Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 545–551

ISBN 978-83-60810-51-4

Adaptive-Blocking Hierarchical Storage Format for Sparse Matrices

Daniel Langr, Ivan Šimeček, Pavel Tvrdík
Czech Technical University in Prague
Faculty of Information Technology
Thákurova 9, 160 00, Praha, Czech Republic
Email: langrd@fit.cvut.cz

Tomáš Dytrych, Jerry P. Draayer
Louisiana State University
Department of Physics and Astronomy
Baton Rouge, LA 70803, USA

***Abstract*—Hierarchical storage formats (HSFs) can significantly reduce the space complexity of sparse matrices. They vary in storage schemes that they use for blocks and for block matrices. However, the current HSFs prescribe a fixed storage scheme for all blocks, which is not always space-optimal. We show that, generally, different storage schemes are space-optimal for different blocks. We further propose a new HSF that is based on this approach and compare its space complexity with current HSFs for benchmark matrices arising from different application areas.**

I. Introduction

PARTITIONING an $m \times n$ sparse matrix into $s \times s$ blocks, we can represent it as a two-level hierarchical data structure (see Fig. 1). Level 0 consists of nonzero *blocks*, which are, in fact, submatrices of size $s \times s$ with at least one nonzero element. Level 1 forms a *block matrix* of size $\lceil m/s \rceil \times \lceil n/s \rceil$, whose elements are pointers to particular blocks. Storage formats for sparse matrices based on such partitioning are called *hierarchical storage formats*[1] (HSFs).

Blocks as well as a block matrix can be either sparse or dense and we can use various *storage schemes* to store them in a computer memory. Let us mention the most frequently used ones:

- The **dense** storage scheme consists of a single array that stores all elements in particular order.
- The **bitmap** storage scheme consists of an array of nonzero elements and a bitmap that stores their pattern.
- The **coordinate (COO)** storage scheme consists of three arrays that contain nonzero elements, their row indexes, and their column indexes [4], [5].
- The **compressed sparse row (CSR)** storage scheme consists of three arrays. Two arrays contain nonzero elements and their column indexes, respectively. The third array contains pointers for all rows to their data in the first two arrays. The last member of the third array contains the number of nonzero elements [4], [5].

This work was supported by the Czech Science Foundation under Grant No. P202/12/2011, by the Grant Agency of the Czech Technical University in Prague under grant SGS12/097/OHK3/1T/18, by the U.S. National Science Foundation under Grant No. OCI-0904874, and by the U.S. Department of Energy under Grant No. DOE-0904874.

[1] *Recursive storage formats*, which are based on recursive top-down partitioning of a matrix into smaller blocks [1]–[3], are sometimes also denoted as *hierarchical*. These types of formats are not discussed within this paper.

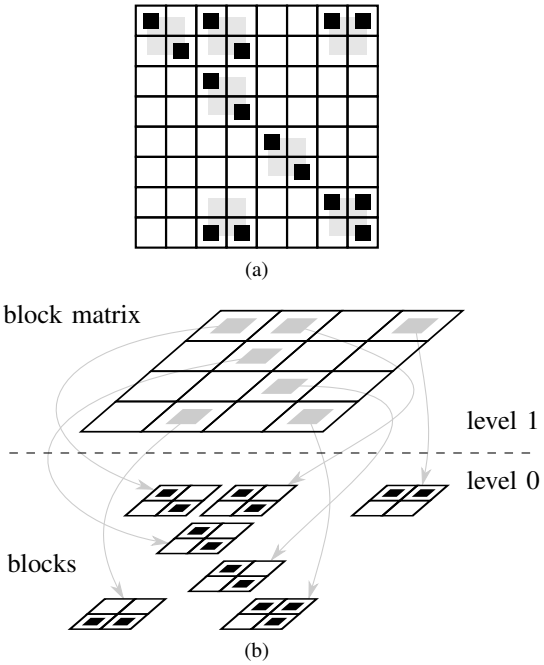
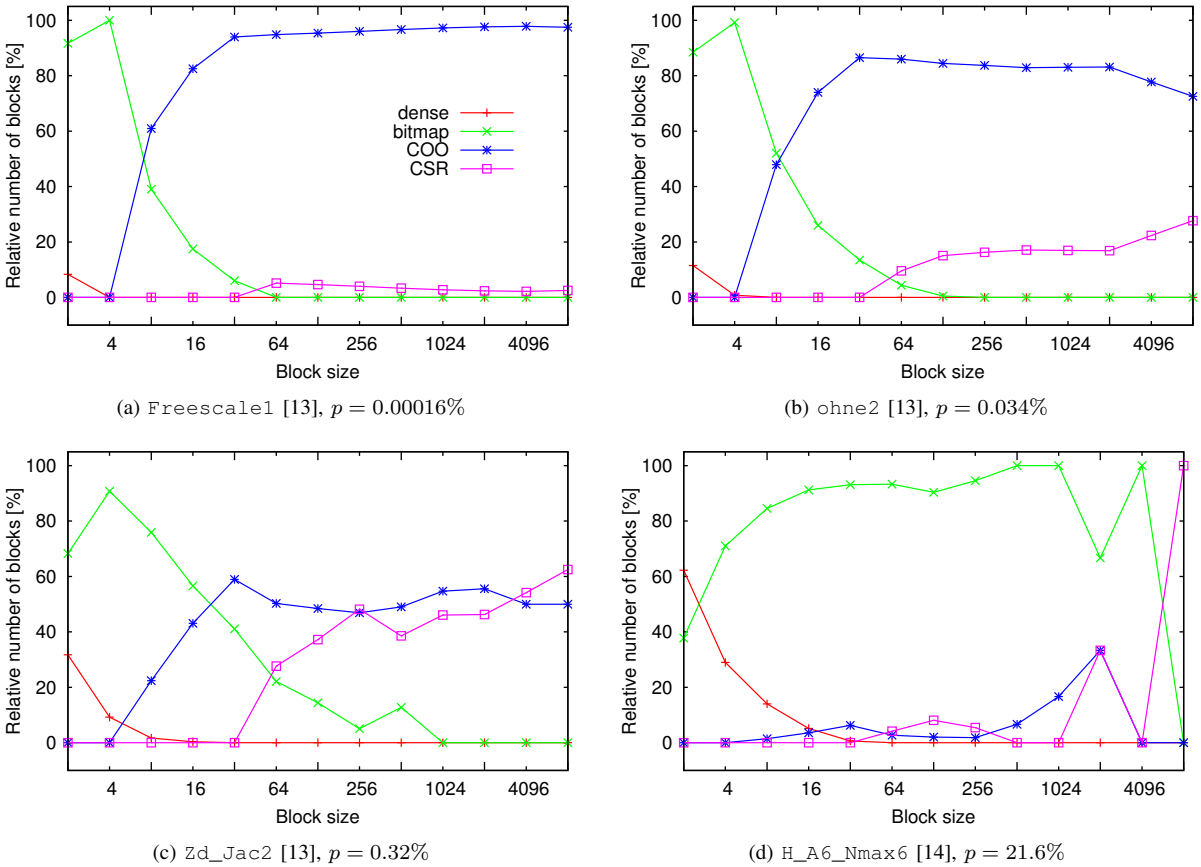


