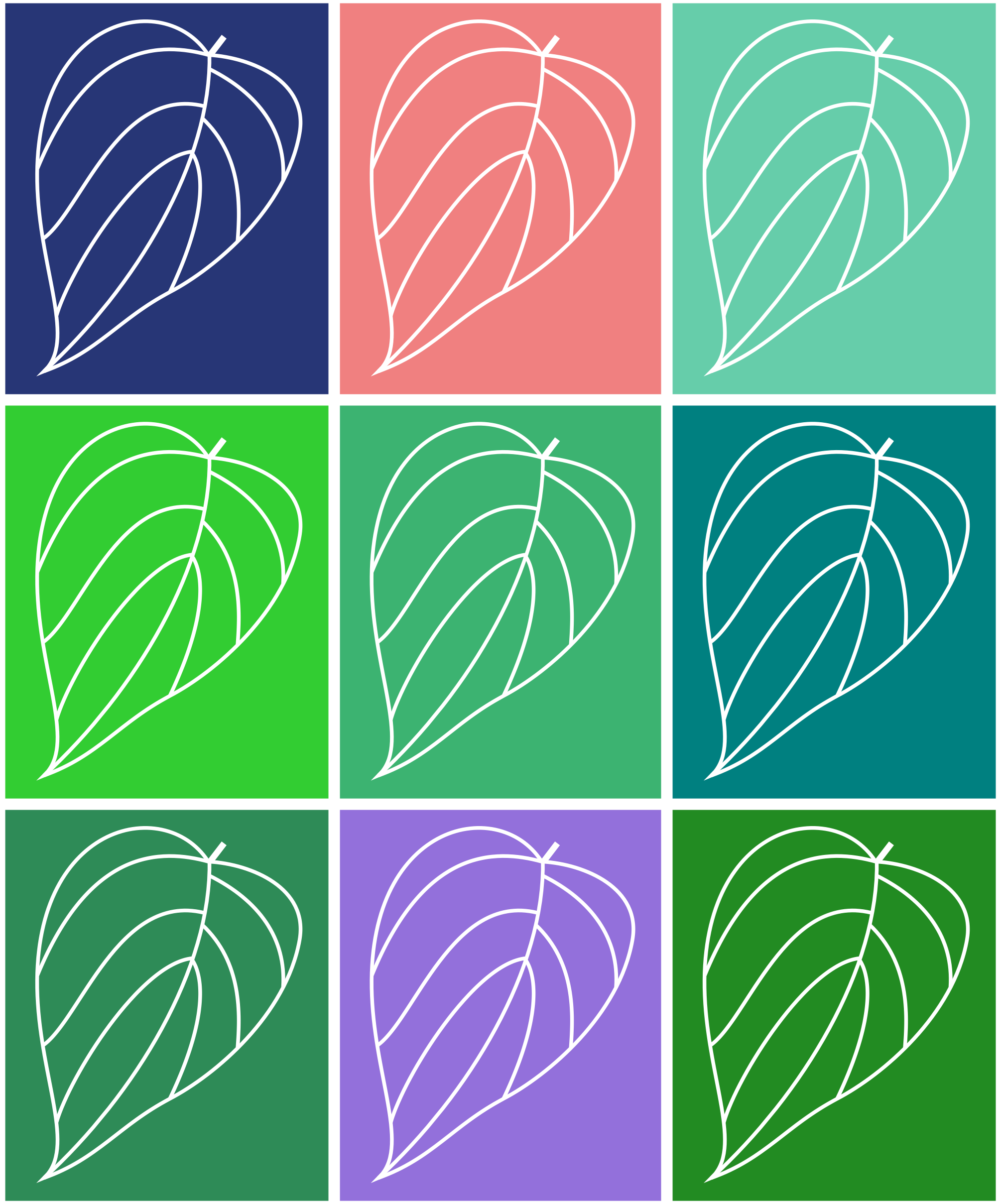
Francesco Tosoni, Philip Bille,
Valerio Brunacci,
Alessio De Angelis, Paolo Ferragina,
and Giovanni Manzini

Toward Greener Matrix Operations by Lossless Compressed Formats

A.D. 1308 unipg
DIPARTIMENTO
DI INGEGNERIA
DIPARTIMENTO DI ECCELLENZA
MUR 2023/2027

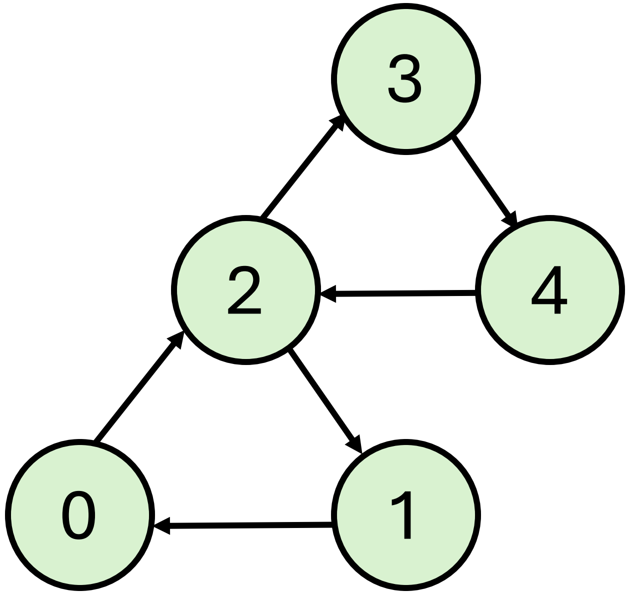


Introduction

Sparse Matrix-Vector Multiplication (**SpMV**) are relevant in ML, scientific computing, and graph algorithms.

Research focus: Investigate **space**, **time**, and **energy** efficiency of SpMV.

We challenge prevailing assumptions about a straightforward linear correlation between time and energy.



$$\underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}}_{M} \times \underbrace{\begin{bmatrix} 2.1 \\ 1.5 \\ 3.8 \\ 0.7 \\ 4.4 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 3.8 \\ 2.1 \\ 2.2 \\ 4.4 \\ 3.8 \end{bmatrix}}_{y}$$

Motivation

POPULARITY OF ML ALGORITHMS (E.G. CHATGPT) AND DATA GENERATION SURPASSING MOORE'S LAW

AI NOT JUST ON SERVER MACHINES BUT ALSO ON EDGE AND IOT DEVICES → BATTERY DURATION AS CRITICAL CONCERN.

ENERGY AS A LEADING DESIGN CONSTRAINT IN COMPUTING DEVICES; HOWEVER, GREEN SOFTWARE ENGINEERING IS STILL IN ITS INFANCY.

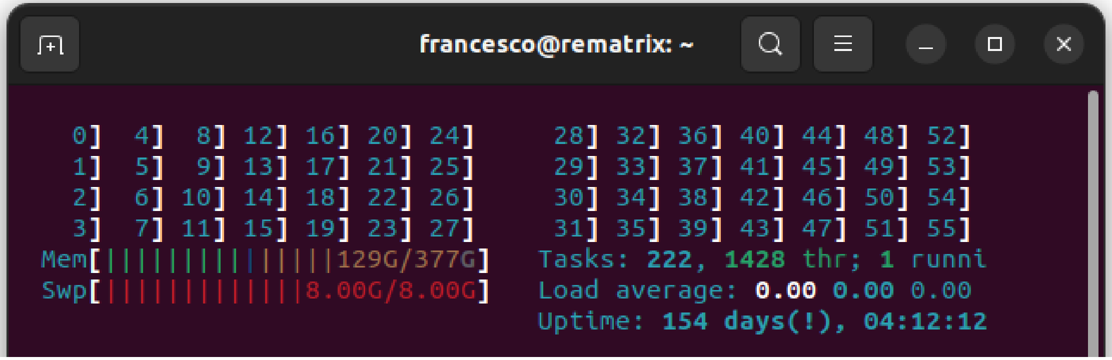
OVERSIMPLIFIED ENERGY COMPLEXITY MODELS FAIL TO CAPTURE REAL-WORLD DYNAMICS → NEED FOR COMPREHENSIVE REFERENCE MODELS

Platforms

Raspberry Pi 4 model B @1.5GHz (4 cores) and a Fluke 8845A benchtop **multimeter**

vs.

Intel® Xeon® Gold 6132 CPU @2.60GHz (28 cores, 2-way hyperthreading) and the **RAPL** energy profiler



Research questions

How does compression affect the efficiency of space, time, and energy? Some matrix formats are more space-efficient but consume order of magnitude more energy than others.

What trade-offs exist between time optimisation and energy consumption? Often the energy-optimal parallelism degree is lower than the time-optimal one.

Which runtime metrics impact energy efficiency? The number of L1 and L3 cache accesses impact performances.

Compressed matrix formats

VLDB 2022

UNIVERSIDADE DA CORUÑA

UNIVERSIDAD DE CHILE

Compressed matrix formats

We compare *three* computation-friendly compression schemes for **large** yet sparse **binary matrices**:

- Google's Zuckerli (Versari *et al.*, 2020)
- k^2-tree (Brisaboa, Ladra, and Navarro 2014)
- RePair-compressed matrices [mm-repair] (Ferragina *et al.*, 2022)

Why lossless?

Lossy compression solutions for space reduction:

- Low-precision storage (e.g. FP32)
- Sparsification
- Quantisation (binary, ternary)

→careful & manual application

Lossless compression is a better "automated" alternative

- data independent
- no need a priori knowledge about the input data.

Computation-friendly compression

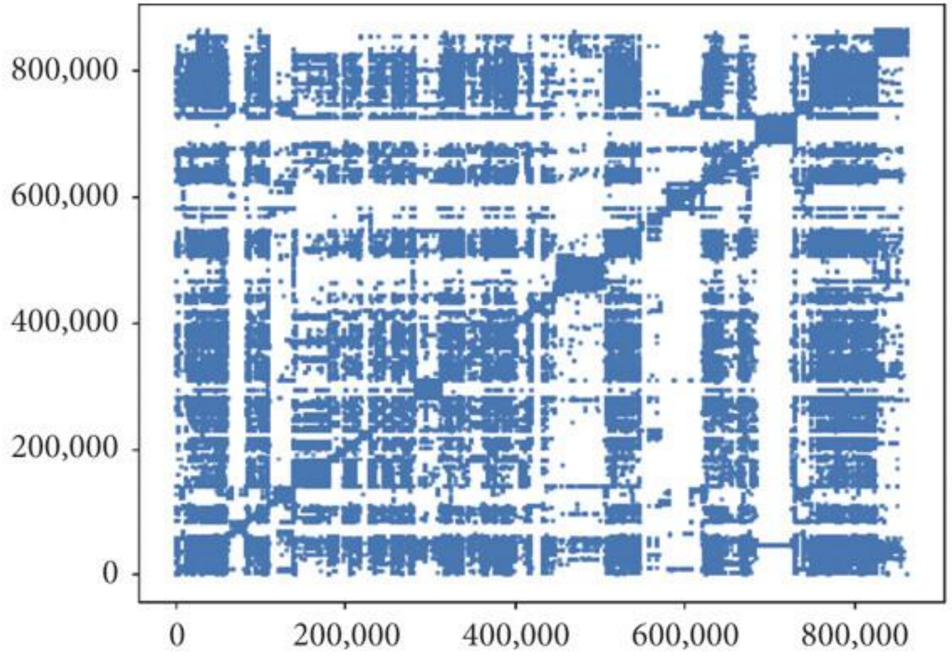
All formats we tested are **computation-friendly** for matrix-vector multiplications:

- Exploit more than mere sparsity
- Enable direct operations on data without prior decompression
- Allow operating in time proportional to the size of the compressed representation (*win-win*!)

