV = 2.82%. This low CV confirms that the energy estimations are consistent across runs and robust to fluctuations beyond our control, such as those introduced by system calls from the underlying operating system.

VII. RESULTS AND DISCUSSION

A. COLLECTED METRICS

Table 2 presents the disk occupancy for the single-threaded matrix formats in bits per edge. The two rightmost columns provide a comparative analysis of the disk space used by the same matrix when compressed with `gzip` and `xz` under their default settings. More specifically, we compressed an edge-based description for each graph, consisting of sourceID-destinationID pairs listed in row-major order, with each ID occupying 64 bits in uncompressed form. One can see that, in terms of disk occupancy, `re_32` > `re_iv` > `re_ans` > k^2-tree > Zuckerli. `gzip` is sometimes better and other times worse than Zuckerli, and `xz` is the most succinct representation. However, we stress the fact that experiments on matrix-vector multiplications in [30] showed that `gzip` is at least an order of magnitude slower than algorithm `re_32`; we did not test multiplications based on on-the-fly decompression for `xz`-compressed matrices since `xz` is even slower in decompression than `gzip`. For these reasons, we did not test Pagerank on `gzip`- or `xz`-compressed matrices.

In all our experiments we measured the peak memory usage (PMU) for PageRank executions using the command-line `time` utility and we report it using again the number of bits per edge, as in Table 2. The PMU will be consistently larger than the disk usage reported in Table 2 for two factors: the first one is that the execution of the algorithm requires additional temporary data structures, and the second one is that splitting the matrix into blocks, to support multi-threaded multiplication, usually reduces the overall compression.

B. EXPERIMENTS ON THE INTEL® CORE™ I9-7960X SERVER

The log-log scale plot in Figure 6 illustrates the total elapsed time for 100 iterations of PageRank as a function of peak memory usage (PMU) measured in bits per edge. We employ distinct markers to differentiate among the algorithms. Each subfigure corresponds to a different dataset and presents seven data points for each algorithm, representing executions with 1, 2, 4, 8, 16, 24, and 32 threads.

We observe that Zuckerli is generally the most compressed but the slowest solution, while `re_32` is the fastest, albeit more space-consuming. When using a single thread, the k^2-tree is often more than 3× faster than Zuckerli; for the datasets `eu-2005`, `in-2004`, and `uk-2005`, the k^2-tree outperforms Zuckerli across all degrees of parallelism, resulting in approximately 3× more lightweight in space. For the remaining six datasets, the single-threaded k^2-tree requires 2× more disk space than Zuckerli which, however, degrades significantly faster with increasing parallelism. We believe this is due to Zuckerli's more complex compression strategy, which involves a sophisticated method for selecting the reference list to represent each adjacency list. Indeed, as the degree of parallelism increases, the dimension of the adjacency-matrix row block assigned to each single

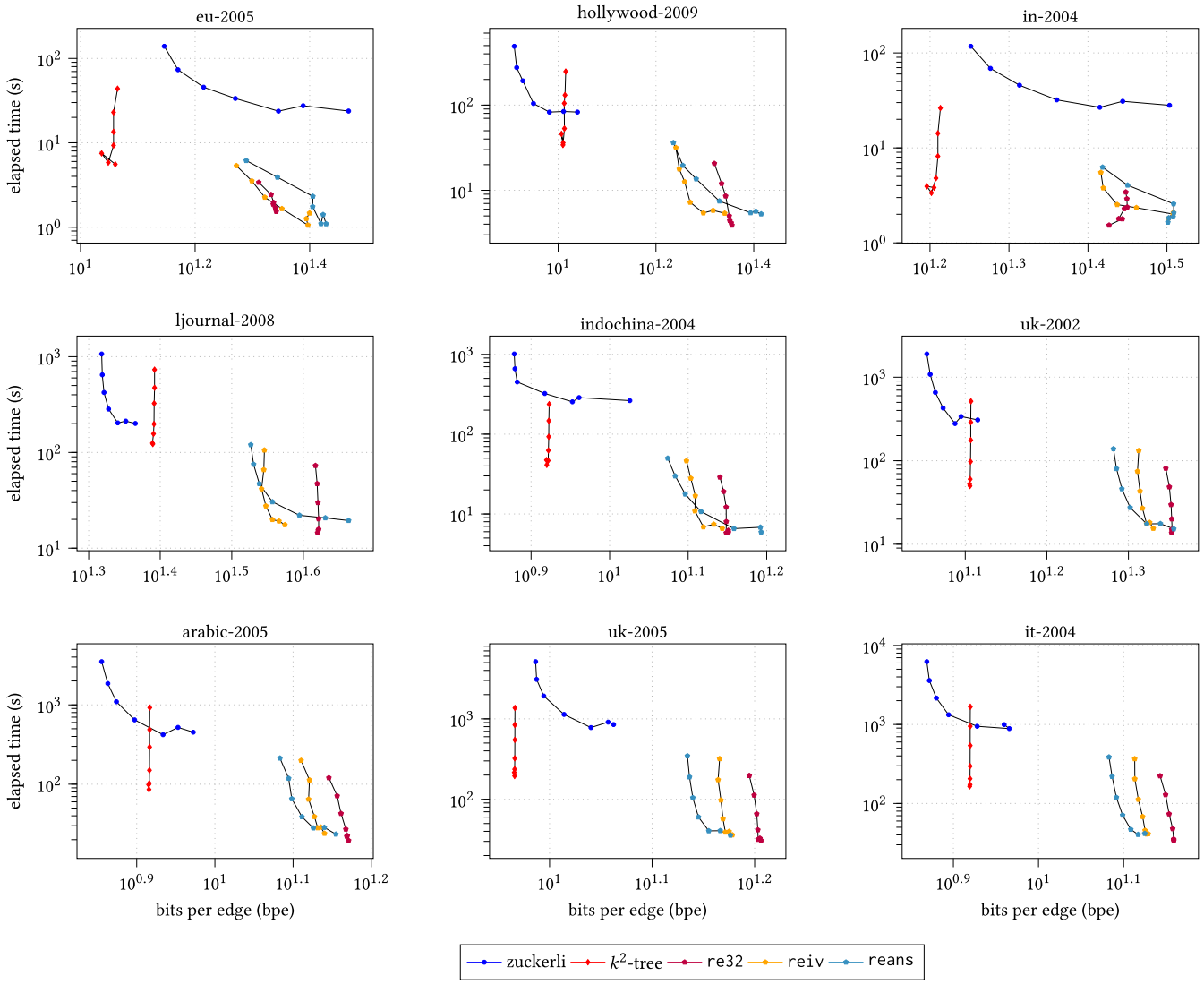


FIGURE 6. Bits per edge (x-axis) and elapsed times (y-axis) for 100 iterations of PageRank on the Intel® Core™ i9-7960X for 1, 2, 4, 8, 16, 24, and 32 threads.

thread decreases, leading to a reduction in the number of candidate reference lists for each adjacency list.

The single-threaded grammar-based solutions `re_32`, `re_iv`, and `re_ans` provide faster alternatives, but they are approximately 1.5× more space-consuming than Zuckerli and the k^2-tree. Recall that the grammar-based compressors can in principle handle any input matrix, that is, they are not specialized to binary matrices such as Zuckerli and the k^2-tree. In addition, the grammar-based algorithms also support efficient left matrix-to-vector multiplication, which though not needed in our PageRank implementation, is essential for instance in conjugate gradient methods [30]. Among the grammar-based algorithms, `re_ans` is often the most space-efficient. However, as the number of threads increases, its space requirements increase significantly compared to `re_32` and `re_iv`, which are more robust and less sensitive to increases in thread count.