Fig. 1: An 8×8 matrix (a) represented as a hierarchical data structure with 2×2 blocks (b).

In Table I, we present a summary of the previously developed HSFs, which we further call *current HSFs*. Each prescribes a fixed scheme for all blocks, which has the following drawbacks:

1) Generally, different HSFs are space-optimal for different matrices (see [6] or Table III). In other words, none of the current HSFs is space-optimal for all matrices.
2) Within a single matrix, different storage schemes are generally space-optimal for different blocks, as illustrated by Fig. 2. Consequently, some blocks might be stored in a non-optimal way if only a single storage scheme is used.

In this paper, we present a new HSF called *Adaptive-blocking hierarchical storage format* (ABHSF) that is based on the idea of using a space-optimal storage scheme for each individual block. We further compare the space complexities of ABHSF with the current HSFs for benchmark matrices arising



(a) `Freescale1` [13], $p = 0.00016\%$

(b) `ohne2` [13], $p = 0.034\%$

(c) `Zd_Jac2` [13], $p = 0.32\%$

(d) `H_A6_Nmax6` [14], $p = 21.6\%$

Fig. 2: The relative number of nonzero blocks, for which is the given storage scheme space optimal, as a function of the block size s. Measurements are presented for a subset of benchmark matrices, each identified by its name, source, and the number of nonzero elements p.

HSF	Source	Level 1	Level 0
BCRS BCSR	[5], [7] [8]	CSR	dense
HiSM COOCOO	[7], [9] [6]	COO	COO
SBSRS SPBCRSX CSRCOO	[10] [11] [6]	CSR	COO
CSB	[12]	dense	COO
COOCSR	[6]	COO	CSR
CSRCSR	[6]	CSR	CSR
AHF	[6]	bitmap	COO

TABLE I: Current HSFs and their storage schemes.

from different application areas and discuss the results.

II. Analysis

Let A be an $m \times n$ sparse matrix partitioned into blocks of size $s \times s$. Let further:

- z be the number of nonzero elements of A;
- $p = z/(m \cdot n) \cdot 100$ be the relative number of nonzero elements of A in percents;
- $M = \lceil m/s \rceil$ be the number of rows of the block matrix;
- $N = \lceil n/s \rceil$ be the number of columns of the block matrix;
- Z be the number of nonzero blocks, i.e., blocks that contain at least one nonzero element;
- d be the number of bytes required for storing a single matrix element (for example, $d = 4$ in case of single-precision real numbers, $d = 8$ in case of double-precision real numbers or single-precision complex numbers, etc.);
- $\mathcal{B}(\xi)$ be the minimum number of bytes of an unsigned integer data type that can index an array of ξ elements, i.e.,

$$\mathcal{B}(\xi) = \min\{\eta \in \{8, 16, 32, 64\} : \xi \leq 2^{\eta}\} / 8,$$

if zero-based indexing is used.

A. Space complexity of HSFs

The *space complexity* of some data structure is defined as a minimum number of bytes needed to store this data structure in a computer memory. Let $\mathcal{S}_X$ denote the space complexity of A that is stored in HSF X. For HSFs, we consider the following assumptions:

1) Pointers are implemented as zero-based array indexes.