WebGraph & Zuckerli (1/2)

Exploit redundancies of outgoing links within the same domain.

- **Webgraph** (2004) exploit the copying property of consecutive adjacency lists to compress each list based on a reference list (Java, c++, Rust).
- Google's **Zuckerli** (2020) applies novel compression heuristics on top of Zuckerli.



Adjacency matrix for eu-2005. Figure taken from DOI: 10.1155/2020/2354875

WebGraph & Zuckerli (2/2)

These graph formats allow for *compression-friendly* matrix-to-vector multiplications (Francisco *et al*. 2022).

Idea: exploit deltas between similar adjacency lists.

We implement multiplications on top of **Zuckerli**.

Node	Outdegree	Successors
…	…	…
15	11	13, 15, 16, 17 , 18, 19, 23, 24, 203, 315, 316, 317, 3041
16	10	15, 16, 17, 22, 23, 24, 315, 316, 317, 3041
17	0	
18	5	13, 15, 16, 17, 50
…	…	…

k^2-tree (1/3)

Exploit sparsity and clustering of **0**'s.

Submatrices are recursively split into k^2 smaller submatrices.

- **0**: empty submatrix;
- **1**: non-empty ones.

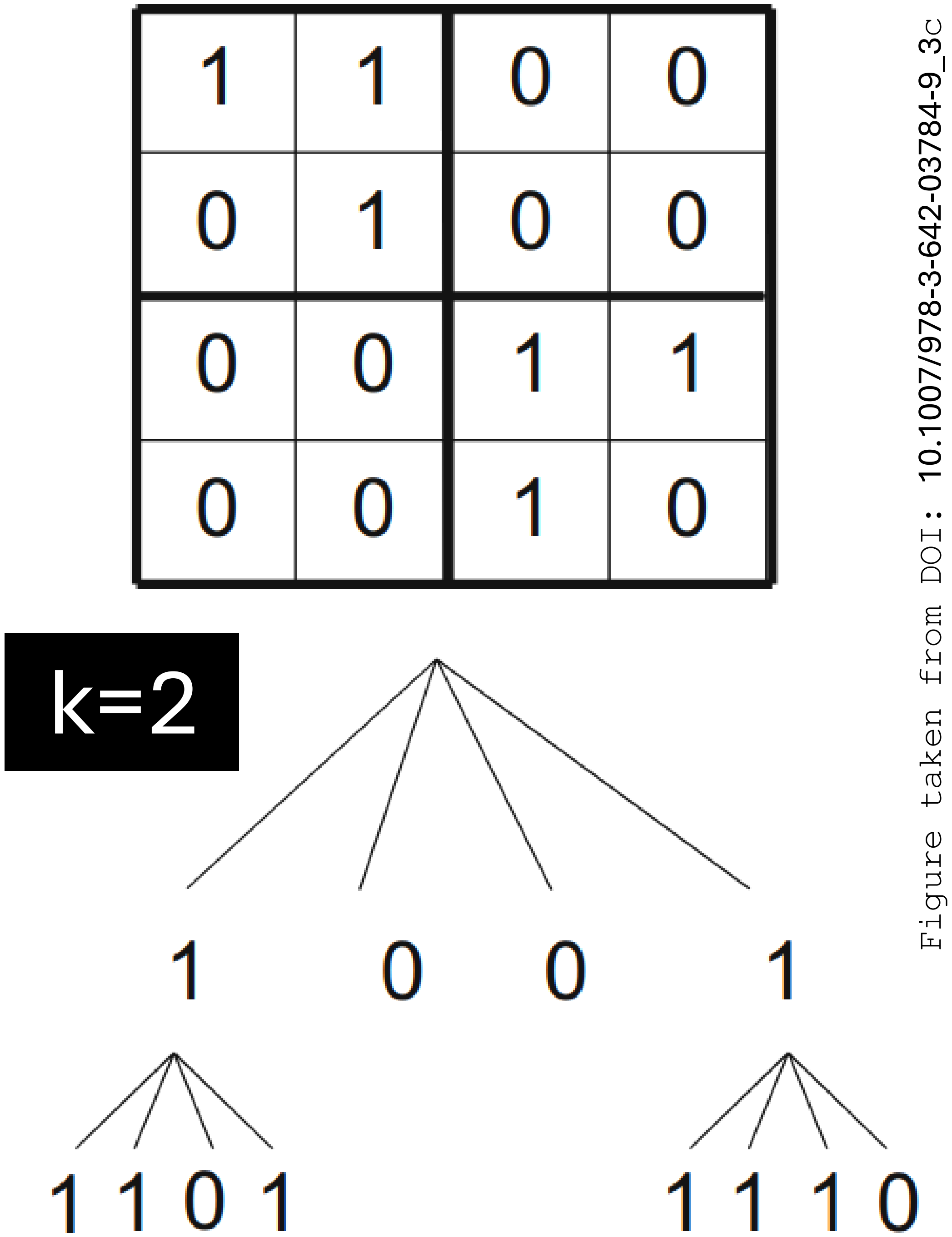


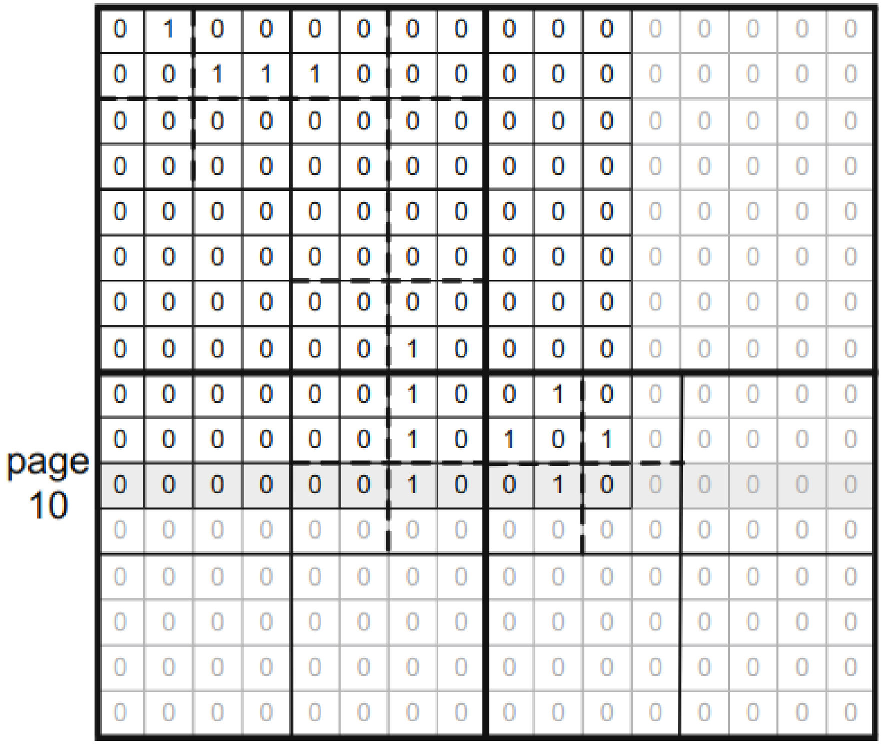
Figure taken from DOI: 10.1007/978-3-642-03784-9_3c

k^2-tree (2/3)

Height is always $\mathbf{log_k n}$ → access to each single cell $\mathcal{O}(\mathbf{log_k n})$.

Matrix-vector **multiplication** = visit to the tree

Retrieving neighbours for page 10



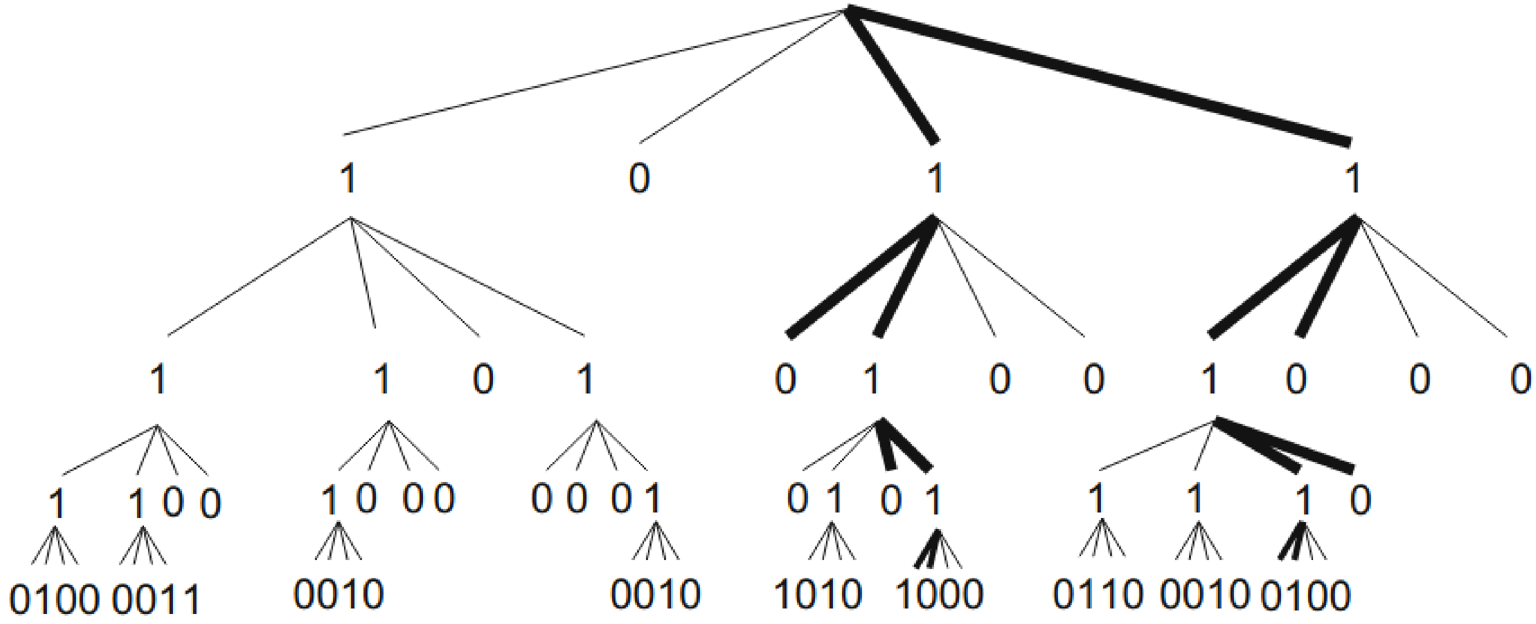


Figure taken from DOI: 10.1007/978-3-642-03784-9_3c

k^2-tree (3/3)

Implementations: University of A Coruña, SDSL library, and one by the University of Chile/Millennium Institute.

