České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Czech Technical University in Prague
Faculty of Information Technology
Department of Computer Sysytems

Ing. I. Šimeček, Ph.D.

Paměťově úsporné formáty uložení velkých řídkých matic

Space-efficient Formats for Storing Large Sparse Matrices

Summary

Large sparse matrices are widely used in high-performance computing. This work deals with new storage formats and algorithms for sparse matrices. Our search for new algorithms and formats was motivated by absence of an available solution (at the beginning of the research). There was no satisfactory solution for storing/loading large sparse matrices to/from a distributed file system.

This work presents some solutions for this application domain. Space-efficient formats reduce the space complexity of the representation of the sparse matrix. These formats can be applied in situations when a capacity or a bandwidth is the main bottleneck. New formats and algorithms (usually with possibilities for their parallel execution) are presented. The theoretical part of this work contains a formal description of sparse matrix formats and a survey of state-of-the-art solutions. The work also discusses principles, algorithms for transformation from/to these formats, and suitability for practical application.

Souhrn

Velké řídké matice jsou používány v mnoha oborech intenzivních výpočtů. Tato práce pojednává o nových formátech a algoritmech pro řídké matice. Absence těchto formátů a algoritmů (na začátku výzkumu) byla naší hlavní motivací. Neexistovalo totiž uspokojivé řešení pro ukládání nebo načítání velké řídké matice v rámci distribuovaného souborového systému.

Tato práce přináší některá možná řešení pro tuto aplikační doménu. Paměťově úsporné formáty jsou takové, které se snaží minimalizovat množství paměti nutné pro uložení velkých řídkých matic. Jejich využití leží v oblastech, kde je omezena dostupnost nebo propustnost úložných zařízení. Jsou zde popsány nové formáty a algoritmy (včetně možností jejich paralelizace). Teoretická část práce obsahuje formální popis formátů a přehled současných řessení. V práci budou diskutovány principy těchto formátů, algoritmy pro převod z/do těchto formátů a jejich výhody či nevýhody při praktickém uplatnění.

Klíčová slova

Řídká matice, paměťová složitost, násobení řídké matice vektorem, paralelní I/O, distrubovaná paměť, vícevláknové zpracování, intenzivní výpočty, MPI knihovna, OpenMP, cache paměť, formát používající kvadrantový strom.

Keywords

Sparse matrix format, space complexity, sparse matrix-vector multiplication, parallel I/O, distributed memory, multithreaded execution, high performance computing, MPI, OpenMP, cache memory, quadtree format.

Contents

1 Introduction

1.1 Motivation

The maximal memory bandwidth and parallel I/O subsystem can be performance bottlenecks in sparse computations, e.g., loading or storing large sparse matrices from/to a distributed file system can take significant amounts of time. Sparse storage formats (SSFs) describe a way how sparse matrices are stored in computer memory. In this thesis, new formats are shown to minimize the indexing overhead and consequently reduce main memory traffic or the parallel I/O traffic.

The space complexity (sometime called memory footprint) of representation of sparse matrices depends strongly on the used matrix storage format.Within our papers [13, 14, 11, 10, 12], weaknesses of previously developed solutions for space-efficient formats for storing of large sparse matrices were discussed. This thesis investigates memory-efficient storage formats for large sparse matrices (LSMs). These matrices that due to their sizes must be stored and processed by massively parallel computer systems (MPCSs) with distributed memory architecture consisting of processor cores. The motivation of our work was in applications with LSMs that must be stored in a distributed file system using a parallel I/O subsystem. The parallel I/O subsystem can be performance bottleneck and loading or storing such matrices from/to a distributed file system are costly operations. We reduced this time by reducing the space complexity of the LSMs.

1.2 Terminology and notation

1.2.1 General assumption and notation

- We consider a large sparse matrix A of order $n \times n$, $A = (a_{i,j})$. The number of its nonzero elements is denoted by N.
- Matrix A is considered **sparse** if it is worth (for performance or any other reason) not to store this matrix in memory in a dense array.
- We assume that $1 \ll n \leq N \ll M = n^2$.
- The pattern of nonzero elements in A is unknown or random.
- The number of nonzero elements in submatrix B of matrix A is denoted by $\eta(B)$, thus $\eta(A) = N$.