In our green computing scenario, we focus specifically on the energy consumption of various algorithms. Figure 7 compares the performance of PageRank computations in terms of time (dashed lines) and energy (solid lines) on a semilogarithmic scale, as a function of the number of threads. Intel's RAPL, represented by the dotted line, generally overestimates energy consumption, reporting values 14% to 18% higher than those measured by the power meter (solid lines) for dynamic energy at lower levels of parallelism. Conversely, at higher thread counts, RAPL slightly underestimates energy consumption by approximately 1% to 5%. The only exception occurs with the `in-2004` dataset using grammar-based algorithms, where RAPL continues to overestimate energy consumption by up to 40%, even at higher degrees of parallelism. In many cases, however, the dotted line representing RAPL aligns with the solid line of the power meter. As anticipated, performance improves

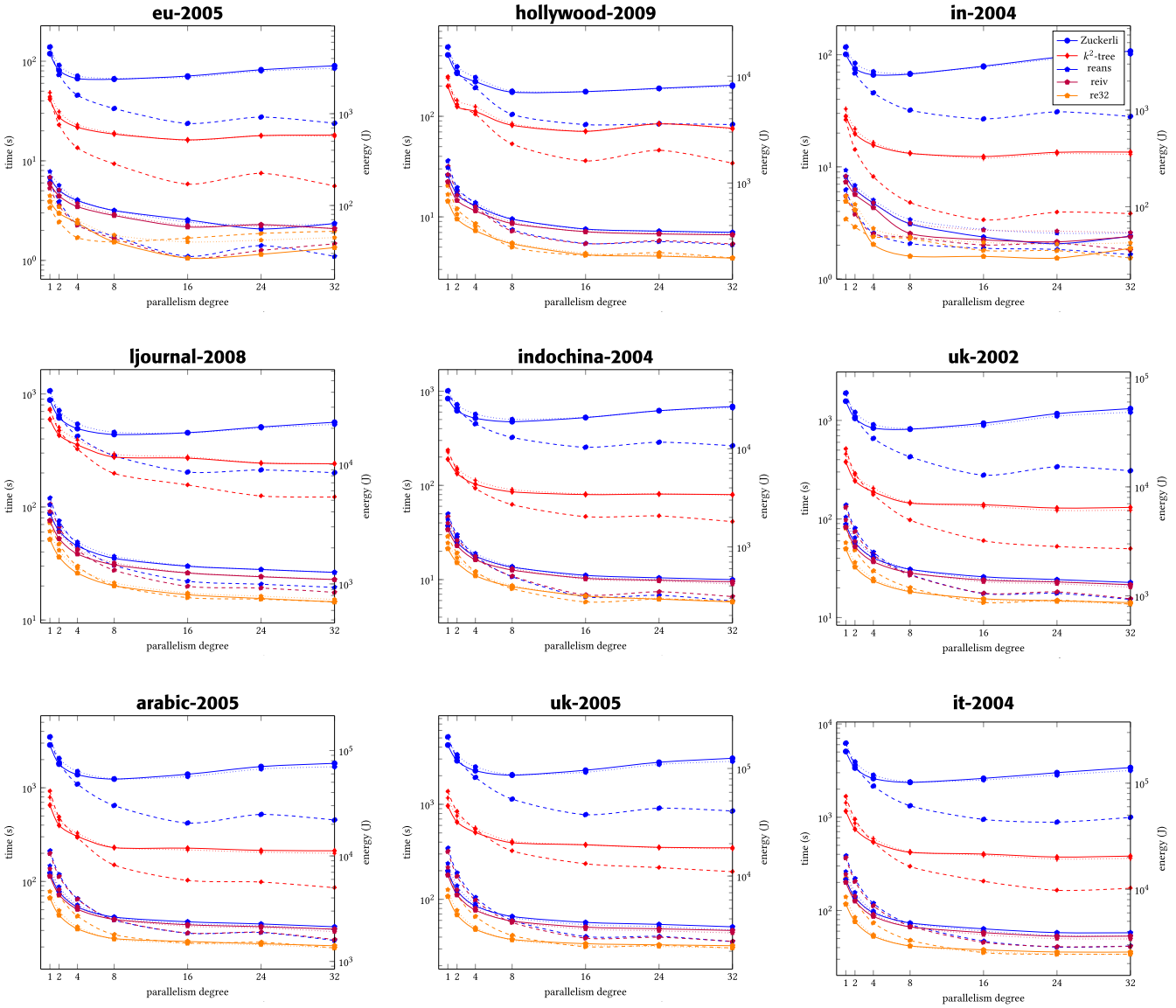


FIGURE 7. The time performance, measured in seconds (dashed line; left y-axis), and the energy estimation obtained from the power meter, expressed in Joules (solid line; right y-axis), on the Intel® Core™ i9-7960X are presented as a function of the number of threads (x-axis). The energy estimation from Intel RAPL is depicted with a dotted line, which often closely aligns with the solid line representing the meter's estimation.

with increasing degrees of parallelism until the optimal level is reached. Beyond this point, no further speedup occurs, and the algorithms may experience slower execution times. In examining the energy performances reported in the same graph, it becomes clear that energy does not scale as well as time. Energy savings are generally more limited than time savings, and the optimal parallelism degree for energy is usually lower than that for time. This trend is consistently evident, particularly for the slower algorithms, namely Zuckerli and the k^2-tree. For instance, for `hollywood-2009`, time performance keeps slightly improving up to 32 threads, while energy consumption does not decrease beyond 8 or 16 threads. In the cases of `in-2004` and Zuckerli, we find an optimal parallelism degree for the time performances for 16 threads, while energy consumption does not decline past 4 threads.

This observation challenges the traditional assumption that energy optimization strategies are inherently equivalent to time optimization strategies. Instead, our experiments align with recent work (such as [39]) and suggest that developers of multi-threaded applications should adopt a multicriteria approach, considering both time and energy performance. For instance, they could select the fastest solution that satisfies specific energy consumption constraints, or vice versa, select the smallest energy consumption solution that satisfies specific time constraints.

To gain a clearer understanding of the performance of various algorithms, Figure 8 illustrates the cycles-per-

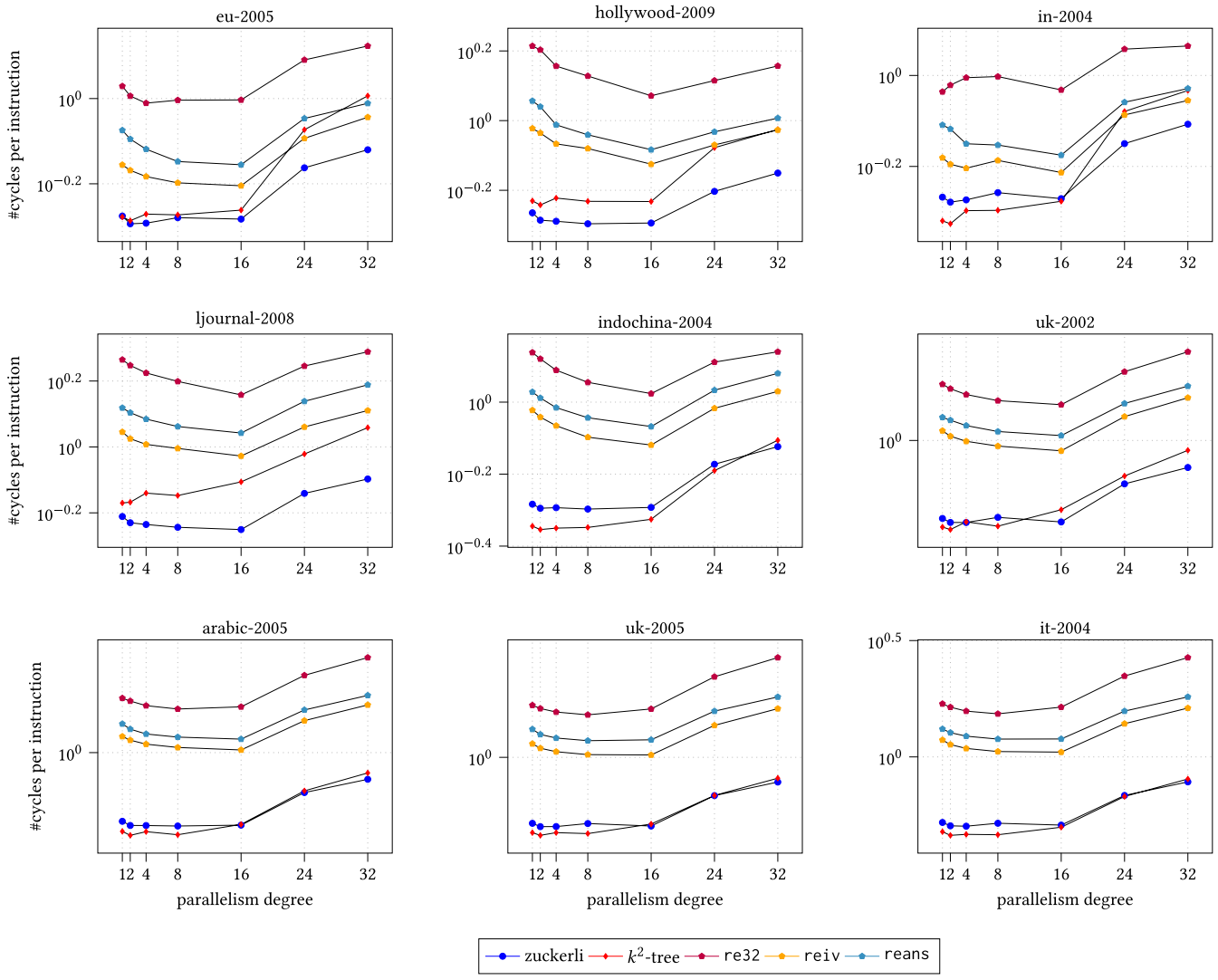


FIGURE 8. Number of cycles per instruction (y-axis) as a function of the parallelism degree (x-axis) for 100 iterations of PageRank on the Intel® Core™ i9-7960X.

instruction throughput, as reported by the `perf` profiler, as a function of the parallelism degree. A lower number of cycles per instruction indicates higher instruction throughput and, consequently, enhanced energy efficiency per unit of time for the tested algorithms. Among these, Zuckerli and the k^2-tree exhibit the fewest cycles per instruction. Yet, it is important to note that these two formats have higher overall energy requirements; indeed, their compact nature necessitates more instructions, resulting in longer completion times and increased energy consumption. We believe that the lower cycles per instruction observed are due to the greater memory requirements of grammar-based solutions, which lead to more cache accesses, potentially causing pipeline stalls and increased latencies at the hardware level.