Matrix-to-vector multiplications can be implemented by a tree traversal (in any order). There's **no need of `rank`** and `select` data structures.

mm-repair ~ Step #1: exploit sparsity

Compute **the CSRV representation of a matrix**, a modification of the CSR representation.

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

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

Set of nonzeros

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

Sequence of pairs

\$ is the EOL character

mm-repair ~ Step #1: exploit sparsity

$$
\begin{array}{c}
\begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\
\begin{bmatrix}
\boxed{4.1} & 5.3 & 0 & 5.3 \\
4.1 & 0 & 2.0 & 4.1 \\
0 & 5.3 & 2.0 & 5.3 \\
4.1 & 5.3 & 2.0 & 4.1 \\
4.1 & 5.3 & 2.0 & 5.3
\end{bmatrix}
\end{array}
$$

$$
\downarrow
$$

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

$S =$
$\boxed{\langle 2,1 \rangle}\langle 3,2 \rangle\langle 3,4 \rangle\$\langle 2,1 \rangle\langle 1,3 \rangle\langle 2,4 \rangle\$$
$\langle 3,2 \rangle\langle 1,3 \rangle\langle 3,4 \rangle\$\langle 2,1 \rangle\langle 3,2 \rangle\langle 1,3 \rangle\langle 2,4 \rangle\$$
$\langle 2,1 \rangle\langle 3,2 \rangle\langle 1,3 \rangle\langle 3,4 \rangle\$$

Generate column-value index pairs

$$\langle l, j \rangle$$

Non-zero index

Column index

mm-repair ~ Step #1: exploit sparsity

$$\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 4.1 & \boxed{5.3} & 0 & 5.3 \\ 4.1 & 0 & 2.0 & 4.1 \\ 0 & 5.3 & 2.0 & 5.3 \\ 4.1 & 5.3 & 2.0 & 4.1 \\ 4.1 & 5.3 & 2.0 & 5.3 \end{bmatrix} \end{array}$$

$$\begin{array}{cccc} & 1 & 2 & 3 \\ V = (& 2.0 & 4.1 & \boxed{5.3}\ ) \end{array}$$

$S =$
$\langle 2,1\rangle\boxed{\langle 3,2\rangle}\langle 3,4\rangle\$\langle 2,1\rangle\langle 1,3\rangle\langle 2,4\rangle\$$
$\langle 3,2\rangle\langle 1,3\rangle\langle 3,4\rangle\$\langle 2,1\rangle\langle 3,2\rangle\langle 1,3\rangle\langle 2,4\rangle\$$
$\langle 2,1\rangle\langle 3,2\rangle\langle 1,3\rangle\langle 3,4\rangle\$$

Generate column-value index pairs

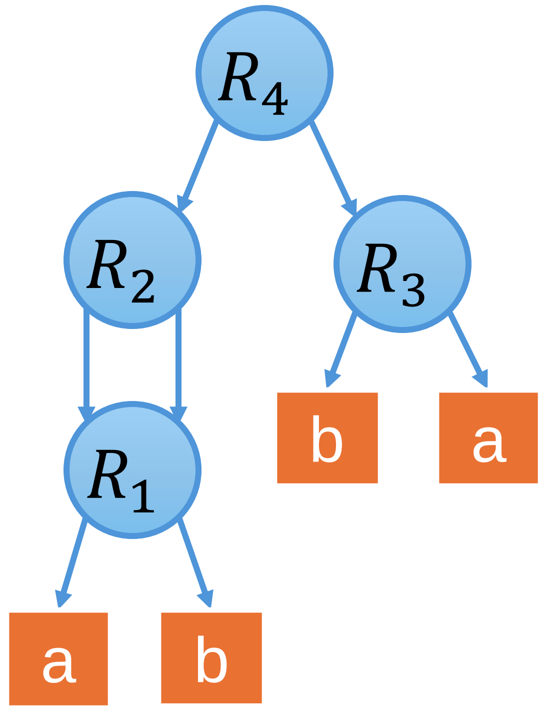
$$\langle l, j\rangle$$

Non-zero index

Column index

mm-repair ~ Step #2: grammar-compress

In the following, as a (lossless) compressor we use RePair, which is a **grammar compressor** based on **straight-line programs (SLP).**



$$S = ababba$$

$$R_1 \to ab$$
$$R_2 \to R_1 R_1$$
$$R_3 \to ba$$
$$R_4 \to R_2 R_3$$

mm-repair ~ Step #2: grammar-compress

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

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

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

$$\left.\begin{array}{l} N_1 \rightarrow \langle 2,1\rangle\langle 3,2\rangle \\ N_2 \rightarrow \langle 1,3\rangle\langle 2,4\rangle \\ N_3 \rightarrow \langle 1,3\rangle\langle 3,4\rangle \\ N_4 \rightarrow N_1\langle 3,4\rangle \\ N_5 \rightarrow \langle 2,1\rangle N_2 \\ N_6 \rightarrow \langle 3,2\rangle N_3 \\ N_7 \rightarrow N_1 N_2 \\ N_8 \rightarrow N_1 N_3 \end{array}\right\} R$$

rules

final string

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

mm-repair ~ Step #2: grammar-compress

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

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

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

$$\left.\begin{array}{l}
N_1 \to \langle 2,1\rangle\langle 3,2\rangle \\
N_2 \to \langle 1,3\rangle\langle 2,4\rangle \\
N_3 \to \langle 1,3\rangle\langle 3,4\rangle \\
N_4 \to N_1\langle 3,4\rangle \\
N_5 \to \langle 2,1\rangle N_2 \\
N_6 \to \langle 3,2\rangle N_3 \\
N_7 \to N_1 N_2 \\
N_8 \to N_1 N_3
\end{array}\right\} R$$

rules

final string

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

mm-repair ~ Step #2: grammar-compress

nonzeros

$$\begin{array}{cccc} & 1 & 2 & 3 \\ V = & (2.0 & 4.1 & 5.3) \end{array}$$

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

final string

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

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

$$\left.\begin{array}{l} N_1 \to \langle 2,1\rangle\langle 3,2\rangle \\ N_2 \to \langle 1,3\rangle\langle 2,4\rangle \\ N_3 \to \langle 1,3\rangle\langle 3,4\rangle \\ N_4 \to N_1\langle 3,4\rangle \\ N_5 \to \langle 2,1\rangle N_2 \\ N_6 \to \langle 3,2\rangle N_3 \\ N_7 \to N_1 N_2 \\ N_8 \to N_1 N_3 \end{array}\right\} R$$

rules

mm-repair ~ Step #3: multiply

nonzeros

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

$$y = M \begin{array}{c} x \\ \begin{pmatrix} 2 \\ 0 \\ 2 \\ 4 \end{pmatrix} \end{array}$$

$$\langle l, j \rangle \quad \Rightarrow \quad eval_x(\langle l, j \rangle) = V[l] \cdot x[j]$$

$$N_i \to AB \quad \Rightarrow \quad eval_x(N_i) = eval(A) + eval(B)$$

$\langle 2,1 \rangle$

$\langle 3,2 \rangle$

$\langle 1,3 \rangle$

$\langle 2,4 \rangle$

$\langle 3,4 \rangle$

$$\left.\begin{array}{l} N_1 \to \langle 2,1 \rangle \langle 3,2 \rangle \\ N_2 \to \langle 1,3 \rangle \langle 2,4 \rangle \\ N_3 \to \langle 1,3 \rangle \langle 3,4 \rangle \\ N_4 \to N_1 \langle 3,4 \rangle \\ N_5 \to \langle 2,1 \rangle N_2 \\ N_6 \to \langle 3,2 \rangle N_3 \\ N_7 \to N_1 N_2 \\ N_8 \to N_1 N_3 \end{array}\right\} R$$

rules

mm-repair ~ Step #3: multiply

nonzeros

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

$$y = M \begin{array}{c} x \\ \begin{pmatrix} 2 \\ 0 \\ 2 \\ 4 \end{pmatrix} \end{array}$$

$$\langle l, j \rangle \quad \Rightarrow \quad eval_x(\langle l, j \rangle) = V[l] \cdot x[j]$$

$$N_i \rightarrow AB \quad \Rightarrow \quad eval_x(N_i) = eval(A) + eval(B)$$

8.2	$\langle 2,1 \rangle$
0	$\langle 3,2 \rangle$
4.0	$\langle 1,3 \rangle$
16.4	$\langle 2,4 \rangle$
21.2	$\langle 3,4 \rangle$

8.2	$N_1 \rightarrow \langle 2,1 \rangle \langle 3,2 \rangle$
20.4	$N_2 \rightarrow \langle 1,3 \rangle \langle 2,4 \rangle$
25.2	$N_3 \rightarrow \langle 1,3 \rangle \langle 3,4 \rangle$
29.4	$N_4 \rightarrow N_1 \langle 3,4 \rangle$
28.6	$N_5 \rightarrow \langle 2,1 \rangle N_2$
25.2	$N_6 \rightarrow \langle 3,2 \rangle N_3$
28.6	$N_7 \rightarrow N_1 N_2$
33.4	$N_8 \rightarrow N_1 N_3$

R

rules

mm-repair ~ Step #3: multiply

nonzeros

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

$$y = M \begin{array}{c} x \\ \begin{pmatrix} 2 \\ 0 \\ 2 \\ 4 \end{pmatrix} \end{array}$$

$\langle l, j \rangle \quad \Rightarrow \quad eval_x(\langle l, j \rangle) = V[l] \cdot x[j]$

$N_i \rightarrow AB \quad \Rightarrow \quad eval_x(N_i) = eval(A) + eval(B)$

$O(|R|)$ extra space

$O(|R| + |C|)$ time

8.2	$\langle 2,1 \rangle$
0	$\langle 3,2 \rangle$
4.0	$\langle 1,3 \rangle$
16.4	$\langle 2,4 \rangle$
21.2	$\langle 3,4 \rangle$

8.2	$N_1 \rightarrow \langle 2,1 \rangle \langle 3,2 \rangle$
20.4	$N_2 \rightarrow \langle 1,3 \rangle \langle 2,4 \rangle$
25.2	$N_3 \rightarrow \langle 1,3 \rangle \langle 3,4 \rangle$
29.4	$N_4 \rightarrow N_1 \langle 3,4 \rangle$
28.6	$N_5 \rightarrow \langle 2,1 \rangle N_2$
25.2	$N_6 \rightarrow \langle 3,2 \rangle N_3$
28.6	$N_7 \rightarrow N_1 N_2$
33.4	$N_8 \rightarrow N_1 N_3$