1.2.2 Matrix properties

- A matrix A is **regular** if $det(A) \neq 0$, and it is **singular** otherwise.
- A matrix A is **symmetric** if $A = A^T$.
- A matrix A is **diagonal** if $\forall i \neq j;\ a_{i,j} = 0$.
- If A has the same number of nonzero elements in each row then nonzero elements are distributed in A uniformly and we denote A as a **uniform matrix**.

2 Space-efficient formats

2.1 Our assumptions

Our research addresses computations with LSMs satisfying at least one of the following conditions:

1. The LSM is used repeatedly and the computation of its elements is slow and it takes more time than its later reading from a file system.
2. Construction of a LSM is memory-intensive. It needs significant amount of memory for auxiliary data structures, typically of the same order of magnitude as the amount of memory required for storing the LSM itself.
3. A solver requires the LSM in another format than is produced by a matrix generator and the conversion between these formats cannot be performed effectively on-the-fly.
4. Computational tasks with LSMs need check-pointing and recovery from failures of the MPCSs. We assume that a distributed-memory parallel computation with a LSM needs longer time. To avoid recomputations in case of a system failure, we need to save a state of these long-run processes to allow fast recovery. This is especially important nowadays (and will be more in the future) when MPCSs consist of tens or hundreds of thousands of processor cores.

If at least one of these conditions is met, we might need to store LSMs into a file system. And since the file system access is usually of orders of magnitude slower compared to the memory access, we want to store matrices in a way that minimizes their memory requirements.

2.2 Our requirements

The requirements for a new storage format are as follows:

1. One of MPCS's bottleneck lies in parallel I/O bandwidth. Therefore we require that the new format should be space-efficient, in order to keep resulting file sizes as low as possible.
2. We want to access LSMs files linearly. Since nowadays I/O file operations are processed by hard discs, linear access with minimal amount of seek operations leads to a maximal efficiency of reading.
3. For the designed format, there must also exist a space-efficient algorithm with small algorithmic complexity for remapping from/to the common storage formats. Due to this assumption, the time complexity of remapping is negligible in comparison to I/O file operations complexity.

Unfortunately, it is hard to satisfy all requirements at the same time, because they are generally in contradiction. This work is inspired by some real applications, for example ab initio calculations of medium-mass atomic nuclei.

There are several other storage formats specialized for given areas (e.g., compression of text, picture or video). They can be used for compression of sparse matrices, but none of them satisfies all these four requirements:

1. non-lossy compression,
2. possibility of massively parallel execution,
3. space efficiency (high compression rate),
4. high speed compression/decompression.

In this thesis, the compression of the information describing the *structure* of LSMs (i.e., the locations of nonzero elements) is discussed. The values of the nonzero elements are unchanged, because their compression depends strongly on the application. For some application areas, the values of nonzero elements are implicit and only the information about the structure of a LSM is stored (for example, incident matrices of unweighed graphs).

3 State-of-art

In this section short survey of state-of-art sparse storage formats is given.

3.1 Common sparse storage formats

SSFs describe a way how sparse matrices are stored in a computer memory. The following three SSFs are most common for storing sparse matrices.

3.1.1 The Coordinate (COO) Format

The coordinate (COO) format is the simplest SSF (see [19, 2]). The matrix A is represented by three linear arrays *values*, *xpos*, and *ypos*. The array $values[0, \ldots, N-1]$ stores the nonzero values of A, arrays $xpos[0, \ldots, N-1]$ and $ypos[0, \ldots, N-1]$ contain column and row indexes, respectively, of these nonzero values. COO does not prescribe any order of these arrays.