Interestingly, the throughputs of Zuckerli and, in particular, the k^2-tree tend to decline as the number of threads increases. In contrast, for the grammar-based approaches, the number of cycles required for each instruction initially often follows a monotonically decreasing trend up to a parallelism degree of 8 or 16, followed by an increase.

Figure 9 illustrates the number of accesses to L1 data caches (L1D) for each matrix format as a function of the degree of parallelism. The total height of each bar represents the overall number of operations in the L1D cache, calculated by summing the number of load cache misses (red), load cache hits (green), and store operations (blue). The x-axis denotes the degree of parallelism, with each group of five bars corresponding to the five tested compression algorithms, presented in the following left-to-right order: Zuckerli, k^2-tree, `re_ans`, `re_iv`, and `re_32`.

The most compact format, Zuckerli, requires fewer cache loads than all other algorithms, resulting in an almost negligible number of cache misses. However, its cache operations increase with the degree of parallelism. In contrast,

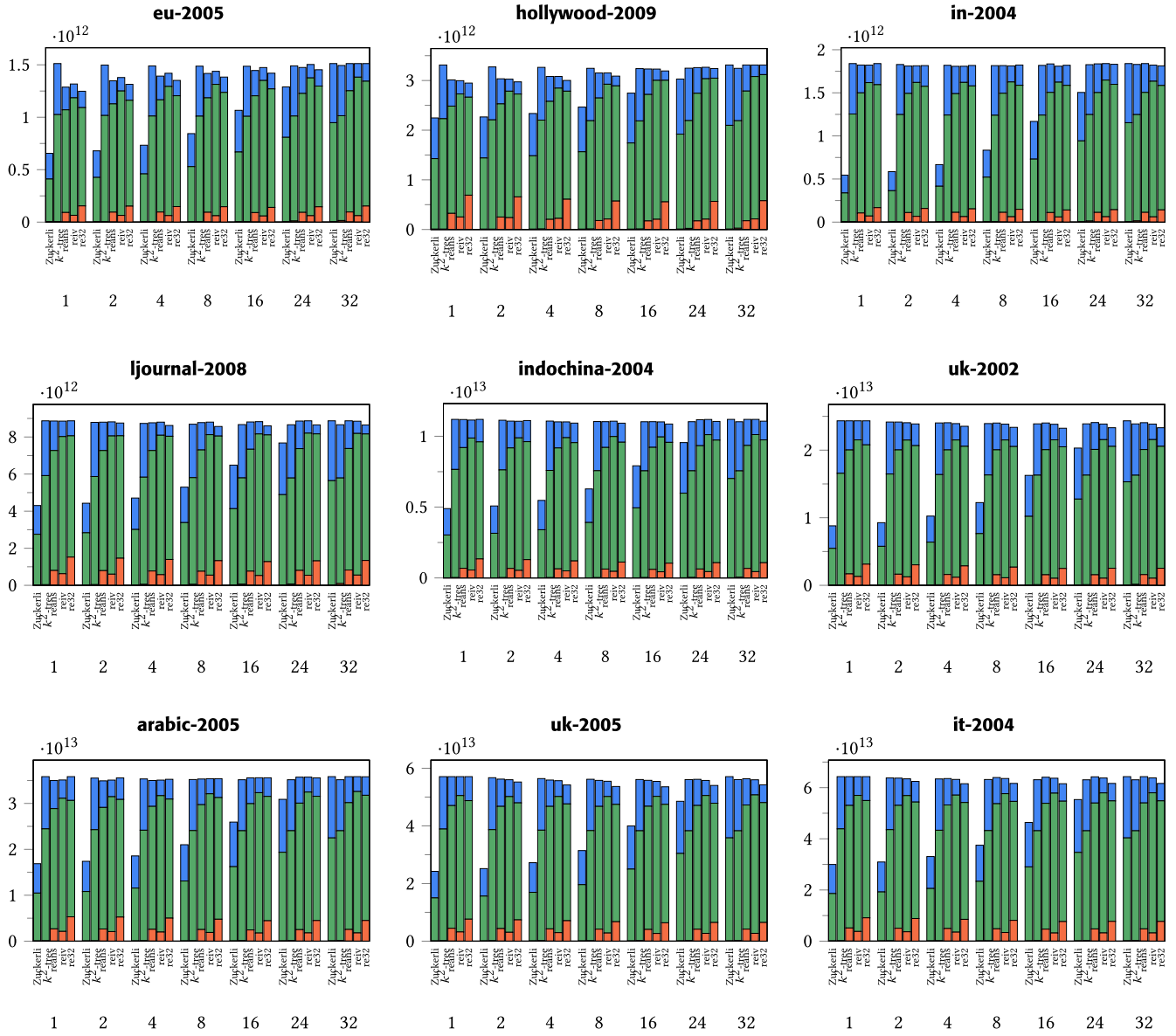


FIGURE 9. Accesses to L1d cache memories on the Intel® Core™ i9-7960X (y-axis) as a function of the parallelism degree (x-axis). The colored bars indicate the number of load cache misses (red), load cache hits (green), and store operations (blue). Each group of five bars corresponds to the five tested compression algorithms, presented in the following left-to-right order: Zuckerli, k^2-tree, `re_ans`, `re_iv`, and `re_32`.

the remaining algorithms exhibit a nearly constant amount of cache operations regardless of the thread count. Notably, the k^2-tree, the second most compressed format, also incurs a negligible number of cache misses. This suggests that both Zuckerli and the k^2-tree exhibit higher data locality, as they need to load less data.

Similarly, Figure 10 shows the number of accesses to the L3 caches. For most datasets and formats, we observe in this case a reduction in the number of accesses as the degree of parallelism increases. As noted in [18], the efficiency of cache accesses can help explain instruction throughput and thus energy consumption.

For instance, in the PageRank computations on the k^2-tree for the `arabic-2005` dataset, the number of L3 cache operations increases up to 4 threads, then decreases at 8 threads, and subsequently rises again, mirroring the behavior observed in Figure 8. Similarly, for the same dataset, Zuckerli exhibits an increase in L3 accesses starting from 16 threads, once again aligning with the trends in Figure 8. This correlation between the L3 operations and instruction throughput is evident across other algorithms as well, and for higher degrees of parallelism in particular.