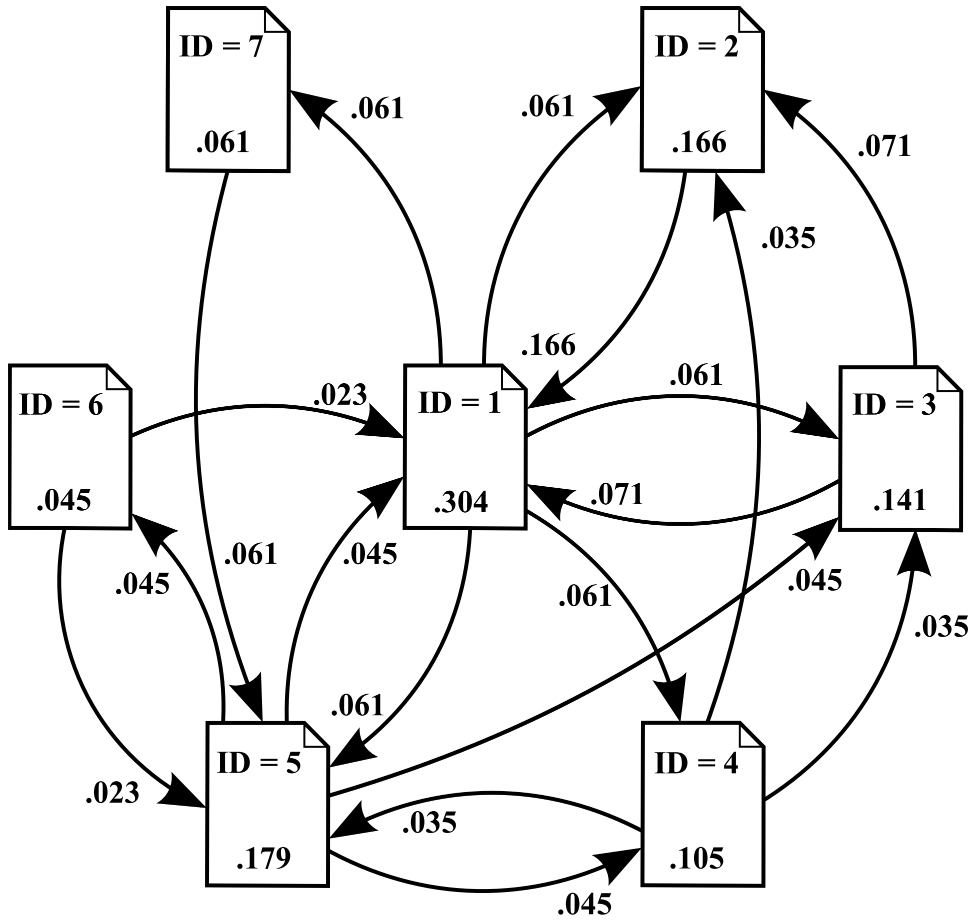
R rules

	Final string C	Rules R	Nonzeros V
re_32	32 bit	32 bit	64 bit
re_iv	packed arrays	packed arrays	64 bit
re_ans	ANS (entropy coder)	packed arrays	64 bit

mm-repair ~ physical representations

Different space–time tradeoffs

Experimental setup



PageRank

A classical algorithm in graph analysis.

Teleporting parameter (α=0.15)

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

Initial probability distribution

Left matrix-vector multiplication

File:Rhacophorus nigropalmatus.jpg originally uploaded to Commons by User:Rushenb

File:Linkstruct2.svg uploaded to Commons in SVG format by User:RobertG

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	✓

Datasets

From the WebGraph framework, via the SuiteSparse Matrix Collection. Most graphs are derived from **web crawls**, with vertices ordered by the reversed URL lexicographically.

We also included two **social network** graphs:

- `hollywood-2009`: An undirected graph representing movie actors, where edges connect actors who co-starred in films.
- `ljournal-2008`: A directed graph illustrating asymmetric friendships in the LiveJournal social network.

Specifications

Intel® Xeon® Server

CPU: 2 x Intel® Xeon® Gold 6132 @ 2.60GHz
Cores: 28 physical cores (56 logical)

Memory:
Total RAM: 384 GB (12 x 32 GB DDR4)
Memory Speed: 2666 MT/s

Operating system:
Ubuntu 22.04.3 LTS (64-bit)

Cache architecture:
L1d: 896 KiB (8-way set associative)
L1i: 896 KiB (8-way set associative)
L2: 28 MiB (16-way set associative)
L3: 38.5 MiB (11-way set associative)

Raspberry Pi 4 Model B

- **CPU**: 4 x ARM Cortex-A72 @ 1.5GHz
- **Cores**: 4 physical cores

Memory:
- Total RAM: 4 GB LPDDR4 SDRAM

Operating system:
- Ubuntu Server 24.04 LTS (64-bit)

Cache architecture:
- L1d: 128 KiB per core; L1i: 192 KiB per core
- L2: 1 MB shared (16-way set associative)
- No L3!

Datasets Used: $\leq 10^7$ vertices

Code setup

Preprocessing steps

Transposed the matrix

Compress via Zuckerli, k2-tree, mm-repair

Store out-degree array O[1, n] for each vertex

Experiment execution

Executed 100 iterations of PageRank

Repeated for each compression format and dataset

Code optimisation

All codes written in C/C++

Compiled with -O3 flag for maximum optimization → energy savings (<43% compared to –O0)

Data parallelism

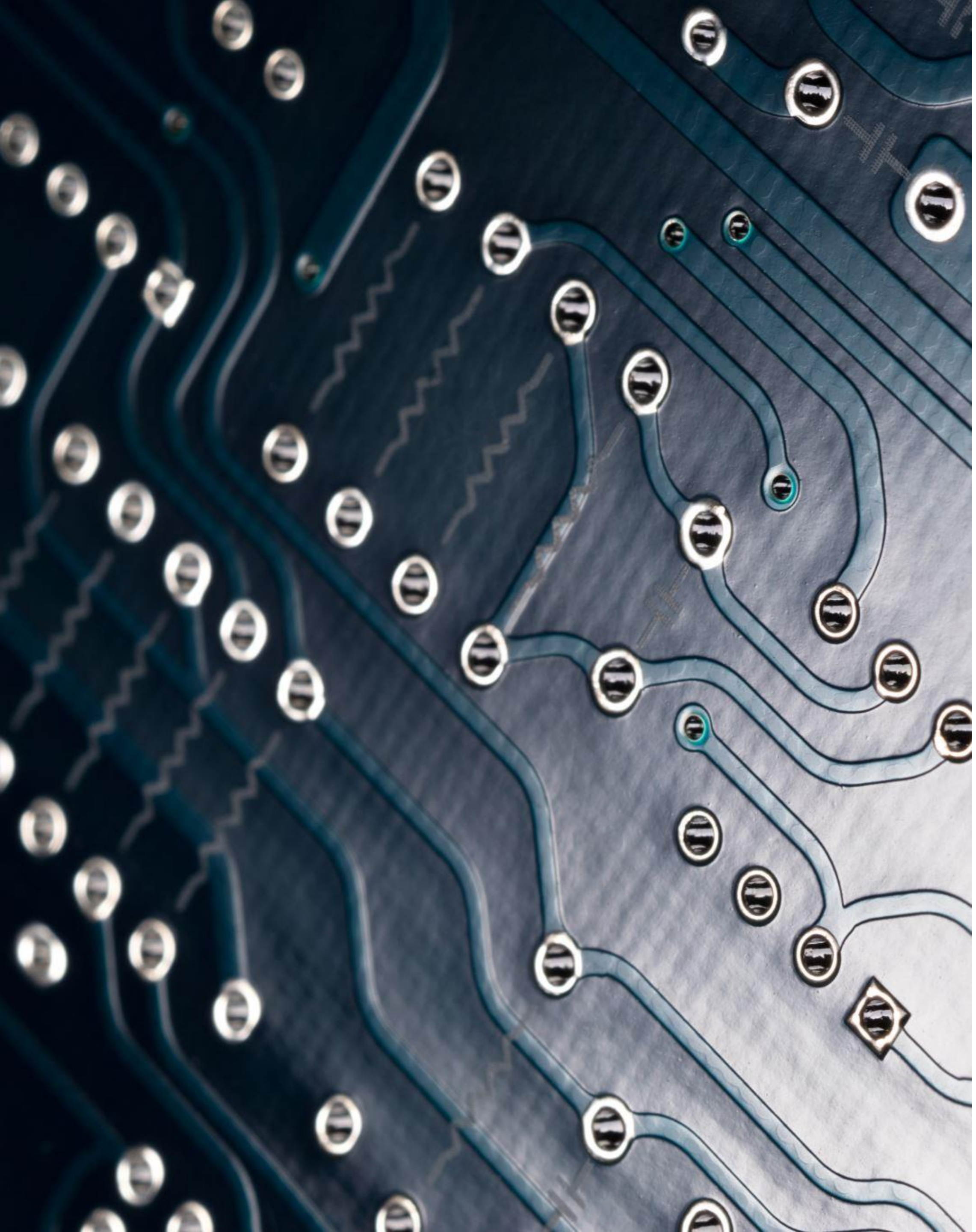
$$
\begin{bmatrix}
1.2 & 3.4 & 5.6 & 0 & 2.3 \\
2.3 & 0 & 2.3 & 4.5 & 1.7 \\
\hline
1.2 & 3.4 & 2.3 & 4.5 & 0 \\
3.4 & 0 & 5.6 & 0 & 2.3 \\
\hline
2.3 & 0 & 2.3 & 4.5 & 0 \\
1.2 & 3.4 & 2.3 & 4.5 & 3.4
\end{bmatrix}
$$

b=3

We exploit multicore architectures with **data-parallel versions**.

We divide each matrix into b row blocks. y=Mx consists of b independent multiplications over a single block.

We use C/C++ with POSIX Threads (Pthreads) and fork-join mechanisms



Power Measurement
on the Intel® Xeon® Server

Intel RAPL (Running Average Power Limit) interface, accessible via the Linux profiler **`perf`** (version 5.15.149).

Energy estimations at
- Core level: All cores in a processor
- Package level: Includes cores, memory controller, last-level cache, and other components

Accuracy:
- RAPL offers reasonably accurate measurements, as supported by prior studies.
- Integrated with tools like *Scaphandre*, *CodeCarbon*, and *Green Metrics Tool*.

Metrics Collected via `perf`: Energy estimations, CPU cycles, cache hits/misses, and instruction counts.

Power Measurement
on Raspberry Pi 4

Current drawn during PageRank computations measured using a **Fluke 8845A benchtop multimeter**.

- Multimeter connected in series with the USB-C power cable.
- Configured to a range of **10 A,** with a resolution of **4½ digits** and a sampling frequency of **3 Hz**.

Sampling rate deemed sufficient for accurately reconstructing the current signal with high fidelity.