3.1.2 The Compressed Sparse Row (CSR) format

The most common SSF is the *compressed sparse row* (CSR) format (see [19, 2]). The matrix A stored in the CSR format is represented by three linear arrays *values*, *addr*, and *ci*. The array $values[0, \ldots, N-1]$ stores the nonzero elements of A, the array [1] $addr[0, \ldots, n-1]$ contains indexes of initial nonzero elements of rows of A; the first nonzero element of the row j is stored at index $addr[j]$ in array *values*. The row i contains $addr[i+1] - addr[i]$ elements. If row i does not contain any nonzero element, then $addr[i] = addr[i+1]$ and matrix A is singular. Hence, all elements of the array *addr* should satisfy the condition $\ldots addr[i-1] \leq addr[i] \leq addr[i+1] \ldots$. The array $ci[1, \ldots, \eta]$ contains column indexes of nonzero elements of A.

3.1.3 Register blocking formats

Widely-used SSFs are easy to understand, however, sparse operations (like matrix-vector or matrix-matrix multiplication) using these formats are slow (mainly due to indirect addressing). Sparse matrices often contain dense submatrices (blocks), so various blocking SSFs were designed to accelerate matrix operations. Compared to the CSR format, the aim of these formats (like SPARSITY[7]) is to allow a better use of registers and more efficient computations. But these specialized SSFs have usually large transformation overhead and consume approximately the same amount of memory as the CSR format.

[1] Usually the array *addr* is by one element larger ($[0, \ldots, n]$), this simplifies many algorithms.

3.2 State-of-the-art survey

What were the possibilities to manage storing/loading LSMs to/from a distributed file system (before our research)? There were several widely used and well documented text-based file formats for sparse matrices, mainly Matrix Market [3], Harwell-Boeing [4], and Matlab (ASCII) [6]. There are, however, reasons why text-based storage formats are not suitable for VLSMs— they must be accessed sequentially and they usually consume much more space than binary formats.

As for binary file formats, there were no satisfactory solutions. Many modern sparse solvers, such as Trilinos [5] or PETSc [1], provide the functionality of storing matrices into a file. However,

1. the matrices must already be loaded into the solver, whereas we might need to store matrices as they are constructed;

2. the binary formats of such files are usually proprietary and poorly or not at all documented, and therefore they cannot be simply used anywhere else.

Just few papers have been published about SSFs in the context of minimization of the required memory (before our research), which is the optimization criterion for a file I/O. Some recent research of hierarchical blocking SSFs, though primarily aimed at SpMV optimization, also addresses optimization of memory requirements [16, 15, 17, 18].

4 Our new space-efficient formats

In this section, new sparse matrix storage formats that minimize the space complexity of information about matrix structure are proposed and evaluated.

4.1 The entropy-based (EB) format

4.1.1 The main idea

The space complexity of any sparse matrix storage format depends strongly on its structural pattern. If the sparsity pattern of a matrix is completely known (for example, if a matrix is tridiagonal) then the space complexity for storing the information on its structure is zero. If a random distribution of nonzero elements is assumed, then it is equal to the value of the entropy of a bit vector of size M, in which

N bits are set to 1 and $M - N$ bits are set to 0. Thus, such a format is denoted as *entropy-based.* Unfortunately, the EB format is very difficult to compute, thus it serves only for comparison and no practical algorithm to achieve this space complexity was given. In [12], the arithmetical-coding-based (ACB) format was introduced. Since a random distribution of nonzero elements is assumed, the bit vector B is considered to be an order-0 source (each bit is selected independently on other bits). The total number of bits to encode vector B is equal to the value of binary entropy of vector B, thus EBF and ACB formats have the same space complexity.

4.1.2 Results and applicability

A comparison to common SSF was done in [8, 11, 12]. This format is suitable for matrices without any locality. A drawback of the ACB format is its computational complexity. Since each bit of vector B is encoded in time $\Theta(1)$, the complete vector B (representation of sparse matrix A) is encoded in time $\Theta(n^2)$. This is too much for sparse matrices with a constant number of nonzero elements per row (i.e., $N \in \Theta(n)$).