C. EXPERIMENTS ON THE RASPBERRY PI

To validate our results, we repeated the PageRank computations on an ARM-based Raspberry Pi (see the specifics in Section VI-C). Due to the limitations of this architecture, we only considered the five smallest graphs (following item 2

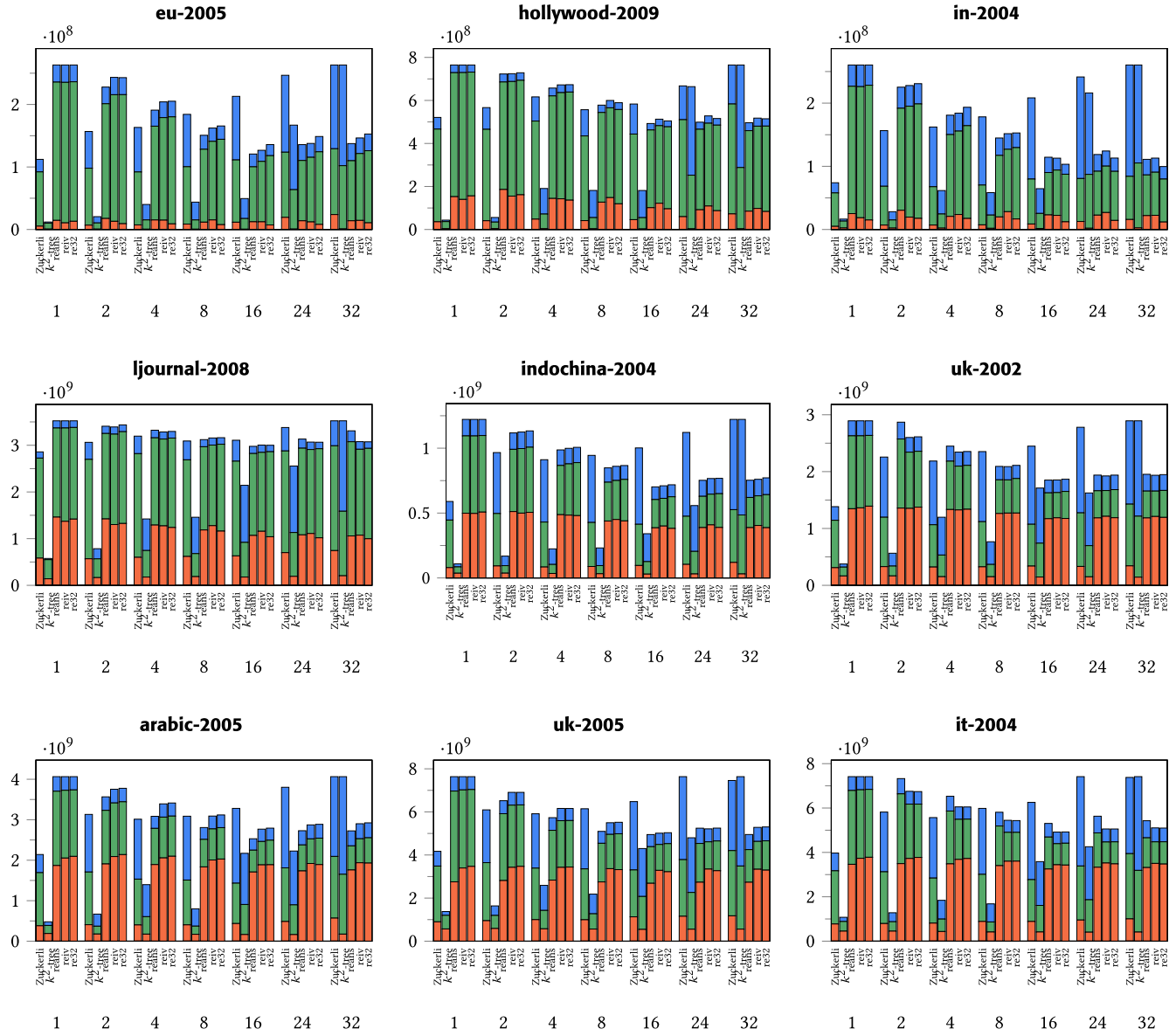


FIGURE 10. Number of accesses to L3 cache memories on the Intel® Core™ i9-7960X (y-axis) as a function of the parallelism degree (x-axis). The colored bars indicate the number of load cache misses (red), load cache hits (green), and store operations (blue). Each group of five bars corresponds to the five tested compression algorithms, presented in the following left-to-right order: Zuckerli, k^2-tree, `re_ans`, `re_iv`, and `re_32`.

of Section VI-D) and ran the algorithms with only 1, 2, 4, and 8 threads.

Figure 11 illustrates the space–time performances on the Raspberry Pi. We observe that time and space are not always monotonic when scaling from 4 to 8 threads, as PMU and completion time increase, notably for Zuckerli. In three out of five datasets, Zuckerli is the most compact, whereas, in the remaining two datasets, the k^2-tree demonstrates superior compactness, consistent with the results in Figure 6 for our server machine. Regarding time performance, the ranking positions the grammar-based solutions `re_32`, `re_iv` and `re_ans` as the fastest (in this order), followed by the k^2-tree, with Zuckerli trailing behind. The grammar-based solutions achieve speeds at least 4× faster than the k^2-tree, albeit at the cost of significantly increased memory usage.

In comparing Figure 6 with Figure 11, we observe that on the Raspberry Pi, even the single-threaded version of the k^2-tree has a faster completion time than all Zuckerli parameterizations across all datasets. In contrast, in the server experiments, the multithreaded version of Zuckerli outperformed the single-threaded k^2-tree on 7 out of 9 datasets (all except `in-2004` and `indochina-2004`).

Figure 12 shows time and energy usage as a function of the number of threads; we used at most 8 threads since the Raspberry has only 4 cores. As we mentioned in Section VI-F, the energy consumption was estimated

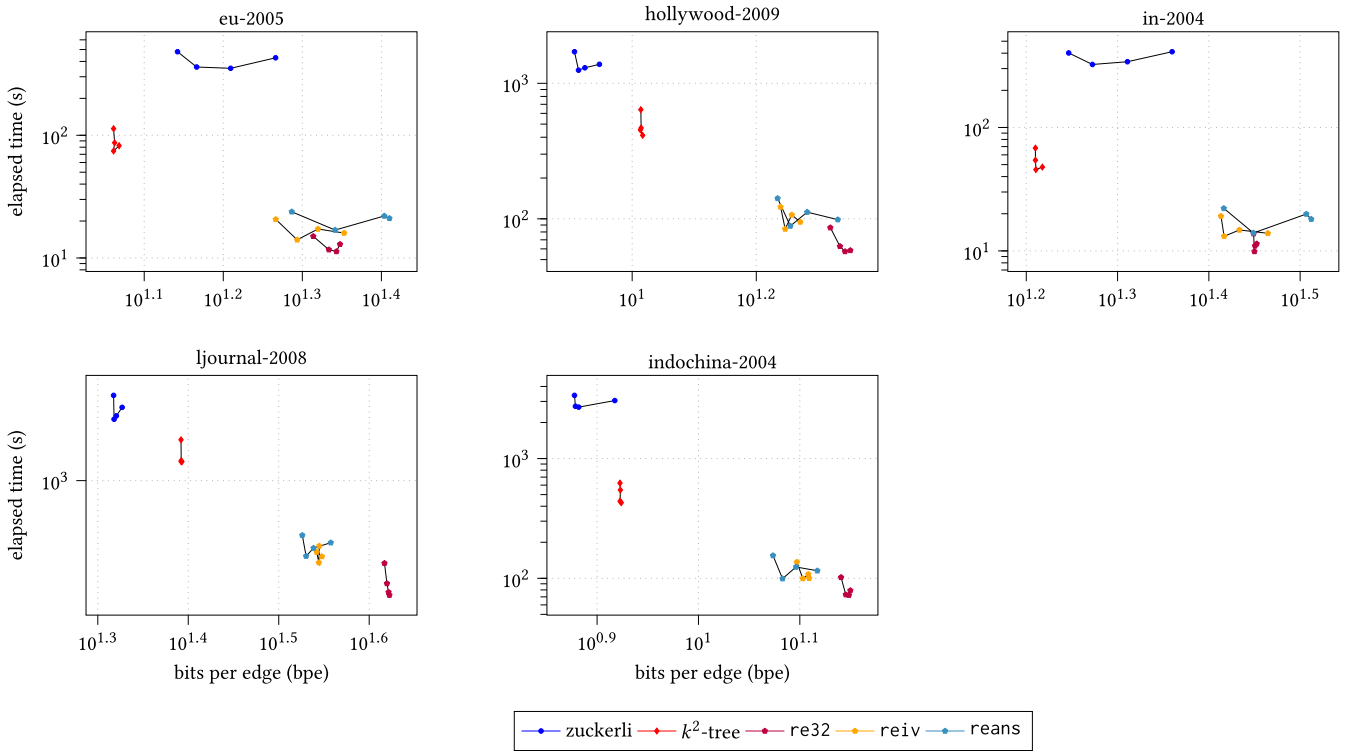


FIGURE 11. Bits per edge (x-axis) and elapsed times (y-axis) for 100 iterations of PageRank on the Raspberry Pi, for 1, 2, 4, and 8 threads.