2) For indexes of an array with ξ elements, the minimum data type is used, i.e., an unsigned integer data type of size $\mathcal{B}(\xi)$ bytes[2].
3) All indexes are stored as local to the particular level. For example, rows within blocks are indexed from 0 to $s-1$ and rows within the block matrix are indexed from 0 to $M-1$.
4) Arrays at level 0 are accessed sequentially. Since we know the index of the first element as well as the number of elements that belong to each block in a particular array, we do not need to store explicitly any pointers to that array at level 1.

Providing the following definitions:

$$\begin{aligned}\mathrm{COO}_0 &= z \cdot \left[2 \cdot \mathcal{B}(s) + d\right],\\ \mathrm{CSR}_0 &= z \cdot \left[\mathcal{B}(s) + d\right] + Z \cdot (s+1) \cdot \mathcal{B}\left(s^2\right),\\ \mathrm{COO}_1 &= Z \cdot \left[\mathcal{B}(M) + \mathcal{B}(N)\right],\\ \mathrm{CSR}_1 &= Z \cdot \mathcal{B}(N) + (M+1) \cdot \mathcal{B}(Z),\end{aligned}$$

we can express the space complexity of A stored in the current HSFs as follows[3]:

$$\begin{aligned}\mathcal{S}_{\mathrm{BCSR}} &= \mathrm{CSR}_1 + Z \cdot s^2 \cdot d,\\ \mathcal{S}_{\mathrm{HiSM}} &= \mathrm{COO}_1 + Z \cdot \mathcal{B}\left(s^2\right) + \mathrm{COO}_0,\\ \mathcal{S}_{\mathrm{SBSRS}} &= \mathrm{CSR}_1 + Z \cdot \mathcal{B}\left(s^2\right) + \mathrm{COO}_0,\\ \mathcal{S}_{\mathrm{CSB}} &= M \cdot N \cdot \mathcal{B}\left(s^2+1\right) + \mathrm{COO}_0,\\ \mathcal{S}_{\mathrm{COOCSR}} &= \mathrm{COO}_1 + \mathrm{CSR}_0,\\ \mathcal{S}_{\mathrm{CSRCSR}} &= \mathrm{CSR}_1 + \mathrm{CSR}_0,\\ \mathcal{S}_{\mathrm{AHF}} &= \lceil M \cdot N \,/\, 8 \rceil + Z \cdot \mathcal{B}\left(s^2\right) + \mathrm{COO}_0.\end{aligned}$$

B. Space-optimal storage schemes for blocks

Let ζ denote a number of nonzero elements of a block B. Let σ_X denote the space complexity of B that is stored in the storage scheme X. For the previously considered storage schemes, we can express the space complexities as follows:

$$\sigma_{\mathrm{dense}} = s^2 \cdot d, \tag{1a}$$

$$\sigma_{\mathrm{bitmap}} = \lceil s^2/8 \rceil + \zeta \cdot d, \tag{1b}$$

$$\sigma_{\mathrm{COO}} = \zeta \cdot \left[2 \cdot \mathcal{B}(s) + d\right], \tag{1c}$$

$$\sigma_{\mathrm{CSR}} = \zeta \cdot \left[\mathcal{B}(s) + d\right] + (s+1) \cdot \mathcal{B}\left(s^2\right). \tag{1d}$$

By evaluating and comparing these expressions, we can find their minimum and hence the corresponding space-optimal storage scheme for B.

[2]Accurately, we need $\lceil \log_2 \xi \rceil$ bits to index an array of size ξ. However, most of nowadays computer systems and programming languages naturally work only with integer numbers of width 8, 16, 32, or 64 bits.

[3]Minimizing the storage requirements, we should calculate CSR_0 as

$$\mathrm{CSR}_0 = z \cdot \left[\mathcal{B}(s) + d\right] + \sum_{k=1}^{Z} (s+1) \cdot \mathcal{B}(\zeta_k),$$

where ζ_k is the number of nonzero elements of the k-th nonzero block. However, the implementation of such a variant of the CSR storage scheme would be very impractical, since we would need, generally, different integer data types for different blocks to store pointers to data for all rows. Therefore, we consider the variant of the CSR storage scheme, where all pointers can be stored in a single array. The same approach is used in (1d).

For demonstration, examples of space complexities (1) are shown in Fig. 3.

III. Adaptive-Blocking Hierarchical Storage Format

Here, we propose a new HSF called *Adaptive-Blocking Hierarchical Storage Format* (ABHSF) that is based on the following concept:

1) The space-optimal storage scheme from the set {dense, bitmap, COO, CSR}, that is, the storage scheme with the minimum space complexity (1), is used for every block.
2) For the block matrix, the COO storage scheme is used.

The evaluation of the space complexity of ABHSF for the whole matrix is not as trivial as for the current HSFs. For every block at level 1, we need:

1) row index inside the block matrix—$\mathcal{B}(M)$ bytes,
2) column index inside the block matrix—$\mathcal{B}(N)$ bytes,
3) (optimal) storage scheme—coded in 1 byte,
4) number of nonzero elements if the storage scheme is COO[4]—$\mathcal{B}\left(s^2\right)$ bytes[5].