4.2 Minimal quadtree (MQT) format

4.2.1 The main idea

Definition 1. *The* Quadtree *(QT) is a tree data structure in which all inner nodes have exactly four child nodes.*

A big drawback of the some QT formats from the viewpoint of space complexity is a larger data overhead (caused by pointers *up_left*, *up_right*, *lo_left*, *lo_right*) compared to the COO and CSR formats. Since our aim is to minimize the space complexity of QT-based formats, in [10] a new QT format called *minimal quadtree (MQT) format* is proposed that extends ideas of the standard QT format as follows:

- All nodes in the MQT are stored in one array. Since we can compute locations of all child nodes, we can omit pointers. We lose the advantage of the possibility to easily modify the QT, but it is not an important property for our application area.
- Instead of pointers, each node of the MQT contains only 4 flags (i.e., 4 bits only) indicating whether given subquadtrees are nonempty.

So, the space complexity of every MQT node is only 4 bits.

4.2.2 Results and applicability

A space complexity comparison to common SSF was done in [10]. The derivation of lower and upper bounds for this format was also included. Experiments proved that this format minimize space complexity of the sparse matrix structure.

4.3 The minimal binary tree (MBT) format

4.3.1 The main idea

The *full binary tree* (FBT) is a widely used data structure in which all inner nodes have exactly two child nodes. Binary trees especially those used for binary space partitioning can also be used for storing sparse matrices. In standard implementations, every node in a FBT is represented by structure `standard_BT_struct` consisting of the following items:

- two pointers (*left*, *right*) to child nodes,
- (only for leaves) the value of a nonzero element.

If a FBT is used as a basis for SSF, it describes a partition of the sparse matrix into submatrices and each node in the FBT represents a submatrix. Equally as in k-d trees, the decomposition is performed in alternating directions: first horizontally, then vertically, and so on. From the viewpoint of space efficiency, a drawback of the standard FBT representation is the overhead caused by pointers *left*, *right*. To eliminate this drawback, we propose a new k-d-tree-based SSF. Each tree node represents again a submatrix, but we modify the standard representation of the FBT and we call this data structure the *minimal binary tree* (MBT) format. The idea is very similar to that in the MQT format.

- All nodes of a MBT are stored in one array (or stream). Since the size of the input matrix is given, we can compute locations of all child nodes, we can omit pointers *left*, *right*.
- All nodes of a MBT contain only two flags (it means only two bits). Each of them is set to 1 if the corresponding half of the submatrix (left/right or upper/lower) contains at least one nonzero element, otherwise it is set to 0.

So, the space complexity of every MBT node is only 2 bits.

4.3.2 Results and applicability

A space complexity comparison to common SSF was done in [12]. The derivation of lower and upper bounds for this format was also included.

Space complexities using this format are comparable to MQT format (see Section 4.2).

4.4 Minimal compressed formats

4.4.1 The main idea

The space complexity of MBT and MQT formats (see Section 4.2 and 4.3) can be further reduced by compression as was discussed in [12]. The MBT and MQT formats have minimal space complexity only if we assume fixed number of bits for each node (2 bits for MBT and 4 bits for MQT). We can relax this assumption to achieve more space efficient formats. Based on this idea, we propose another new format, called *compressed binary tree (CBT).* and another new *compressed quadtree (CQT)* format.

4.4.2 Results and applicability

A space complexity comparison to common SSF is done in [12]. It also includes the derivation of lower and upper bounds for this format.