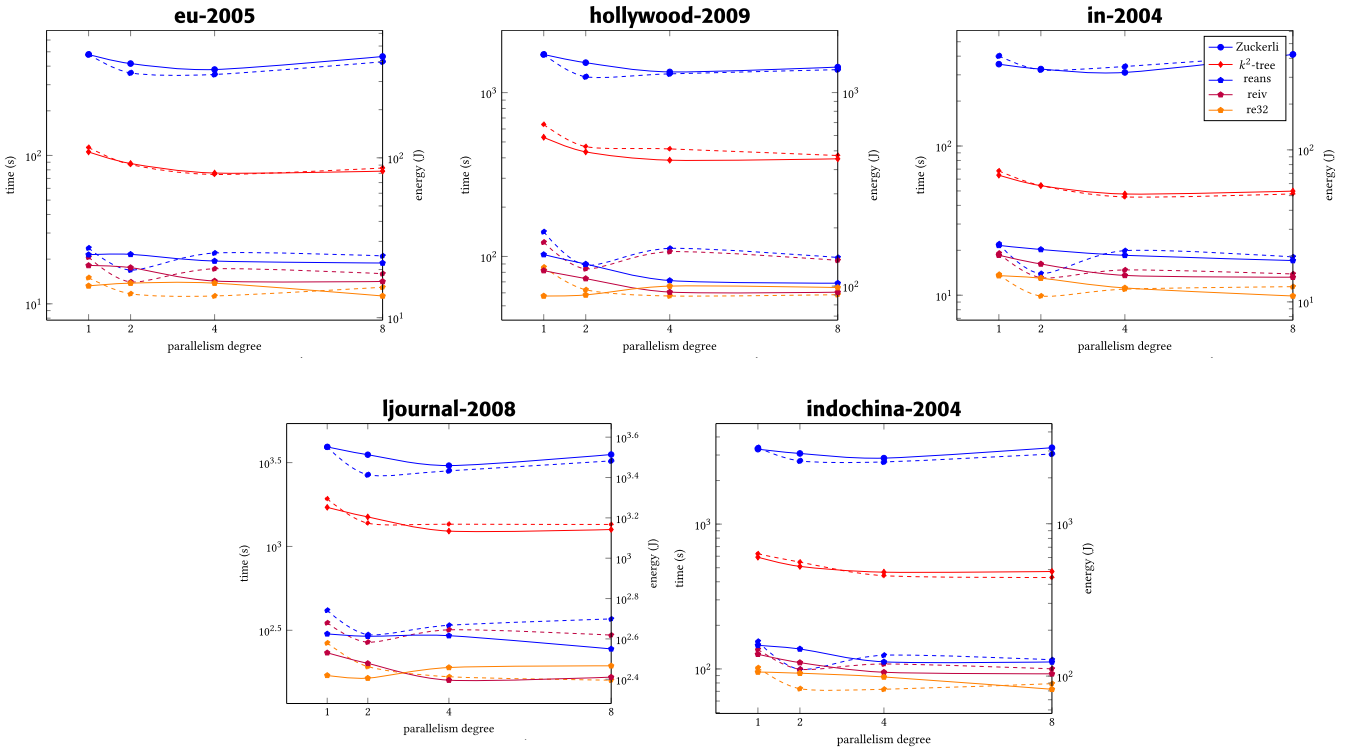


FIGURE 12. Time performance in seconds (dashed line; left y-axis) and energy requirements in Joules (solid line; right y-axis) on the Raspberry Pi 4 as a function of the number of threads (x-axis).

by measuring the input current during the computation. The results align well with those for the server. Since the time-energy behavior in Figure 12 substantially confirms those of Figure 7, we omit reporting the metrics we captured

for L1d caches for the Raspberry Pi (the Raspberry Pi does not have any L3 caches).

Even for the Raspberry Pi, we observe a convex trend, particularly with Zuckerli. In this case, allocating additional threads (8 threads, exceeding the 4 available cores) prolongs completion time and increases resource utilization, resulting in greater energy inefficiency. In contrast, the data for Zuckerli on the Intel® Core™ i9-7960X server in Figure 7 exhibit a less pronounced convex trend when scaling to high levels of parallelism. We believe that, in the server environment, when approaching the optimal parallelism degree, the increased energy required to operate with more threads offsets the benefit of slightly faster execution gained from the time speedup. Our results suggest that the qualitative behavior observed on the Intel® Core™ i9-7960X server can be extended to resource-constrained edge devices.

VIII. CONCLUSION AND FUTURE WORK

We evaluated the completion time, disk usage, memory consumption, and energy efficiency of three algorithms for binary sparse matrix-vector multiplications (SpMVs) in computing the PageRank of large web graphs and social networks. The key insights from our study and our answers to the research questions Q1–Q3 mentioned in Section II are summarized below:

- In response to question Q1, we have demonstrated that appropriately compressed representations enable efficient processing of large datasets, even on resource-constrained devices. Different algorithms exhibit distinct space-time-energy trade-offs, which are significantly influenced by the quantity of available processing threads.
- For PageRank computations, the k^2-tree emerges as a well-balanced option: it is nearly as fast as grammar-based compressors while being almost as space-efficient as Zuckerli. Additionally, its energy consumption falls between that of Zuckerli and the grammar-based compressors, and it exhibits minimal degradation in compression ratios with increasing thread counts.
- The selection of a compressed representation should be guided by the nature of the input data. While all tested methods performed less effectively on social network graphs, results from `gzip` and `xz` indicate significant compressibility, suggesting that other compressed formats, as discussed in Section V-D, warrant consideration.
- In response to question Q2, we observed that energy consumption does not always correlate directly with runtime, making independent measurement essential in critical scenarios. This distinction becomes particularly evident when adjusting the number of threads: our experiments confirm that the thread configuration that minimizes energy consumption may differ from the one that optimizes completion time. This finding suggests that further multi-objective optimization research on compressed matrix and vector formats could provide valuable insights into the interplay between time and energy in other scientific applications.
- Careful selection of compressed representations can reduce energy usage by one to two orders of magnitude. This advantage is evident across server-grade hardware and the resource-constrained Raspberry Pi board.
- Our analysis of Q3, illustrated in Figure 9 and particularly in Figure 10, highlights the impact of cache operations on cycles per instruction (Figure 8). Inefficient utilization of the cache hierarchy degrades instruction throughput, leading to higher latencies and energy inefficiencies.

To translate these findings into actionable insights for software engineers, we propose the following guidelines:

1) Adopt computationally-friendly and domain-specific compressed formats for the efficient handling of large datasets on resource-constrained devices —such as single-board computers.
2) Prioritize cache efficiency, implement cache-oblivious solutions [96] and [80, §2] where possible. Poor cache management, particularly of L1 and L3 caches, can severely degrade performance and increase energy consumption. Streamlining data access patterns can enhance cycles-per-instruction rates and reduce inefficiencies.
3) Evaluate space–time trade-offs by accounting for thread count. Optimizing thread allocation based on specific hardware can balance performance and energy consumption effectively.
4) Use k^2-trees for PageRank and other applications dominated by matrix-vector multiplications, as k^2-trees represent a simple, yet robust and energy-efficient choice.
5) Investigate time-energy relationships: As runtime and energy consumption do not exhibit direct proportionality, explore the interplay among thread counts, runtime, and energy usage in energy-sensitive scenarios. We advocate for further research into developing robust, multi-criteria algorithmic frameworks that address performance and sustainability objectives.

Looking ahead, our future work will focus on exploring additional lossless compression formats for matrices and vectors, as well as extending our methodology to applications beyond PageRank. We believe that deeper investigation into energy-time optimization may yield further theoretical and practical insights, helping software engineers reduce carbon footprints and enhance battery life.

We also aim to examine cross-platform energy-efficient implementations of key compressed data structures, including the FM-index, rank and select support structures, suffix arrays, and succinct tree representations. This research

will underscore the broader potential of data compression in reducing carbon footprints across diverse computing platforms.

Finally, as for machine learning optimization, we propose integrating lossless compression techniques with state-of-the-art lossy strategies. This hybrid approach seeks to balance storage efficiency with accuracy, unlocking new possibilities for data compression and its impact on computational sustainability.

ACKNOWLEDGMENT

The authors would like to thank Marco Danelutto, Gabriele Mencagli, Maurizio Davini, Fabio Pratelli, and the staff of the Green Data Center (GDC), University of Pisa, for their insightful comments, valuable suggestions, and engaging discussions on this work. They also would like to thank the anonymous reviewers whose thorough reviews helped them enhance this article.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[3] T. B. Brown et al., "Language models are few-shot learners," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates Inc., Jan. 2020, pp. 1877–1901.

[4] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in *Proc. 15th Conf. Eur. Chapter Assoc. Comput. Linguistics, Short Papers*, vol. 2, 2017, pp. 427–431.