Results and discussion

Baselines for compression ratios

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

more compressed

Disk occupancy

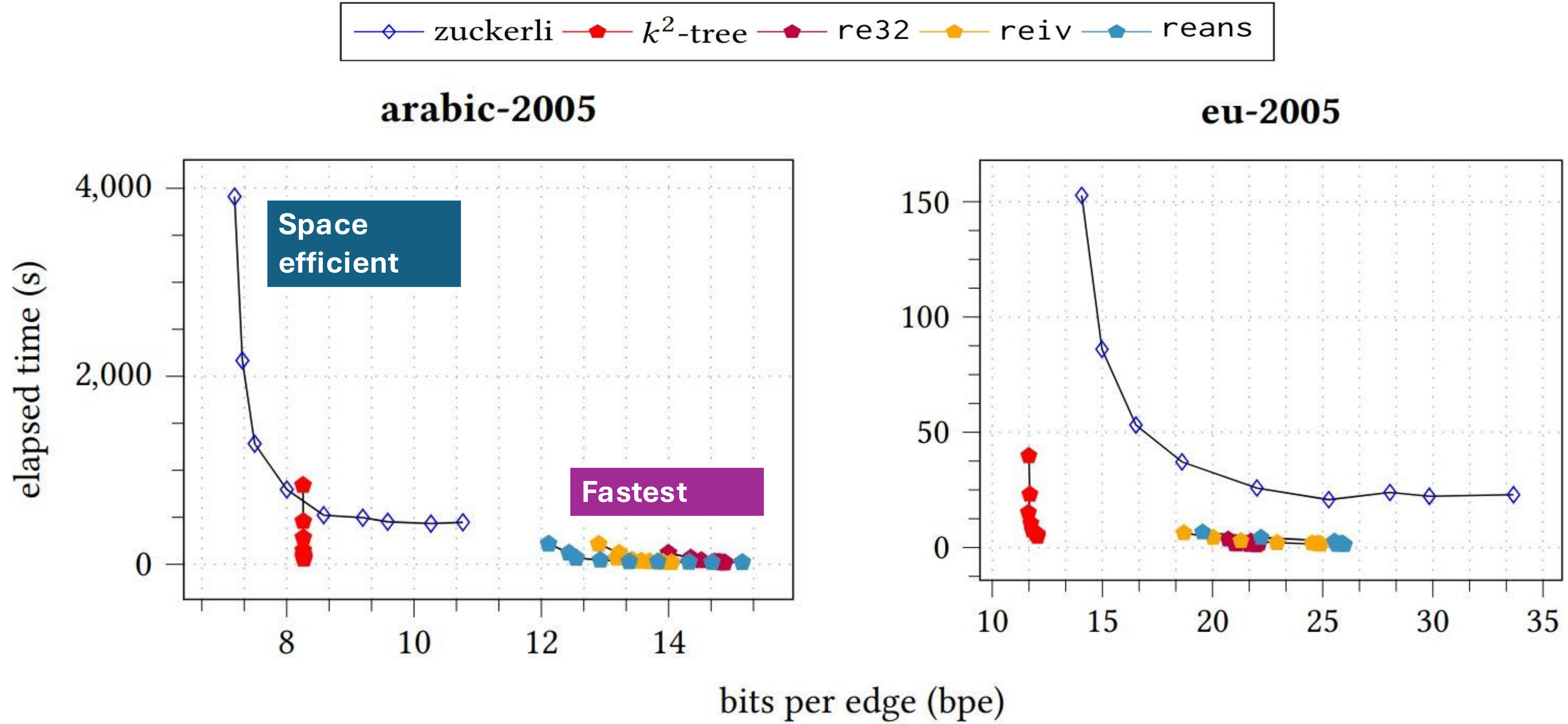
`gzip`: Variable compression ratios compared to Zuckerli

`xz`: Most compressed but the slowest in decompression

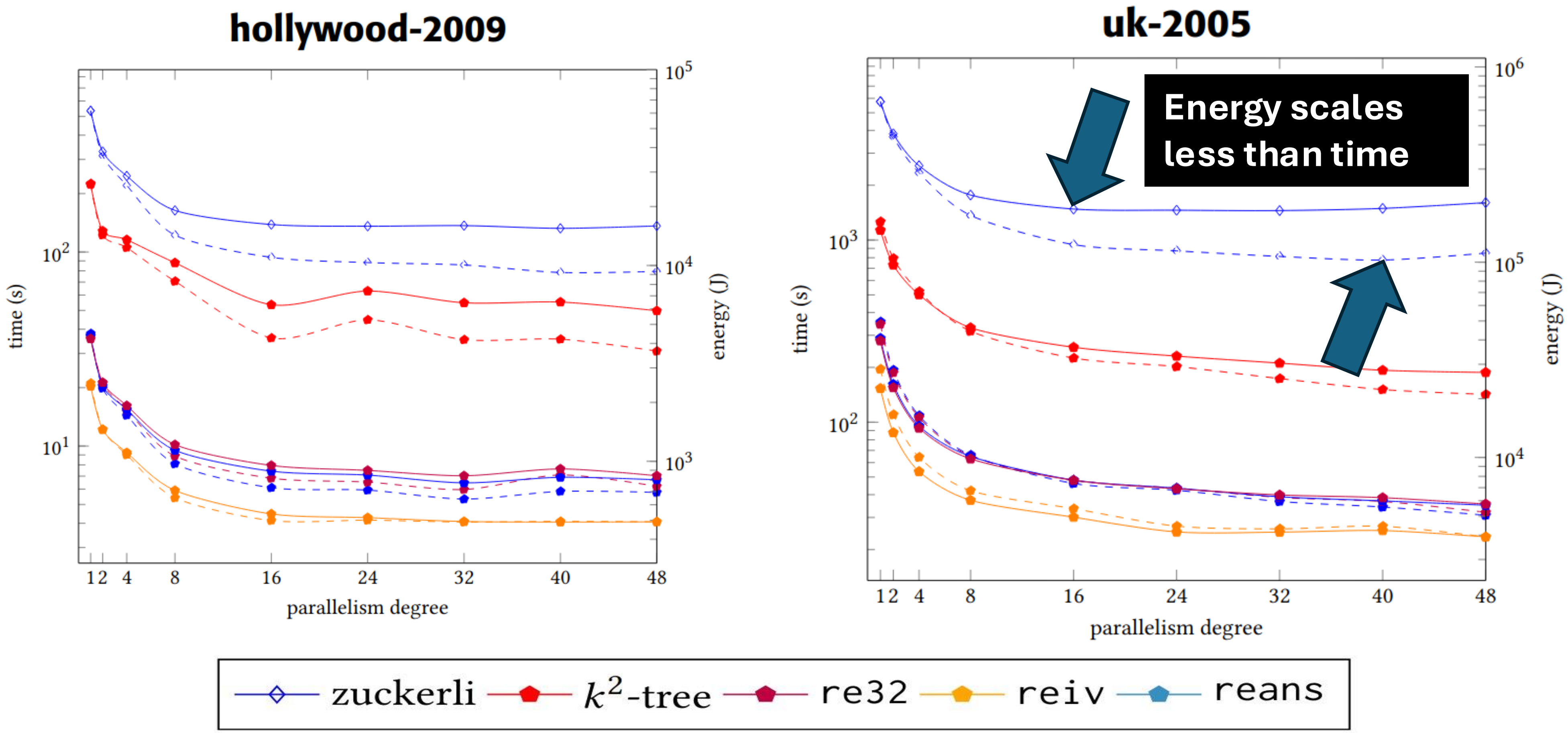
The memory usage surpasses the disk usage due to

- Additional temporary data structures
- Block splitting for multithread multiplication

Space-time performances
on the Intel® Xeon® Server

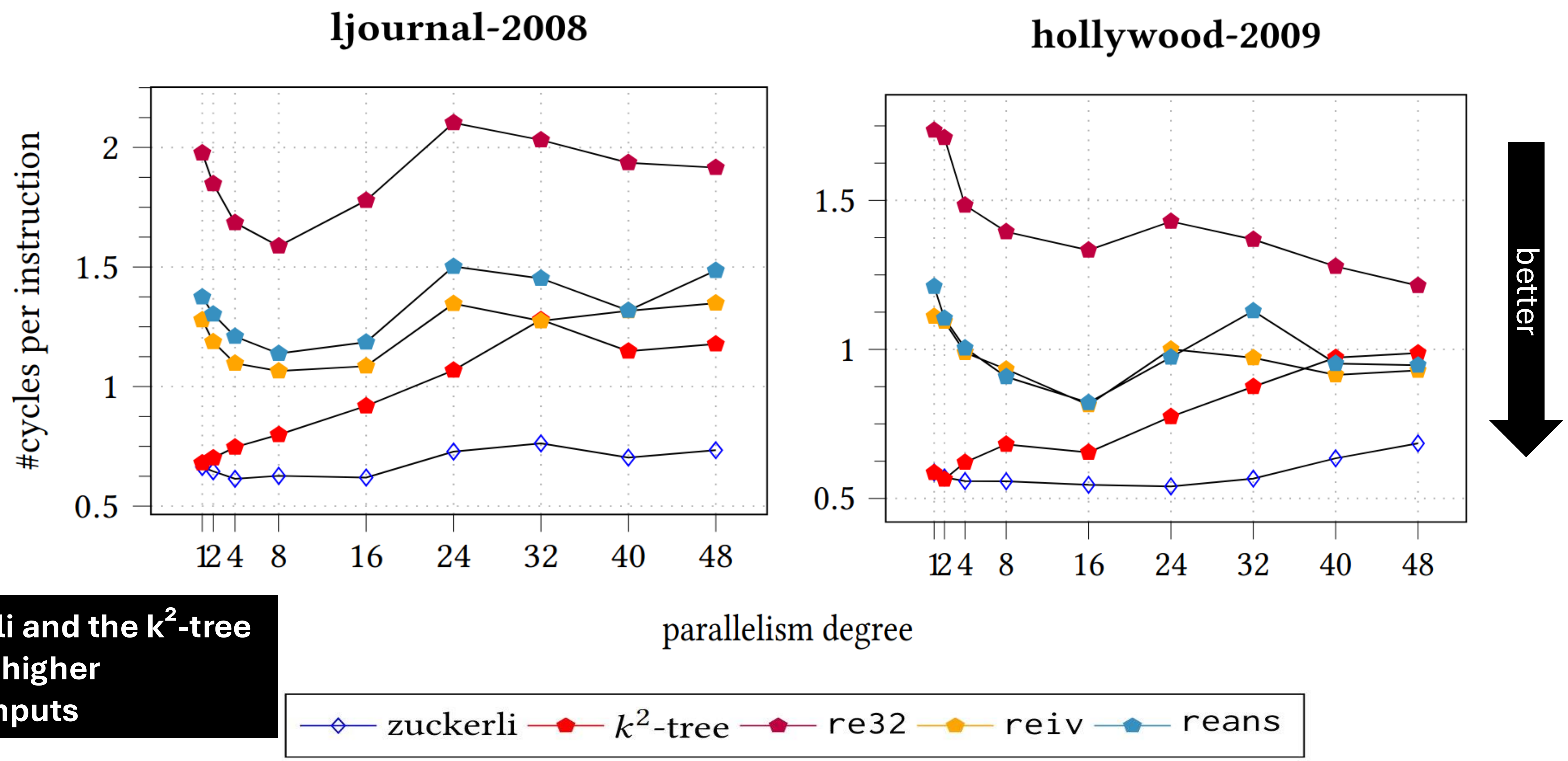


Time (_ _ _) and energy (___) performances
on the Intel® Xeon® Server



Multicriteria approach in multi-threaded applications
(e.g., select fastest solution within energy constraints)