4.5 COOCOO256 and COOCSR256 formats

4.5.1 The main idea

The high memory requirements for the COO format are caused by two arrays of size N. We cannot reduce the size of these arrays, but we can try to reduce the number of bytes for every row/column index. The idea is to partition the matrix into square *blocks* of size $r \times c$ rows/columns. In [8], these parameters were fixed ($c = r = 256$), thus these formats were denoted as COOCOO256 or COOCSR256. Every such block can be identified by *block row* and *block column* indices of size $S(\lceil n/256 \rceil)$ bytes. Let K denotes the number of nonzero blocks for our matrix A (nonzero block is a block that contains at least one nonzero value). Suppose nonzero matrix elements stored in the COO format. If we store nonzero blocks in the coordinate storage format, we need for each one its block row/column index of size $S(\lceil n/256 \rceil)$ bytes, and a pointer into its data (an index into the original arrays of row/column indices and values) that it therefore an index of size $S(N)$ bytes. Now, for each nonzero element, we need only 1-byte local row/column indices valid within a block instead of $S(n)$-byte row/column indices valid within the whole matrix.

4.6 Basic hierarchical (BH) formats

4.6.1 The main idea

In [11], we relaxed the assumption of the fixed block size. The idea was generalized to partition the matrix into square disjoint *blocks* of size $2^c \times 2^c$ rows/columns, where $c \in N^+$ is a formal parameter. Coordinates of the upper left corners of these blocks are aligned to multiples of 2^c. So, indexes of nonzero elements are separated in two parts, indexes of blocks and indexes inside the blocks. Every such a block has *block row* and *block column* indexes of size $S(\lceil n/2^c \rceil)$ bits. Let $B(c)$ denote the number of nonzero blocks for matrix A. A *nonzero block* is a block that contains at least one nonzero matrix element. For storing information about the blocks and elements inside the blocks, we can use the COO or CSR format, which results in four combinations of these formats.

4.6.2 Results and applicability

A space complexity comparison to common SSF was done in [12]. The derivation of lower and upper bounds for this format was also included.

4.7 The advanced hierarchical (AH) format

4.7.1 The main idea

Another type of a hierarchical format (introduced in [11]) is a format that combines a bitmap at the top level and the COO format at the lower level. The COO format is used due to small number of elements inside each block. So, this format consists of:

- One bitmap (in this bitmap each bit=pixel represents a block of $s \times s$ elements in matrix A. If this block is nonempty, then the corresponding bit in the bitmap is set to 1 and vice versa.

4.7.2 Results and applicability

A space complexity comparison to common SSF was done in [11, 12]. The derivation of lower and upper bounds for this format was also included. By comparing results from AH and ACB formats, we see that the AH format is only slightly less efficient (about 0.47 bit per non-zero element) than the ACB format (see Section 4.1).

4.8 Summary of space-efficient formats

As was already said, we can divide our new space-efficient formats further according to the following criteria:

- the purpose of the format:
  - formats for storing large sparse matrices suitable for parallel I/O systems: ACB, BHF, AHF, MQT, MBT, CQT, CBT formats,
  - a format for acceleration of basic numeric algebra routines: $(COO_8)^4$ (described in [9]).
- the principle of the format:
  - tree based: MQT, MBT, CQT, CBT formats
  - hierarchically based:
    - 2-level: BHF, AHF formats
    - multi-level: $(COO_8)^4$ format (described in [9])
  - arithmetic coding based: the ACB format

All listed papers represent a significant contribution to fields of sparse matrix formats and related algorithms.

5 Conclusions

Large sparse matrices are widely used in high-performance computing. These matrices due to their sizes are usually stored and processed by parallel computer systems.

This work was motivated by the fact that a parallel I/O subsystem is typically the main performance bottleneck in computation with large sparse matrices, e.g., loading or storing of large sparse matrices from/to a distributed file system can take significant amounts of time. Weaknesses of the previously developed solutions for space-efficient formats for storage of large sparse matrices were discussed. Reducing the space complexity of the representation of large sparse matrices resulted in reduced time of parallel I/O. New formats for storage of large sparse matrices suitable for parallel I/O systems were designed. In particular, the first new formats were from a large family of hierarchical formats (BH and AH), the next format was arithmetical coding based (ACB) format, one new format was quadtree-based (MQT format), one new format was based on binary tree (MBT format), and the last two formats were compressed variants of the previous two (the CQT and CBT format).