[5] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North American Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol., (Long Short Papers)*, vol. 1, Jan. 2018, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[6] G. Navarro, "Indexing highly repetitive string collections—Part I: Repetitiveness measures," *ACM Comput. Surv.*, vol. 54, no. 2, pp. 1–31, Mar. 2021.

[7] G. Navarro, "Indexing highly repetitive string collections—Part II: Compressed indexes," *ACM Comput. Surv.*, vol. 54, no. 2, pp. 1–32, Mar. 2022.

[8] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Comput. Surv.*, vol. 40, no. 1, pp. 1–39, Feb. 2008, doi: 10.1145/1322432.1322433.

[9] W. Liang, P. D. Meo, Y. Tang, and J. Zhu, "A survey of multi-modal knowledge graphs: Technologies and trends," *ACM Comput. Surv.*, vol. 56, no. 11, pp. 1–41, Nov. 2024, doi: 10.1145/3656579.

[10] (2024). *Graph Database Market Size Projections*. Accessed: Sep. 6, 2024. [Online]. Available: https://finance.yahoo.com/news/graph-database-market-size-projected-142500719.html

[11] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu, "A survey on knowledge graphs: Representation, acquisition, and applications," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 2, pp. 494–514, Feb. 2022, doi: 10.1109/TNNLS.2021.3070843.

[12] C. Peng, F. Xia, M. Naseriparsa, and F. Osborne, "Knowledge graphs: Opportunities and challenges," *Artif. Intell. Rev.*, vol. 56, no. 11, pp. 13071–13102, Nov. 2023, doi: 10.1007/s10462-023-10465-9.

[13] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, D. Metropolitansky, R. Osazuwa Ness, and J. Larson, "From local to global: A graph RAG approach to query-focused summarization," 2024, *arXiv:2404.16130*.

[14] J. Kunecová, A. Bikfalvi, and P. Marques, "Sustainability orientation, industrial big data and product innovation: Evidence from the European manufacturing sector," *Comput. Ind. Eng.*, vol. 191, May 2024, Art. no. 110163.

[15] C. Freitag, M. Berners-Lee, K. Widdicks, B. Knowles, G. S. Blair, and A. Friday, "The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations," *Patterns*, vol. 3, no. 8, Aug. 2022, Art. no. 100576.

[16] R. Verdecchia, J. Sallou, and L. Cruz, "A systematic review of green AI," *WIREs Data Mining Knowl. Discovery*, vol. 13, no. 4, Jul. 2023, Art. no. e1507.

[17] E. Baldini, S. Chessa, and A. Brogi, "Estimating the environmental impact of green IoT deployments," *Sensors*, vol. 23, no. 3, p. 1537, Jan. 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/3/1537

[18] J. Fuentes-Sepúlveda and S. Ladra, "Energy consumption in compact integer vectors: A study case," *IEEE Access*, vol. 7, pp. 155625–155636, 2019.

[19] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, "What do programmers know about software energy consumption?" *IEEE Softw.*, vol. 33, no. 3, pp. 83–89, May 2016.

[20] G. Navarro, *Compact Data Structures: A Practical Approach*. Cambridge, U.K.: Cambridge Univ. Press, 2016.

[21] H. Wang, S. Ma, L. Dong, S. Huang, H. Wang, L. Ma, F. Yang, R. Wang, Y. Wu, and F. Wei, "BitNet: Scaling 1-bit transformers for large language models," 2023, *arXiv:2310.11453*.

[22] S. Ma, H. Wang, L. Ma, L. Wang, W. Wang, S. Huang, L. Dong, R. Wang, J. Xue, and F. Wei, "The era of 1-bit LLMs: All large language models are in 1.58 bits," 2024, *arXiv:2402.17764*.

[23] J. Woo, K. Jung, and S. Mukhopadhyay, "Efficient hardware design of DNN for RF signal modulation recognition employing ternary weights," *IEEE Access*, vol. 12, pp. 80165–80175, 2024.

[24] S. Zhu, L. H. K. Duong, H. Chen, D. Liu, and W. Liu, "FAT: An in-memory accelerator with fast addition for ternary weight neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 3, pp. 781–794, Mar. 2023.

[25] G. Menghani, "Efficient deep learning: A survey on making deep learning models smaller, faster, and better," *ACM Comput. Surv.*, vol. 55, no. 12, pp. 1–37, Mar. 2023.

[26] Y. Liu, K. Chen, and L. Zhu, "Efficient federated learning algorithm using sparse ternary compression based on layer variation classification," *Comput. Netw.*, vol. 247, Jun. 2024, Art. no. 110471.

[27] Z. Xu, L. Li, and W. Zou, "Exploring federated learning on battery-powered devices," in *Proc. ACM Turing Celebration Conf. China*. New York, NY, USA: Association for Computing Machinery, May 2019, pp. 1–6.

[28] J. N. F. Alves, S. Moustafa, S. Benkner, A. P. Francisco, W. N. Gansterer, and L. M. S. Russo, "Accelerating graph neural networks with a novel matrix compression format," 2024, *arXiv:2409.02208*.

[29] A. P. Francisco, T. Gagie, D. Köppl, S. Ladra, and G. Navarro, "Graph compression for adjacency-matrix multiplication," *Social Netw. Comput. Sci.*, vol. 3, no. 3, p. 193, Mar. 2022.

[30] P. Ferragina, G. Manzini, T. Gagie, D. Köppl, G. Navarro, M. Striani, and F. Tosoni, "Improving matrix-vector multiplication via lossless grammar-compressed matrices," *Proc. VLDB Endowment*, vol. 15, no. 10, pp. 2175–2187, Jun. 2022. [Online]. Available: https://www.vldb.org/pvldb/vol15/p2175-tosoni.pdf

[31] N. R. Brisaboa, S. Ladra, and G. Navarro, "Compact representation of web graphs with extended functionality," *Inf. Syst.*, vol. 39, pp. 152–174, Jan. 2014.

[32] L. Versari, I.-M. Comsa, A. Conte, and R. Grossi, "Zuckerli: A new compressed representation for graphs," *IEEE Access*, vol. 8, pp. 219233–219243, 2020.

[33] P. Boldi and S. Vigna, "The webgraph framework I: Compression techniques," in *Proc. 13th Int. Conf. World Wide Web* New York, NY, USA: Association for Computing Machinery, May 2004, pp. 595–602, doi: 10.1145/988672.988752.

[34] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the web," in *Proc. Web Conf.*, 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:1508503

[35] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, "RAPL in action: Experiences in using RAPL for power measurements," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 3, no. 2, pp. 1–26, Mar. 2018, doi: 10.1145/3177754.

[36] D. Arroyuelo, A. Gómez-Brandón, and G. Navarro, "Evaluating regular path queries on compressed adjacency matrices," *VLDB J.*, vol. 34, no. 1, p. 2, Nov. 2024, doi: 10.1007/s00778-024-00885-6.

[37] N. R. Brisaboa, G. De Bernardo, G. Gutierrez, S. Ladra, M. R. Penabad, and B. A. Troncoso, "Efficient set operations over k2-trees," in *Proc. Data Compress. Conf.*, Snowbird, UT, USA, Apr. 2015, pp. 373–382.

[38] D. Belazzougui, M. Cáceres, T. Gagie, P. Gawrychowski, J. Kärkkäinen, G. Navarro, A. Ordóñez, S. J. Puglisi, and Y. Tabei, "Block trees," *J. Comput. Syst. Sci.*, vol. 117, pp. 1–22, May 2021, doi: 10.1016/j.jcss.2020.11.002.

[39] S. Khokhriakov, R. R. Manumachu, and A. Lastovetsky, "Multicore processor computing is not energy proportional: An opportunity for bi-objective optimization for energy and performance," *Appl. Energy*, vol. 268, Jun. 2020, Art. no. 114957.

[40] R. R. Manumachu and A. Lastovetsky, "On energy nonproportionality of CPUs and GPUs," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2022, pp. 34–44.

[41] A. Shahid, M. Fahad, R. R. Manumachu, and A. Lastovetsky, "Improving the accuracy of energy predictive models for multicore CPUs by combining utilization and performance events model variables," *J. Parallel Distrib. Comput.*, vol. 151, pp. 38–51, May 2021.

[42] S. Roy, A. Rudra, and A. Verma, "Energy aware algorithmic engineering," in *Proc. IEEE 22nd Int. Symp. Modeling, Anal. Simulation Comput. Telecommun. Syst.*, Sep. 2014, pp. 321–330.