Let Z_X be the number of blocks for which the storage scheme X is space-optimal. Let z_X be the total number of nonzero elements that belong to these blocks. The space complexity for level 1 is then

$$Z \cdot \left[\mathcal{B}(M) + \mathcal{B}(N) + 1\right] + Z_{\mathrm{COO}} \cdot \mathcal{B}\left(s^2\right). \tag{2}$$

For level 0, the space complexity can be evaluated according to the storage schemes that are space-optimal for particular blocks, as follows:

- **Dense**—for every block, we store all its elements (including zeros). Overall, we need

$$Z_{\mathrm{dense}} \cdot s^2 \cdot d \tag{3}$$

bytes.
- **Bitmap**—for every block, we store a bitmap of size $\lceil s^2/8 \rceil$ bytes along with nonzero elements for these blocks, which translates into

$$Z_{\mathrm{bitmap}} \cdot \lceil s^2/8 \rceil + z_{\mathrm{bitmap}} \cdot d \tag{4}$$

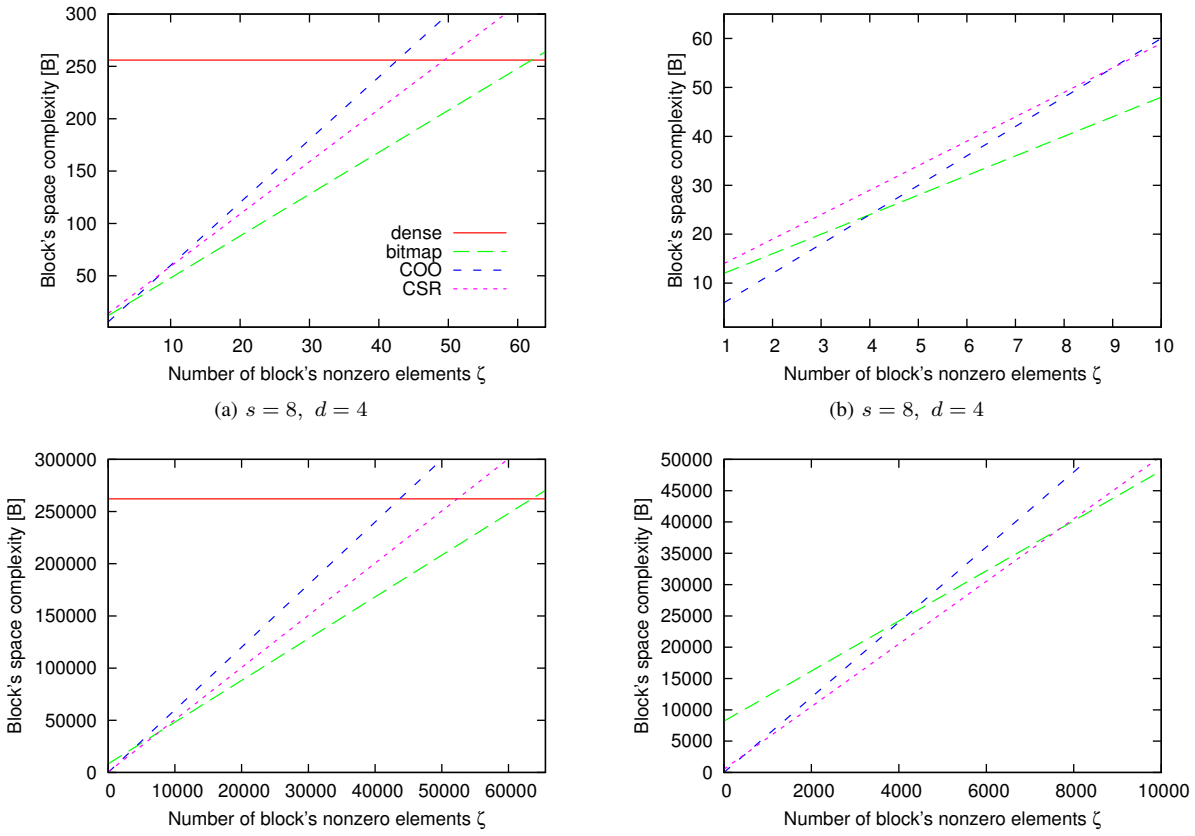
bytes.
- **COO**—we store nonzero elements along with their row and column indexes local to a block. As a result, we need

$$z_{\mathrm{COO}} \cdot \left[2 \cdot \mathcal{B}(s) + d\right] \tag{5}$$

bytes.
- **CSR**—for every block, we store the first element for each row, totally $(s+1)$ indexes (local to a block).

[4]For other storage schemes, one can evaluate the number of block (nonzero) elements in another way.

[5]The number of nonzero elements for nonzero blocks ranges between 1 and s^2, thus, accurately, we need $\mathcal{B}\left(s^2+1\right)$ bytes to store this information. However, when the block contains s^2 nonzero elements, it would never be stored in the COO storage scheme, since the dense storage scheme is space-optimal in such a case.