We performed experiments with matrices arising in many different application areas and compared them with widely used COO or CSR formats. These experiments proved that our new formats could significantly reduce the space complexity of these matrices and consequently reduce amount of data needed for storing these matrices. Low space complexity of these formats made them good candidates for storage of large sparse matrices using parallel I/O systems.

All advances presented in this work concern both theoretical and practical areas.

References

[1] S. Balay et al. PETSc Users Manual. Technical report, Argonne National Laboratory, 2010.

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

[3] R. F. Boisvert, R. Pozo, and K. Remington. The Matrix Market Exchange Formats: Initial Design. Technical Report NISTIR 5935, National Institute of Standards and Technology, Dec. 1996.

[4] I. Duff, R. Grimes, and J. Lewi. User's Guide for the Harwell-Boeing Sparse Matrix Collection (Release I). Technical Report TR/PA/92/86, CERFACS, 1992. `http://people.sc.fsu.edu/~jburkardt/pdf/hbsmc.pdf` (accessed March 27, 2011).

[5] M. A. Heroux and J. M. Willenbring. Trilinos users guide. Technical Report SAND2003-2952, Sandia National Laboratories, 2003.

[6] M. Hoemmen, R. Vuduc, R. Nishtala, and A. Jain. Matlab (ASCII) sparse matrix format. Berkeley Benchmarking and Optimization Group. `http://bebop.cs.berkeley.edu/smc/formats/matlab.html` (accessed April 27, 2011).

[7] E. Im. *Optimizing the Performance of Sparse Matrix-Vector Multiplication - dissertation thesis.* University of California at Berkeley, 2001.

[8] I. Šimeček and D. Langr. Space-efficient sparse matrix storage formats with 8-bit indices. In *Seminar on Numerical Analysis*, pages 161–164, Liberec, 2012. Technical University of Liberec.

[9] I. Šimeček and D. Langr. Space and execution efficient formats for modern processor architectures. In *2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 98–105, Sept 2015.

[10] I. Šimeček, D. Langr, and P. Tvrdik. Minimal quadtree format for compression of sparse matrices storage. In *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'2012)*, SYNASC'2012, pages 359–364, Timisoara, Romania, sept. 2012.

[11] I. Šimeček, D. Langr, and P. Tvrdík. Space-efficient sparse matrix storage formats for massively parallel systems. In *High Performance Computing and Communication and 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-*

ICESS), HPCC'12, pages 54–60, Liverpool, Great Britain, june 2012.

[12] I. Šimeček, D. Langr, and P. Tvrdík. Tree-based space efficient formats for storing the structure of sparse matrices. *Scalable Computing: Practice and Experience*, 15(1):1–20, 2014.

[13] D. Langr, I. Šimeček, P. Tvrdík, T. Dytrych, and J. P. Draayer. Adaptive-blocking hierarchical storage format for sparse matrices. In *Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 545–551, 345 E 47TH ST, NEW YORK, NY 10017 USA, September 2012. IEEE Xplore Digital Library.

[14] D. Langr, I. Šimeček, and P. Tvrdík. Storing sparse matrices in the adaptive-blocking hierarchical storage format. In *Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS 2013)*, pages 479–486. IEEE Xplore Digital Library, September 2013.

[15] M. Martone et al. 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.

[16] M. Martone, S. Filippone, M. Paprzycki, and S. Tucci. On the usage of 16 bit indices in recursively stored sparse matrices. In *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2010 12th International Symposium on*, pages 57–64, Sept 2010.

[17] M. Martone, S. Filippone, S. Tucci, and M. Paprzycki. Assembling recursively stored sparse matrices. In *Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference on*, pages 317–325, Oct 2010.

[18] M. Martone, M. Paprzycki, and S. Filippone. An improved sparse matrix-vector multiply based on recursive sparse blocks layout. In I. Lirkov, S. Margenov, and J. Waśniewski, editors, *Large-Scale Scientific Computing*, volume 7116 of *Lecture Notes in Computer Science*, pages 606–613. Springer Berlin Heidelberg, 2012.