[43] T. Lu, J. Xue, P. Shen, H. Liu, X. Gao, X. Li, J. Hao, D. Huang, R. Zhao, J. Yan, M. Yang, B. Yan, P. Gao, Z. Lin, Y. Yang, and T.-L. Ren, "Two-dimensional fully ferroelectric-gated hybrid computing-in-memory hardware for high-precision and energy-efficient dynamic tracking," *Sci. Adv.*, vol. 10, no. 36, Sep. 2024, Art. no. eadp0174, doi: 10.1126/sciadv.adp0174.

[44] F. Calero, C. A. Cañizares, and K. Bhattacharya, "Dynamic modeling of battery energy storage and applications in transmission systems," *IEEE Trans. Smart Grid*, vol. 12, no. 1, pp. 589–598, Jan. 2021.

[45] J. Koetsier. (Feb. 2023). *ChatGPT Burns Millions Every Day. Can Computer Scientists Make AI One Million Times More Efficient*. [Online]. Available: https://www.forbes.com/sites/johnkoetsier/2023/02/10/chatgpt-burns-millions-every-day-can-computer-scientists-make-ai-one-million-times-more-efficient/?sh=67c3d8256944

[46] W. Knight. (Apr. 2023). *Openai's Ceo Says the Age of Giant AI Models is Already Over*. WIRED. Accessed: Sep. 15, 2024. [Online]. Available: https://www.wired.com/story/openai-ceo-sam-altman-the-age-of-giant-ai-models-is-already-over/

[47] J. O'Donnell. (Jan. 2025). *Deepseek Might not be Such Good News for Energy After All*. MIT Technol. Rev. [Online]. Available: https://www.technologyreview.com/2025/01/31/1110776/deepseek-might-not-be-such-good-news-for-energy-after-all/

[48] A. Guldner et al., "Development and evaluation of a reference measurement model for assessing the resource and energy efficiency of software products and components—Green software measurement model (GSMM)," *Future Gener. Comput. Syst.*, vol. 155, pp. 402–418, Jun. 2024.

[49] S. Roy, A. Rudra, and A. Verma, "An energy complexity model for algorithms," in *Proc. 4th Conf. Innov. Theor. Comput. Sci.* New York, NY, USA: Association for Computing Machinery, Jan. 2013, pp. 283–304, doi: 10.1145/2422436.2422470.

[50] A. Lastovetsky, L. Szustak, and R. Wyrzykowski, "Model-based optimization of EULAG kernel on Intel xeon phi through load imbalancing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 787–797, Mar. 2017.

[51] E. Kern, L. M. Hilty, A. Guldner, Y. V. Maksimov, A. Filler, J. Gröger, and S. Naumann, "Sustainable software products—Towards assessment criteria for resource and energy efficiency," *Future Gener. Comput. Syst.*, vol. 86, pp. 199–210, Sep. 2018.

[52] J. Mancebo, F. García, and C. Calero, "A process for analysing the energy efficiency of software," *Inf. Softw. Technol.*, vol. 134, Jun. 2021, Art. no. 106560.

[53] N. Rteil, R. Bashroush, R. Kenny, and A. Wynne, "Interact: IT infrastructure energy and cost analyzer tool for data centers," *Sustain. Computing: Informat. Syst.*, vol. 33, Jan. 2022, Art. no. 100618.

[54] T. Kennes, "Measuring IT carbon footprint: What is the current status actually?" 2023, *arXiv:2306.10049*.

[55] S. Kreten, "Modellbildung und umsetzung von methoden zur energieeffizienten nutzung von containertechnologien," Doctoral thesis, Universität Trier, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands, 2022.

[56] P. Lago, D. Greefhorst, and E. Woods, "Architecting for sustainability," in *Proc. 36th Int. Conf. Inform. Environ. Protection, Environ. Inf. Commun. Technol. (EnviroInfo)*, 2022, pp. 199–209.

[57] M. Kambadur and M. A. Kim, "An experimental survey of energy management across the stack," *ACM SIGPLAN Notices*, vol. 49, no. 10, pp. 329–344, Oct. 2014, doi: 10.1145/2714064.2660196.

[58] J. Mancebo, C. Calero, F. Garcia, M. A. Moraga, and I. Garcia-Rodriguez De Guzman, "FEETINGS: Framework for energy efficiency testing to improve environmental goal of the software," *Sustain. Comput., Informat. Syst.*, vol. 30, Jun. 2021, Art. no. 100558.

[59] M. Fahad, A. Shahid, R. R. Manumachu, and A. Lastovetsky, "A comparative study of methods for measurement of energy of computing," *Energies*, vol. 12, no. 11, p. 2204, Jun. 2019. [Online]. Available: https://www.mdpi.com/1996-1073/12/11/2204

[60] M. Fahad, A. Shahid, R. R. Manumachu, and A. Lastovetsky, "Accurate energy modelling of hybrid parallel applications on modern heterogeneous computing platforms using system-level measurements," *IEEE Access*, vol. 8, pp. 93793–93829, 2020.

[61] D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield, "PowerMon: Fine-grained and integrated power monitoring for commodity computer systems," in *Proc. IEEE SoutheastCon (SoutheastCon)*, Mar. 2010, pp. 479–484.

[62] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron, "PowerPack: Energy profiling and analysis of high-performance systems and applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 5, pp. 658–671, May 2010.

[63] J. H. Laros, P. Pokorny, and D. DeBonis, "PowerInsight—A commodity power measurement capability," in *Proc. Int. Green Comput. Conf.*, Jun. 2013, pp. 1–6.

[64] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, "Power-management architecture of the Intel microarchitecture code-named sandy bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20–27, Mar. 2012.

[65] (2024). *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2, Intel Corporation*. Order Number: 253669-060US. [Online]. Available: https://www.intel.com/content/www/us/en/content-details/819715/intel-64-and-ia-32-architectures-software-developer-s-manual-volume-3b-system-programming-guide-part-2.html

[66] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan, "Full-system power analysis and modeling for server environments," in *Proc. Workshop Modeling, Benchmarking, Simulation*, Jan. 2006, pp. 70–77.

[67] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory power estimation and capping," in *Proc. ACM/IEEE Int. Symp. Low-Power Electron. Design (ISLPED)*. New York, NY, USA: Association for Computing Machinery, Aug. 2010, pp. 189–194, doi: 10.1145/1840845.1840883.

[68] C. Gough, I. Steiner, and W. Saunders, *Energy Efficient Servers*. Elk Grove, CA, USA: Apress, Jan. 2015.

[69] R. Kavanagh and K. Djemame, "Rapid and accurate energy models through calibration with IPMI and RAPL," *Concurrency Comput., Pract. Exper.*, vol. 31, no. 13, Jul. 2019, Art. no. e5124, doi: 10.1002/cpe.5124.

[70] J. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta, "Evaluating the effectiveness of model-based power characterization," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, Jun. 2011, p. 12.

[71] K. O'brien, I. Pietri, R. Reddy, A. Lastovetsky, and R. Sakellariou, "A survey of power and energy predictive models in HPC systems and applications," *ACM Comput. Surveys*, vol. 50, no. 3, pp. 1–38, Jun. 2017, doi: 10.1145/3078811.

[72] S. Rivoire, P. Ranganathan, and C. Kozyrakis, "A comparison of high-level full-system power models," in *Proc. Conf. Power Aware Comput. Syst.*, Dec. 2008, p. 3.

[73] (2024). *Perf: Linux Profiling With Performance Counters*. Accessed: Mar. 24, 2025. [Online]. Available: https://perfwiki.github.io/main/

[74] M. McCool, J. Reinders, and A. Robison, *Structured Parallel Programming: Patterns for Efficient Computation*, 1st ed., San Francisco, CA, USA: Morgan Kaufmann, 2012.

[75] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2008.

[76] P. Boldi, M. Santini, and S. Vigna, "PageRank as a function of the damping factor," in *Proc. 14th Int. Conf. World Wide Web WWW*. New York, NY, USA: Association for Computing Machinery, 2005, p. 557, doi: 10.1145/1060745.1060827.

[77] *WebGraph++*. Accessed: Mar. 10, 2024. [Online]. Available: https://cnets.indiana.edu/groups/nan/webgraph/

[78] T. Fontana, S. Vigna, and S. Zacchiroli, "WebGraph: The next generation (is in rust)," in *Proc. Companion ACM Web Conf.*, Singapore, May 2024, pp. 686–689. [Online]. Available: https://hal.science/hal-04494627

[79] A. Moffat and M. Petri, "Large-alphabet semi-static entropy coding via asymmetric numeral systems," *ACM Trans. Inf. Syst.*, vol. 38, no. 4, pp. 1–33, Jul. 2020, doi: 10.1145/3397175.