(a) $s = 8,\ d = 4$ (b) $s = 8,\ d = 4$

(c) $s = 256,\ d = 4$ (d) $s = 256,\ d = 4$

Fig. 3: The space complexity of a single block for different storage schemes, as a function of the number of its nonzero elements.

In addition, we store nonzero elements along with their column indexes local to a block. Overall, we need

$$Z_{\mathrm{CSR}} \cdot (s+1) \cdot \mathcal{B}\left(s^2\right) + z_{\mathrm{CSR}} \cdot \left[\mathcal{B}\left(s\right) + d\right] \quad (6)$$

bytes.

Combining the expressions (2), (3), (4), (6), and (5), we finally get the total space complexity of ABHSF as follows:

$$\begin{aligned}\mathcal{S}_{\mathrm{ABHSF}} = {} & Z \cdot \left[\mathcal{B}\left(M\right) + \mathcal{B}\left(N\right) + 1\right] + Z_{\mathrm{COO}} \cdot \mathcal{B}\left(s^2\right) \\ & + Z_{\mathrm{dense}} \cdot s^2 \cdot d \\ & + Z_{\mathrm{bitmap}} \cdot \lceil s^2/8 \rceil + z_{\mathrm{bitmap}} \cdot d \\ & + z_{\mathrm{COO}} \cdot \left[2 \cdot \mathcal{B}\left(s\right) + d\right] \\ & + Z_{\mathrm{CSR}} \cdot (s+1) \cdot \mathcal{B}\left(s^2\right) + z_{\mathrm{CSR}} \cdot \left[\mathcal{B}\left(s\right) + d\right].\end{aligned}$$

IV. Experiments

To evaluate ABHSF, we performed measurements of space complexities for the current HSFs as well as for ABHSF with a set of matrices arising in different application areas. We call these matrices *benchmark* and their list, along with their origin and properties, is presented in Table II.

We first measured the space complexities $\mathcal{S}$ for all discussed HSFs as a function of the block size s. Due to the space limitations, we present results for a subset of benchmark matrices in Fig. 4. These results were obtained for $s = 2^k$, where $k = 1, \ldots, 13$.

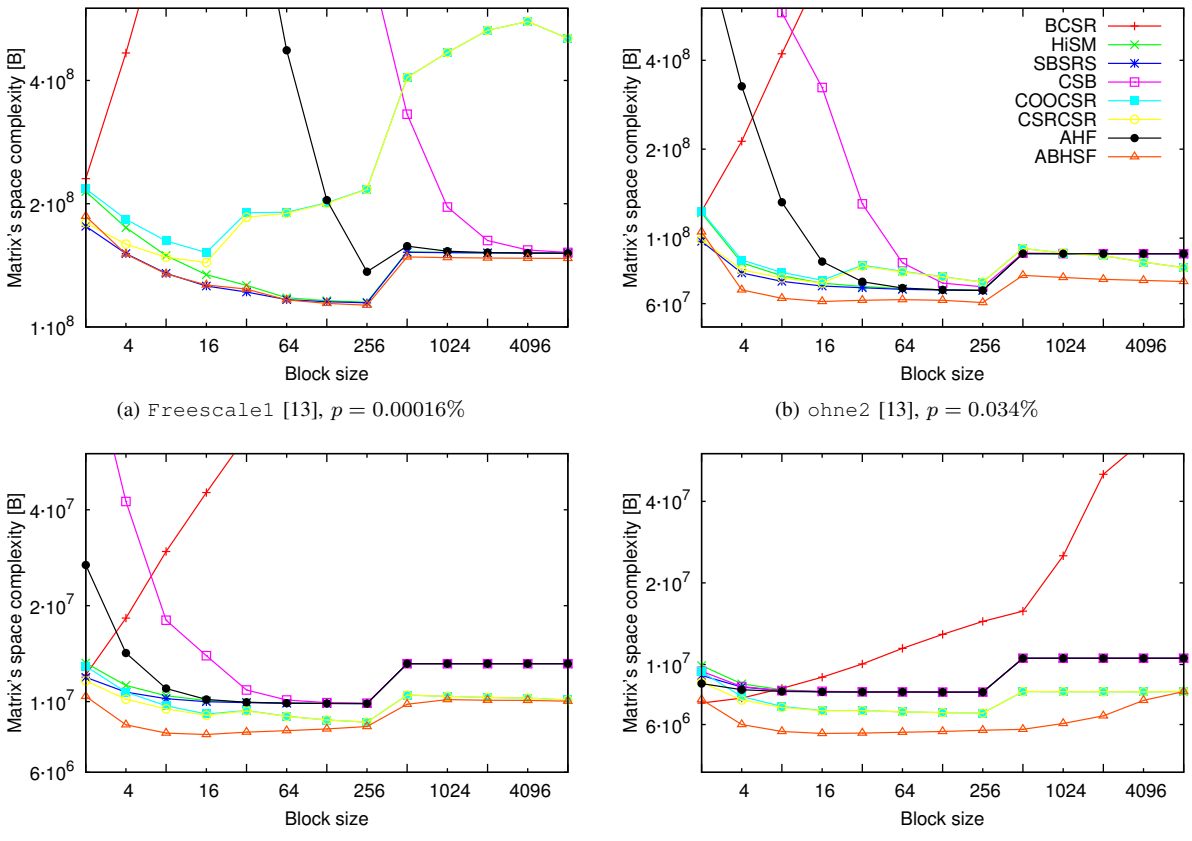
Next, we compared the space complexities $\mathcal{S}$ of the current HSFs with ABHSF for all benchmark matrices—the results are presented in Table III. Because of the limited space, we present results for $s = 256$ only. According to Fig. 4, this value, in most cases, seems to provide the best space complexity. Since we wanted to evaluate ABHSF, we present the relative values of $\mathcal{S}$, where $\mathcal{S}_{\mathrm{ABHSF}}$ corresponds to 100%.