Instruction throughput
on the Intel® Xeon® Server

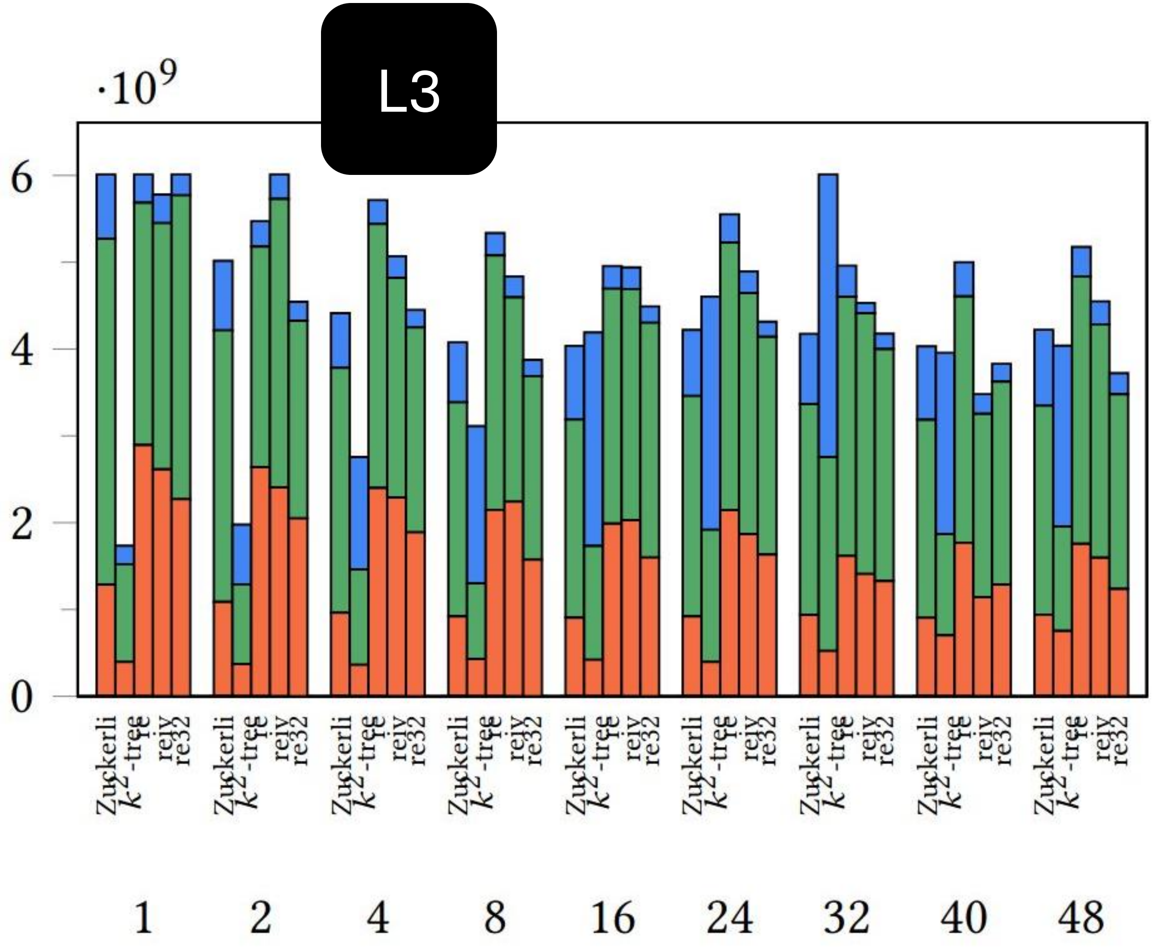


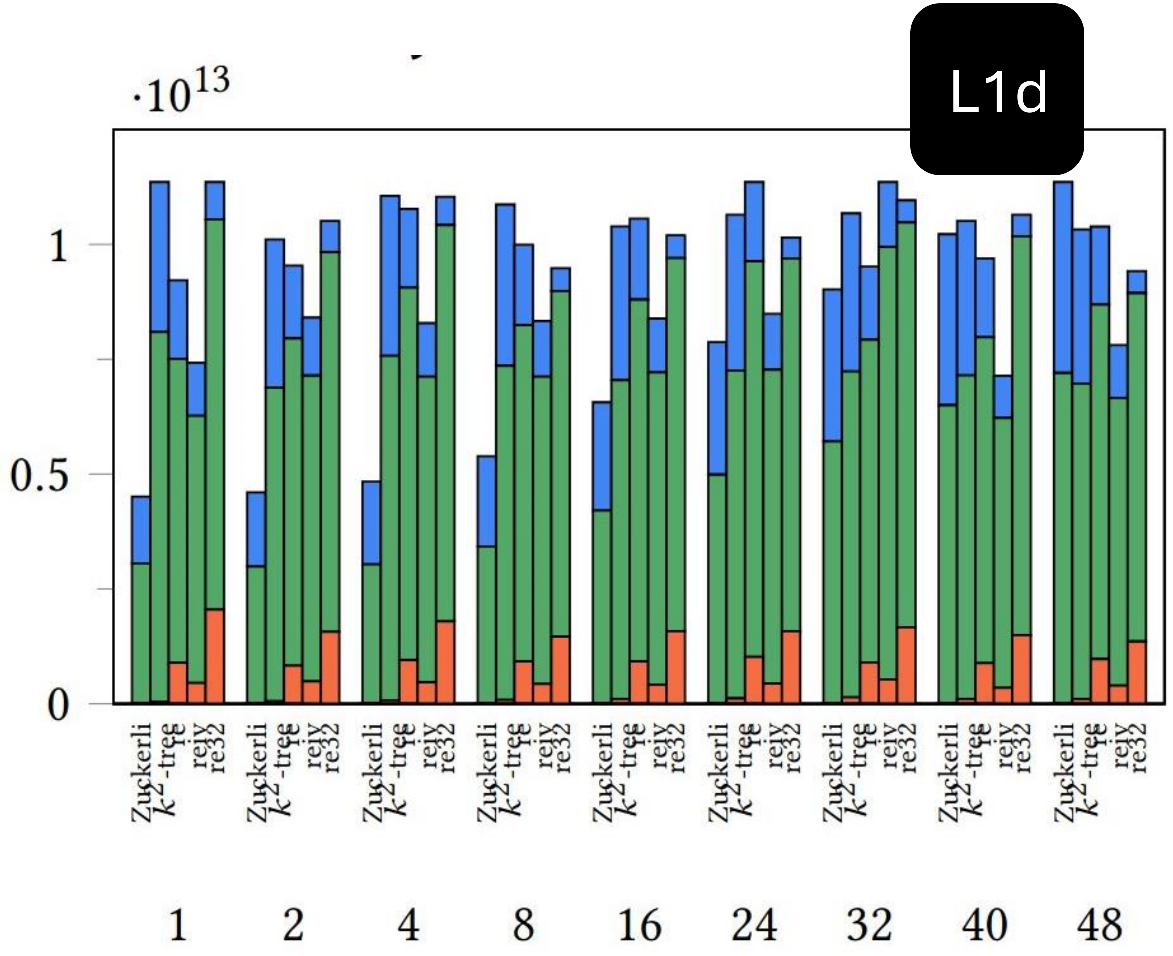
Zuckerli and the k²-tree exhibit higher throughputs

L1d/L3 cache access patterns

on the Intel® Xeon® Server
Dataset: ljournal-2008

Data cache access patterns: load cache misses, load cache hits, and store operations.