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

Ing. I. Šimeček, Ph.D., 5. 5. 1977

Ing. Ivan Šimeček, Ph.D. (born: 5.5.1977 in Český Krumlov) is (from 2009) is a lecturer and computer scientist at the Faculty of Information Technology (FIT), Czech Technical University in Prague, and a member of its Parallel Computing Group.

Qualifications

2008 CTU FEE in Prague, Ph.D. in Information Science and Computer Engineering,

2001 CTU FEE in Prague, Ing. in Computer science,

1995 High school (gymnázium) in Vimperk.

Employment and visiting positions

Sep 2001 – Sep 2004 PhD student, Department of Computer Science FEE, CTU, Prague

Oct 2004 – 2008 Assistant Professor, Department of Computer Science FEE CTU, Prague

2008 — today Assistant Professor, Department of Computer Systems, FIT CTU, Prague

Teaching

Lecturer in bachelor's, magister's programs (Parallel Computer Architecture, Parallel Systems and Algorithms, Programming in CUDA, Efficient Implementation of Algorithms).

Research activities and grants

His research activities include efficient design and architecture-dependent implementations of algorithms and related data structures in multi/many-threaded environments. He primarily focuses on sparse-matrix computations with the respect to cache behaviour, applied crystallography, parallel numerical linear algebra, and other compute-intensive problems. He has long-term experience with numerous types of accelerators, primarily GPGPUs. He has major merit in FIT re-achieving the status "GPU Education Center" (formerly "CUDA Teaching Centre"). From 2009 he was the executive manager of the Prague Nvidia CUDA teaching center.

- He published more than 50 scientific journals and conference publications
- Research statistics:
  - Web of Science: 22 papers, 65 citations, H-index 6

  - Scopus: 26 papers, 78 citations, H-index 6
  - Google Scholar: 61 publications, 143 citations, H-index 8
- Grants: From 2010 to 2012 he managed the CESNET project „Experimental computing grid for numerical linear algebra“. He has participated (from 2012 to 2014) in Czech Science Foundation project “Parallel Input/Output Algorithms for Very Large Sparse Matrices“. He has participated (from 2012 to 2013) in Czech Technology agency project “Data Center Management System” especially in development of CFD solver.

Selected recent papers

[1] I. Šimeček and D. Langr. Space and execution efficient formats for modern processor architectures. In *2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 98–105, Sept 2015.

[2] I. Šimeček, D. Langr, and P. Tvrdik. Minimal quadtree format for compression of sparse matrices storage. In *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'2012)*, SYNASC'2012, pages 359–364, Timisoara, Romania, sept. 2012.

[3] I. Šimeček, D. Langr, and P. Tvrdík. Space-efficient sparse matrix storage formats for massively parallel systems. In *High Performance Computing and Communication and 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS)*, HPCC'12, pages 54–60, Liverpool, Great Britain, june 2012.

[4] I. Šimeček, D. Langr, and P. Tvrdík. Tree-based space efficient formats for storing the structure of sparse matrices. *Scalable Computing: Practice and Experience*, 15(1):1–20, 2014.

[5] I. Šimeček, O. Mařík, and M. Jelínek. Utilization of gpu acceleration in le bail fitting method. *Romanian Journal of Information Science and Technology*, 18(2):182–196, 2015.

[6] I. Šimeček and P. Tvrdík. A new code transformation technique for nested loops. *COMSIS - Computer Science and Information Systems*, 11(4):1381–1416, 2014.

[7] D. Langr, P. Tvrdík, I. Šimeček, and T. Dytrych. Downsampling algorithms for large sparse matrices. *International Journal of Parallel Programming*, 43(5):679–702, 2015.

[8] I. Šimeček, J. Rohlíček, T. Zahradnický, and D. Langr. A new parallel and gpu version of a *treor*-based algorithm for indexing powder diffraction data. *Journal of Applied Crystallography*, 48(1):166–170, Feb 2015.