By inspecting Fig. 4 and Table III, we can make the following observations:

- None of the current HSFs works well—in the sense of space optimality—for all matrices. For instance, only COOCSR and CSRCSR yield results close to the minimum for matrices such as `Zd_Jac2`. Looking at Fig. 2(c), we can see that there exist a lot of blocks for which the CSR storage scheme is space-optimal. Hence, only COOCSR and CSRCSR, which are the only formats that use CSR at level 0, provide good results in this particular case.
  On the other hand, these formats provide space complexities far from the minimum values for matrices such as `Freescale1`. The reason is that, according to Fig. 2(a),

Matrix	Source	Domain	m, n	p	Symmetric
amazon0312	[13]	directed graph	$4.0 \cdot 10^5$	$2.0 \cdot 10^{-3}$	no
atmosmodj	[13]	computational fluid dynamics	$1.3 \cdot 10^6$	$5.5 \cdot 10^{-4}$	no
bmw7st_1	[13]	structural problem	$1.4 \cdot 10^5$	$1.8 \cdot 10^{-2}$	yes
c-29	[13]	optimization problem	$5.0 \cdot 10^3$	$9.6 \cdot 10^{-2}$	yes
cage12	[13]	directed weighted graph	$1.3 \cdot 10^5$	$1.2 \cdot 10^{-2}$	no
FEM_3D_thermal2	[13]	thermal problem	$1.5 \cdot 10^5$	$1.6 \cdot 10^{-2}$	no
Freescale1	[13]	circuit simulation	$3.4 \cdot 10^6$	$1.6 \cdot 10^{-4}$	no
gupta2	[13]	optimization problem	$6.2 \cdot 10^4$	$5.6 \cdot 10^{-2}$	yes
H_A6_Nmax6	[14]	nuclear structure	$2.5 \cdot 10^3$	$2.2 \cdot 10^1$	yes
H_A6_Nmax6_all	[14]	nuclear structure	$5.3 \cdot 10^4$	2.9	yes
ldoor	[13]	structural problem	$9.5 \cdot 10^5$	$2.6 \cdot 10^{-3}$	yes
mouse_gene	[13]	undirected weighted graph	$4.5 \cdot 10^4$	$7.1 \cdot 10^{-1}$	yes
nlpkkt120	[13]	optimization problem	$3.5 \cdot 10^6$	$4.0 \cdot 10^{-4}$	yes
ohne2	[13]	semiconductor device	$1.8 \cdot 10^5$	$3.3 \cdot 10^{-2}$	no
thread	[13]	structural problem	$3.0 \cdot 10^4$	$2.5 \cdot 10^{-1}$	yes
Trel_6Li_Nmax6	[14]	nuclear structure	$2.0 \cdot 10^5$	$1.0 \cdot 10^{-2}$	yes
Zd_Jac2	[13]	chemical process	$2.3 \cdot 10^4$	$3.2 \cdot 10^{-1}$	no

TABLE II: The list of the benchmark matrices used for the performed experiments.



(a) `Freescale1` [13], $p = 0.00016\%$

(b) `ohne2` [13], $p = 0.034\%$

(c) `Zd_Jac2` [13], $p = 0.32\%$

(d) `H_A6_Nmax6` [14], $p = 21.6\%$

Fig. 4: The space complexity of the whole matrix for different HSFs and $d = 4$, as a function of the block size s. Measurements are presented for a subset of benchmark matrices, each identified by its name, source and the number of nonzero elements.