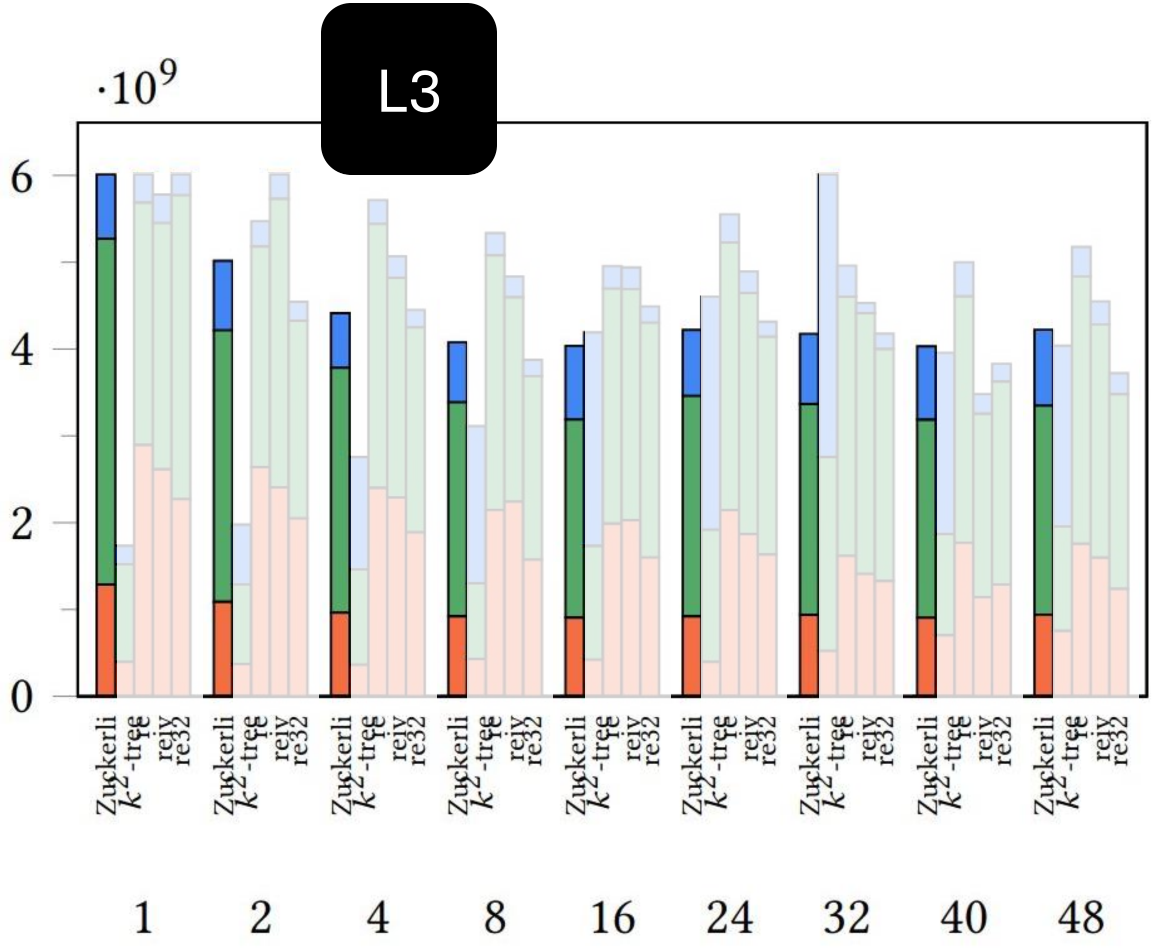


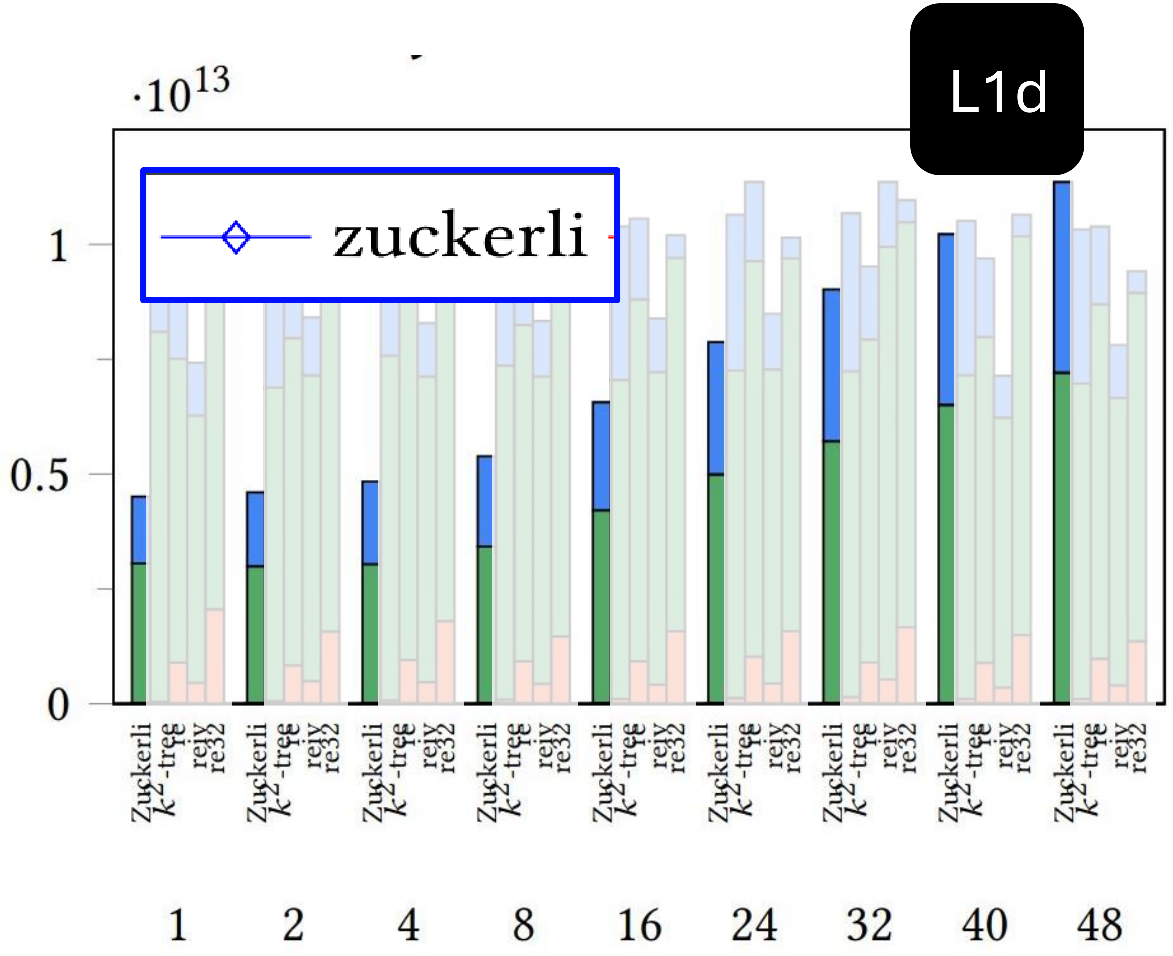


L1d/L3 cache access patterns

on the Intel® Xeon® Server
Dataset: ljournal-2008

Data cache access patterns: load cache misses, load cache hits, and store operations.

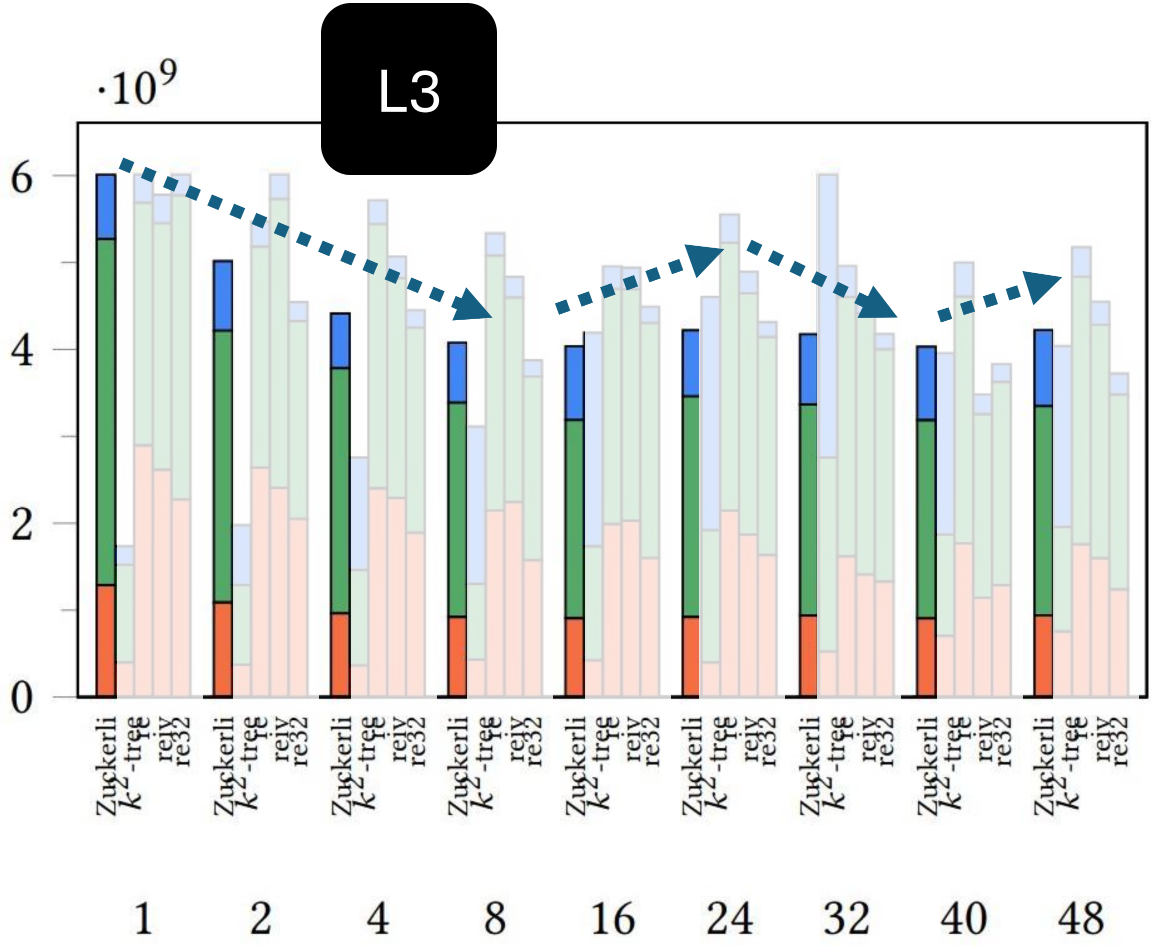


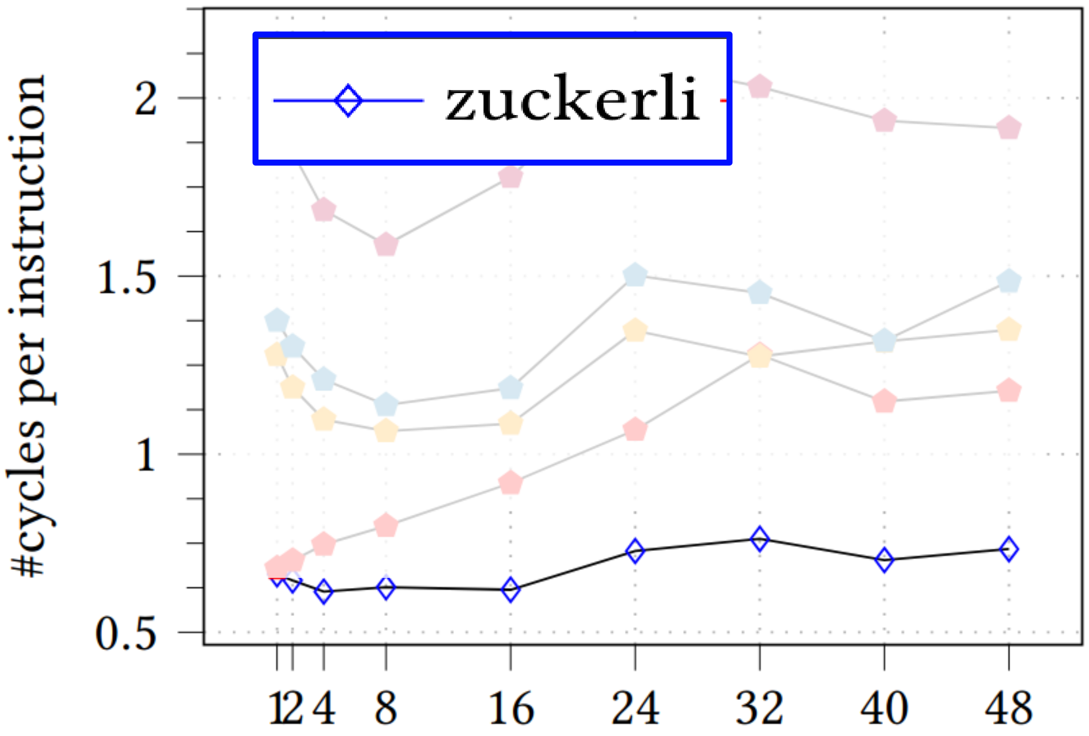


L1d/L3 cache access patterns

on the Intel® Xeon® Server
Dataset: ljournal-2008

Data cache access patterns: load cache misses, load cache hits, and store operations.