[80] P. Ferragina, *Pearls of Algorithm Engineering*. Cambridge, U.K.: Cambridge Univ. Press, 2023.

[81] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed., Cambridge, MA, USA: MIT Press, 2009.

[82] N. R. Brisaboa, T. Gagie, A. Gómez-Brandón, and G. Navarro, "Two-dimensional block trees," *Comput. J.*, vol. 67, no. 1, pp. 391–406, Jan. 2024, doi: 10.1093/comjnl/bxac182.

[83] S. Ladra, J. R. Paramá, and F. Silva-Coira, "Scalable and queryable compressed storage structure for raster data," *Inf. Syst.*, vol. 72, pp. 179–204, Dec. 2017.

[84] N. R. Brisaboa, A. Cerdeira-Pena, G. de Bernardo, G. Navarro, and Ó. Pedreira, "Extending general compact querieable representations to GIS applications," *Inf. Sci.*, vol. 506, pp. 196–216, Jan. 2020, doi: 10.1016/j.ins.2019.08.007.

[85] S. Gog, T. Beller, A. Moffat, and M. Petri, "From theory to practice: Plug and play with succinct data structures," in *Proc. 13th Int. Symp. Exp. Algorithms*, Copenhagen, Denmark, Jan. 2014, pp. 326–337.

[86] D. Arroyuelo, A. Gómez-Brandón, and G. Navarro, "Evaluating regular path queries on compressed adjacency matrices," in *Proc. 30th Int. Symp. String Process. Inf. Retr.*, vol. 34, Pisa, Italy, Nov. 2024, pp. 35–48.

[87] N. J. Larsson and A. Moffat, "Off-line dictionary-based compression," *Proc. IEEE*, vol. 88, no. 11, pp. 1722–1732, Nov. 2000.

[88] A. Elgohary, M. Boehm, P. J. Haas, F. R. Reiss, and B. Reinwald, "Compressed linear algebra for large-scale machine learning," *VLDB J.*, vol. 27, no. 5, pp. 719–744, Oct. 2018.

[89] C. Hernández and G. Navarro, "Compressed representations for web and social graphs," *Knowl. Inf. Syst.*, vol. 40, no. 2, pp. 279–313, Aug. 2014, doi: 10.1007/s10115-013-0648-4.

[90] P. Boldi, M. Rosa, M. Santini, and S. Vigna, "Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks," in *Proc. 20th Int. Conf. World Wide Web*. New York, NY, USA: Association for Computing Machinery, Mar. 2011, pp. 587–596, doi: 10.1145/1963405.1963488.

[91] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan, "On compressing social networks," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*. New York, NY, USA: Association for Computing Machinery, Jun. 2009, pp. 219–228, doi: 10.1145/1557019.1557049.

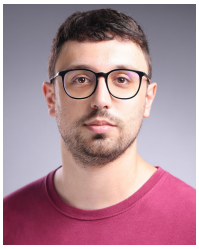
[92] P. Carazo, R. Apolloni, F. Castro, D. Chaver, L. Pinuel, and F. Tirado, "Reducing cache hierarchy energy consumption by predicting forwarding and disabling associative sets," *J. Circuits, Syst. Comput.*, vol. 21, no. 7, Nov. 2012, Art. no. 1250057.

[93] K. Kesrouani, H. Kanso, and A. Noureddine, "A preliminary study of the energy impact of software in raspberry pi devices," in *Proc. IEEE 29th Int. Conf. Enabling Technol., Infrastruct. Collaborative Enterprises (WETICE)*, Sep. 2020, pp. 231–234.

[94] *PZEM-016 AC Power Meter User Manual*, Peacefair Electronics, Shenzhen, China, 2018. [Online]. Available: https://images-na.ssl-images-amazon.com/images/I/81GtkIOyZaL.pdf

[95] *Fluke 8845A/8846A 6.5 Digit Precis. Multimeters User Manual*, Fluke Corporation, Everett, WA, USA, 2006. [Online]. Available: https://assets.fluke.com/manuals/884xaumeng0200.pdf

[96] A. Bhattacharya, A. Chowdhury, H. Xu, R. Das, R. A. Chowdhury, R. Johnson, R. Nithyanand, and M. A. Bender, "When are cache-oblivious algorithms cache adaptive? A case study of matrix multiplication and sorting," in *Proc. 30th Annu. Eur. Symp. Algorithms (ESA)*, vol. 244, S. Chechik, G. Navarro, E. Rotenberg, and G. Herman, Eds., Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum Für Informatik, 2022, pp. 16:1–16:17, doi: 10.4230/LIPIcs.ESA.2022.16.



FRANCESCO TOSONI received the bachelor's degree in computer and electronic engineering from the University of Perugia, in 2017, the master's degree in computer science and networking from the University of Pisa and the Sant'Anna School of Advanced Studies, in 2020, and the Ph.D. degree in computer science from the University of Pisa, in 2024. He is currently a Research Fellow of the Computer Science Department, University of Pisa, where his current research interests include compressed data structures, key-value stores, parallel and I/O-efficient indexing techniques for large source-code archives, and green algorithm engineering. He won the 2020 Best Master's Thesis Award from con.Scienze and Italian National Conference of Presidents and Directors of Scientific and Technological University Facilities.

PHILIP BILLE received the Ph.D. degree in computer science from the IT University of Copenhagen, in 2007. He is currently a Professor in computer science and the Head of Section with the Technical University of Denmark. His research interests include compressed computation, combinatorial pattern matching, and data structures.

VALERIO BRUNACCI (Graduate Student Member, IEEE) received the M.Sc. degree in computer engineering and robotics from the Department of Engineering, University of Perugia, Perugia, Italy, in 2021, where he is currently pursuing the Ph.D. degree in industrial and information engineering. His research interests include localization systems, ultrawideband (UWB) and magnetic ranging systems (RSs), and robotics.

ALESSIO DE ANGELIS (Member, IEEE) received the Ph.D. degree in information engineering from the University of Perugia, Perugia, Italy, in 2009. From 2010 to 2013, he was a Researcher with the Signal Processing Laboratory, KTH Royal Institute of Technology, Stockholm, Sweden. Since July 2013, he has been with the Department of Engineering, University of Perugia, where he became an Associate Professor, in May 2018. His research interests include instrumentation and measurement, positioning systems, and battery measurement and modeling. He has been an Associate Editor of IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT, since 2019.

PAOLO FERRAGINA received the Ph.D. degree in computer science from the University of Pisa, in 1996. He was a Postdoctoral Researcher with the Max Planck Institute for Informatics, Saabrcken, Germany, from 1997 to 1998. He is currently a Professor in computer science with the L'EMbeDS Department, Sant'Anna School of Advanced Studies, Pisa. His research interests include regard the design of algorithms for big data, mainly in the form of texts and graphs, in collaboration with companies worldwide, such as Bloomberg, European Broadcasting Union (EBU), Google, Tiscali, and Yahoo! He has co-authored more than 190 (refereed) publications, some books, and chapters, achieving an H-index of 34 on Scopus and more than 11 000 citations on Google Scholar. His research results got five U.S. patents and some international awards, such as the 1995 Best Land Transportation Paper Award from IEEE Vehicular Technology Society, the 1997 Best Ph.D. Thesis in Theoretical Computer Science by the Italian Chapter of the EATCS, the 1997 Philip Morris Award on Science and Technology, a Yahoo! Research Faculty Award, the three Google Faculty research awards, the 2022 ACM Paris Kanellakis Award, and two Test of Time awards for his papers appeared in the conference proceedings of ACM CIKM and ESA. He is serving on the editorial board for the *Journal of Graph Algorithms and Applications* (JGAA). He is an Area Editor of *Encyclopedias of Algorithms* (Springer) and *Encyclopedias of Big Data Technologies* (Springer).

GIOVANNI MANZINI received the Ph.D. degree in mathematics from the Scuola Normale Superiore of Pisa, in 1995. He is currently a Professor in computer science with the University of Pisa and a Research Associate with the Institute of Informatics and Telematics, National Research Council. He has been a Visiting Scientist with Massachusetts Institute of Technology and a Visiting Professor with Johns Hopkins University and the University of Melbourne. His current research interests include the design of algorithms and data structures for solving theoretical and applied problems in data compression and indexing data structures for massive data sets. His research results got the 2022 ACM Paris Kanellakis Award and a Test of Time award for one of his papers that appeared in the conference proceedings of ESA.