Matrix	BCSR	HiSM	SBSRS	CSB	COOCSR	CSRCSR	AHF	ABHSF
amazon0312	879938.0	96.7	90.0	115.6	1802.5	1795.9	***84.5***	100.0
atmosmodj	17739.4	104.5	*104.4*	297.9	121.8	121.7	110.3	**100.0**
bmw7st_1	8571.4	110.7	110.7	116.6	109.1	*109.0*	110.8	**100.0**
c-29	12555.7	104.8	*104.7*	105.7	111.9	111.8	*104.7*	**100.0**
cage12	24212.3	103.2	*103.0*	111.4	133.4	133.2	103.1	**100.0**
FEM_3D_thermal2	6483.5	112.6	112.5	119.6	106.5	*106.4*	112.7	**100.0**
Freescale1	54890.6	101.7	*101.4*	735.6	192.2	191.8	120.7	**100.0**
gupta2	16356.3	*106.8*	*106.8*	108.5	121.0	121.0	*106.8*	**100.0**
H_A6_Nmax6	252.2	138.4	138.6	138.4	*115.8*	*115.8*	138.4	**100.0**
H_A6_Nmax6_all	969.3	118.7	118.7	118.7	*100.8*	*100.8*	118.7	**100.0**
ldoor	21106.4	105.2	*105.0*	145.4	129.0	128.8	106.1	**100.0**
mouse_gene	5196.9	109.6	109.6	109.7	*101.5*	*101.5*	109.6	**100.0**
nlpkkt120	6950.6	108.0	108.0	382.2	*103.6*	*103.6*	116.5	**100.0**
ohne2	13077.1	110.1	*110.0*	113.1	117.3	117.2	110.0	**100.0**
thread	2978.0	115.0	115.0	115.4	*101.7*	*101.7*	115.0	**100.0**
Trel_6Li_Nmax6	28937.7	100.3	*100.1*	109.7	140.2	140.0	100.2	**100.0**
Zd_Jac2	2477.2	118.1	118.1	118.4	103.3	*103.2*	118.1	**100.0**

TABLE III: The relative space complexity of benchmark matrices in percents for $s = 256$, $d = 4$, and different HSFs, related to ABHSF. The best value out of all HSFs is emphasized with **bold print**, the best value achieved for the current HFSs is emphasized with *italics*.

the COO storage scheme is space-optimal for a vast majority of blocks within this matrix. Even more extreme example of unsuitability of the COOCSR and CSRCSR formats is the matrix `amazon0312`.

- ABHSF clearly provides the best results among all discussed HSFs and hence seems to be the most universal format as to the space optimality. It provides the best space complexity for all matrices except for `amazon0312`. Even for this matrix, ABHSF gives relatively good result, 18.3% above the minimum value of AHF. (For comparison, the worst values for the current HSFs are at least 38.4% above the minimum across matrices.)
  The reason why ABHSF does not work well for the `amazon0312` matrix is the following. This format introduces some memory overhead at level 1 (see Section III). This overhead consists of
  1) the code of the actual storage scheme for each block;
  2) the number of block nonzero elements if its actual storage scheme is COO.

  The `amazon0312` matrix has almost all blocks extremely sparse: $\zeta = 1$ for the majority of blocks and $\zeta \leq 10$ for the others. This is illustrated by the frequency diagram in Fig. 5 (for comparison, we also present the same frequency diagram for the `Zd_Jac2` matrix). According to (1) and provided that $s = 256$, $\sigma_{\mathrm{COO}} < \sigma_{\mathrm{CSR}}$ for $\zeta < 514$. Therefore, all blocks within the `amazon0312` matrix will be stored in the COO storage scheme. And since the amount of data stored at level 0 for each block is very small (especially if $\zeta = 1$), the memory overhead of ABHSF becomes considerable here.
- For most HSFs, there is a significant step increase in the space-complexity values between block sizes $s = 256$ and $s = 512$. It is caused by the required data type for local indexes within blocks, since

$$\mathcal{B}(s) = \begin{cases} 1 & \text{for } s \leq 256, \\ 2 & \text{for } 256 < s \leq 8129. \end{cases}$$

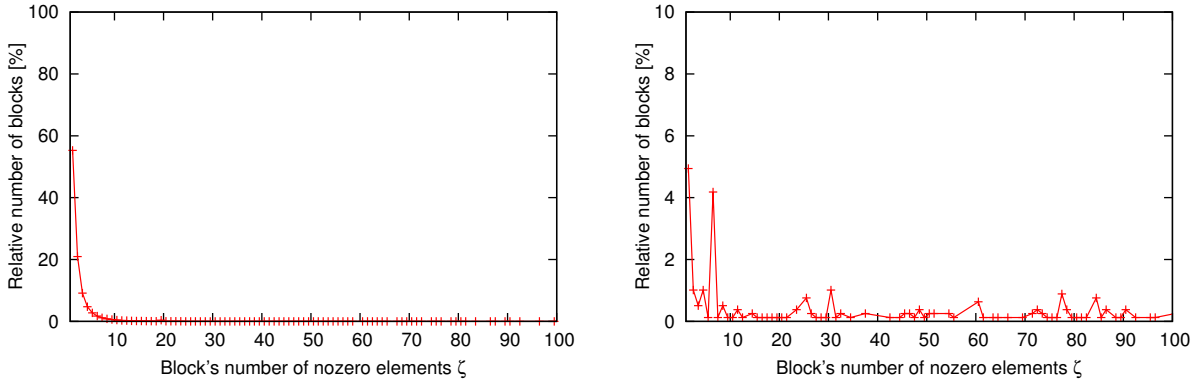
  It is worth noting that this step change is not present for ABHSF and the `H_A6_Nmax6` matrix—see Fig. 4(d). The reason is that the bitmap storage scheme, which does not use any local indexes, is space-optimal for the majority of blocks; see Fig. 2(d). (No HSF other than ABHSF uses the bitmap storage scheme at level 0.)