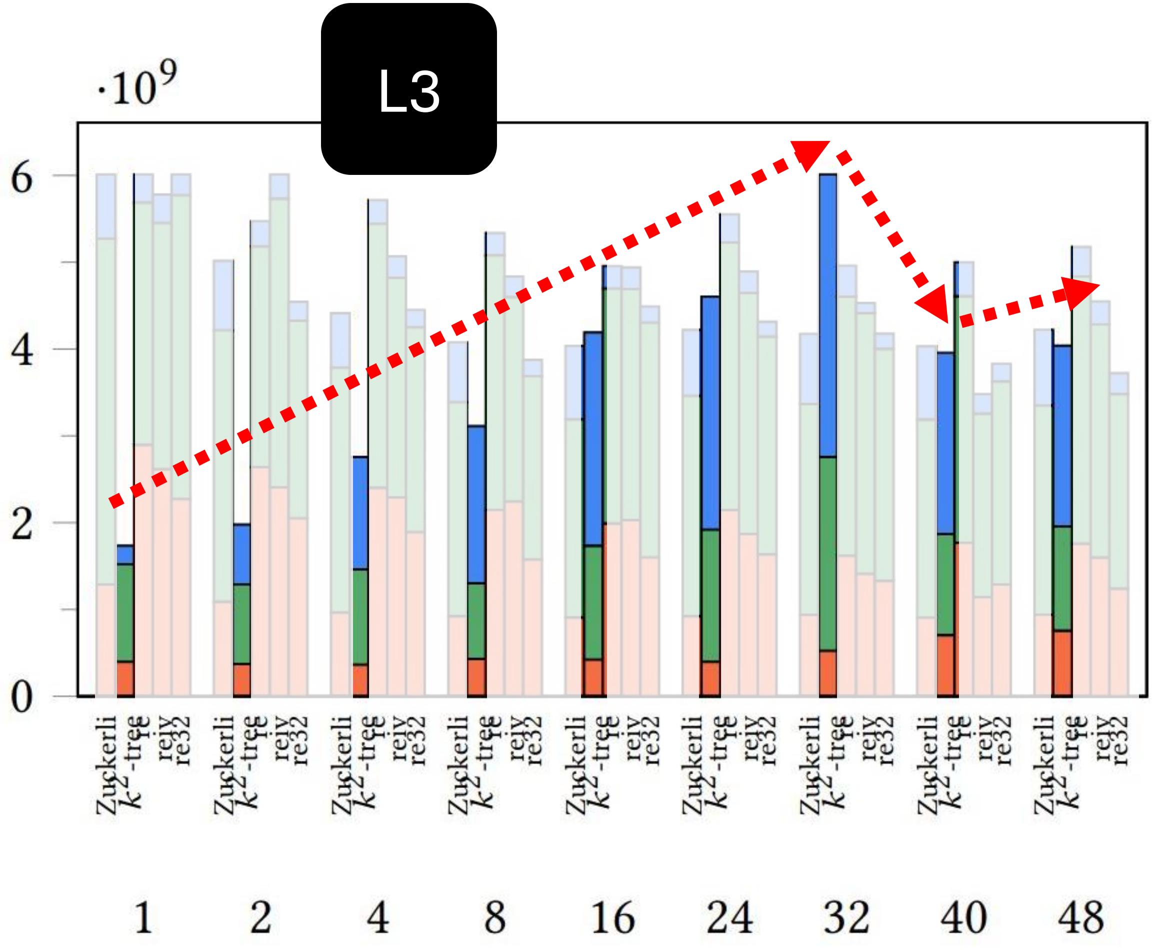


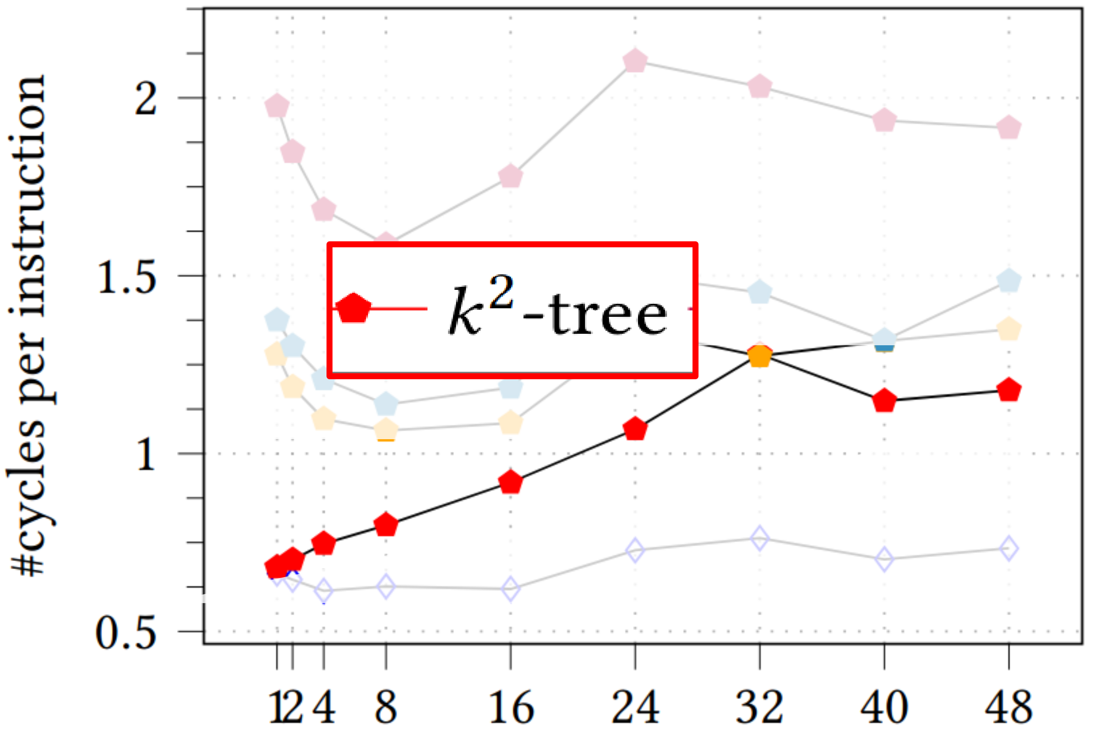


L1d/L3 cache access patterns

on the Intel® Xeon® Server
Dataset: ljournal-2008

Data cache access patterns: load cache misses, load cache hits, and store operations.

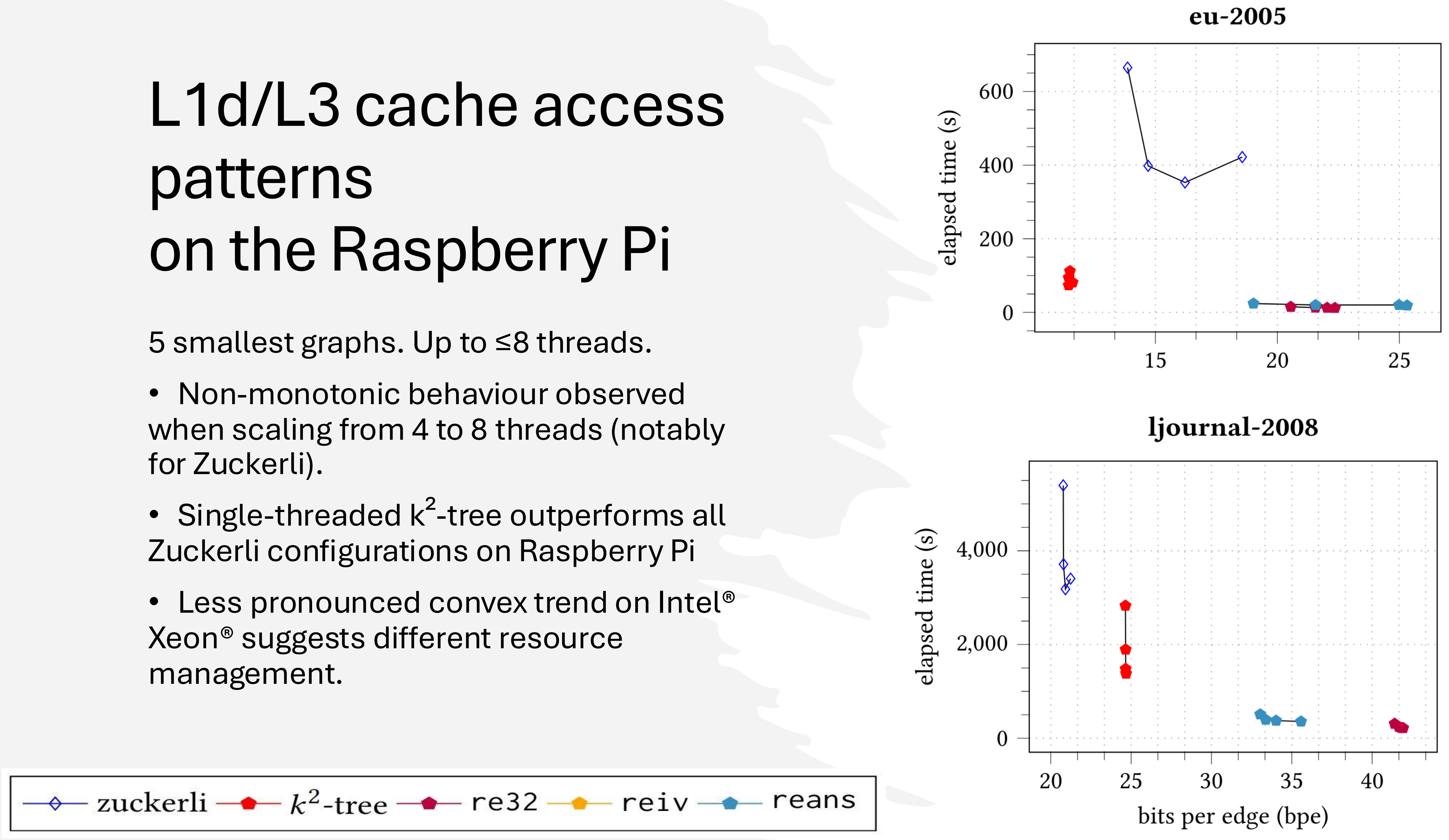




L1d/L3 cache access patterns on the Raspberry Pi

5 smallest graphs. Up to ≤8 threads.

- Non-monotonic behaviour observed when scaling from 4 to 8 threads (notably for Zuckerli).
- Single-threaded k^2-tree outperforms all Zuckerli configurations on Raspberry Pi
- Less pronounced convex trend on Intel® Xeon® suggests different resource management.



Conclusion

- Appropriate compressed representations enable handling large datasets on resource-constrained devices.
- Careful selection of compressed representation can reduce energy usage by one or two orders of magnitude across different environments.
- The k^2-tree is nearly as fast as grammar-based compressors and almost as space-efficient as Zuckerli, with minimal degradation in compression ratio as threads increase.
- Optimal thread counts may differ for minimising energy vs. optimizing runtime.
- Increased L1 and L3 cache operations affect cycles per instruction

Future work

- Investigate additional lossless compression formats for matrices and vectors.
- Expand applications beyond PageRank.
- In-depth studies on optimising energy-time tradeoffs. → Insights for software engineers to reduce carbon footprints and enhance battery life.
- Study energy-efficient implementations of major compressed data structures (e.g., FM-index, Rank and select structures, Suffix arrays, Succinct tree topologies, …)
- ML optimisation: explore combinations of lossless and state-of-the-art lossy compression strategies.

Transparency and reproducibility

The entire codebase to reproduce the experiments is made available at gitlab: https://gitlab.com/ftosoni/green-lossless-spmv

Datasets from https://sparse.tamu.edu/LAW

GitLab

Dr. Francesco Tosoni

Prof. Giovanni Manzini

Mr. Valerio Brunacci

Prof. Alessio De Angelis

Prof. Paolo Ferragina

Prof. Philip Bille

Coauthors

- 4 universities
- 3 research groups
- 2 research areas

Research groups

Visit our pages to stay up-to-date with our research outcomes!

A3 Lab

A.D. 1308
unipg
DIPARTIMENTO
DI INGEGNERIA
DIPARTIMENTO DI ECCELLENZA
MUR 2023/2027

DTU

Acube

EMERLab

AlgoLoG

Questions & Answers

Francesco Tosoni, PhD
Postdoctoral researcher

Informatica
L.go B. Pontecorvo 3
56127 Pisa PI
Italy

francesco.tosoni@di.unipi.it
pages.di.unipi.it/tosoni