V. Conclusions

In this paper, we aimed at optimization of the space complexity of hierarchical storage formats (HSFs) for sparse matrices. Our contribution is the new format called Adaptive-Blocking Hierarchical Storage Format (ABHSF). We tried to answer the question if it is worth to store each individual block in its space-optimal storage scheme. Based on the performed experiments, we can conclude that the answer is yes—ABHSF is clearly the most universal compared to the current HSFs (see Table I) when minimization of the space complexity of sparse matrices is considered.

In our experiments, we found only one matrix for which ABHSF did not provide the best result—a matrix with extremely sparse blocks, namely blocks containing mostly just 1 nonzero element. ABHSF introduces some overhead into level 1 that is significant is such a case.

We did not focus on the particular implementation of ABHSF, that is, we did not show detailed organization of data structures (arrays) used at both levels. Our aim was to show rather the concept of this new format and to compare it with the current HSFs assuming their space-optimal implementations based on rules that are listed in Section II-A.



(a) `amazon0312` (b) `Zd_Jac2`

Fig. 5: The frequency diagrams of the relative number of blocks according to their number of nonzero elements ζ for selected benchmark matrices. For comparison, the x-axis is limited by $\zeta \leq 100$.

We also limited the storage schemes for individual blocks to dense, bitmap, COO, and CSR. Other storage schemes may be considered as well, e.g., the ones based on bitmap compression or the quad-tree data structures [3].

Data structures and optimization of their space complexities were the subject of our work. We thus did not present any algorithm related to ABHSF. These algorithms are dependent on a particular situation, that is, they depend on what we particularly want to do with this format—for instance, we can convert other formats from/to ABHSF, generate matrix elements (in particular order) directly into ABHSF, perform sparse matrix-vector multiplication, etc.

Since we do not focus on algorithms related to ABHSF, we do not discuss their time complexity and efficiency. It can be readily seen, however, that ABHSF brings some overhead to the data structures for sparse matrices and this overhead would lead to higher complexity of related algorithms. Thus, this format would be beneficial particularly in situations, where one attempts to minimize the space-complexity of sparse matrices at any price. Our motivation for this research is the parallel I/O for extremely large sparse matrices that are processed by massively parallel computer systems with distributed memory architectures. On such systems, the number of I/O nodes is usually of several orders of magnitude lower than the number of computational cores and storing/loading of large data into/from a distributed file system may form a significant performance hotspot. In such situations, it is imperative to minimize the space complexity of data structures and ABHSF provides this possibility specially for sparse matrices.

REFERENCES

[1] Michele Martone, Salvatore Filippone, Marcin Paprzycki, and Salvatore Tucci. On the usage of 16 bit indices in recursively stored sparse matrices. *Symbolic and Numeric Algorithms for Scientific Computing*, 0:57–64, 2010.

[2] Michele Martone, Salvatore Filippone, Paweł Gepner, Marcin Paprzycki, and Salvatore Tucci. Use of hybrid recursive CSR/COO data structures in sparse matrices-vector multiplication. In *Proceedings of the International Multiconference on Computer Science and Information Technology*, Wisla, Poland, October 2010.

[3] Ivan Šimeček and Pavel Tvrdík. Sparse matrix-vector multiplication - final solution? In *Proceedings of the 7th International Conference on Parallel Processing and Applied Mathematics*, PPAM'07, pages 156–165, Berlin, Heidelberg, 2008. Springer-Verlag.

[4] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[5] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, PA, 2nd edition, 1994.

[6] Ivan Šimeček, Daniel Langr, and Pavel Tvrdík. Space-efficient sparse matrix storage formats for massively parallel systems. In *Proceedings of the 14th IEEE International Conference of High Performance Computing and Communications*, pages 54–60. IEEE Computer Society, 2012.

[7] Pyrros Theofanis Stathis. *Sparse Matrix Vector Processing Formats*. PhD thesis, Technische Universiteit Delft, 2004.

[8] Richard Vuduc, James W. Demmel, Katherine A. Yelick, Shoaib Kamil, Rajesh Nishtala, and Benjamin Lee. Performance optimizations and bounds for sparse matrix-vector multiply. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, Supercomputing '02, pages 1–35, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[9] Pyrrhos Stathis, Stamatis Vassiliadis, and Sorin Cotofana. A hierarchical sparse matrix storage format for vector processors. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, IPDPS '03, page 61, Washington, DC, USA, 2003. IEEE Computer Society.

[10] Rukhsana Shahnaz and Anila Usman. Blocked-based sparse matrix-vector multiplication on distributed memory parallel computers. *The International Arab Journal of Information Technology*, 8(2):130–136, 2011.

[11] F. S. Smailbegovic, G. N. Gaydadjiev, and S. Vassiliadis. Sparse Matrix Storage Format. In *Proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Processing, ProRisc 2005*, pages 445–448, 2005.

[12] Aydin Buluç, Jeremy T. Fineman, Matteo Frigo, John R. Gilbert, and Charles E. Leiserson. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, SPAA '09, pages 233–244, New York, NY, USA, 2009. ACM.

[13] T. A. Davis and Y. F. Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1), 2011.

[14] J. P. Draayer, T. Dytrych, K. D. Launey, and D. Langr. Symmetry-adapted ab initio theory for many-body correlations in nuclei. *Journal of Physics: Conference Series*, 321(1):012040, 2011.