COSC : Combine Optimized Sparse Matrix-Vector Multiplication for CSR format

Ji-Lin Zhang, Li Zhuang, Jian Wan, Xiang-Hua Xu, Cong-Feng Jiang, Yong-Jian Ren
Cloud Computing Research Center
School of Computer Science and Technology
Hangzhou Dianzi University, Hangzhou 310037, China
Email: jilin.zhang@hdu.edu.cn, zhuangli1987@gmail.com, {wanjian, xuxhcs, cjiang, yongjian.ren}@hdu.edu.cn

***Abstract*—Sparse matrix-vector multiplication (SpMV) is the most important computational kernel, since its widely application in the scientific and engineering fields. In this paper, a series optimization methods of SpMV for CSR format have been approached. We focus on the underlying, paralleling and working set, and implement the library named as Combine Optimized SpMV for CSR format (COSC) which is the first CSR SpMV library combining with above optimization strategies and incorporates the optimizations such as explicit software pipelining, SIMDization, index compression, array padding and etc. Experiments show that the SpMV operation with optimization of COSC has average of 2.11 speedup while comparing with the non-optimized one. The implementation of COSC can be settled in SpMV and improve efficiency of hot spot.**

***Keywords*-COSC; SpMV; CSR; optimization;**

I. INTRODUCTION

In the subfield of numerical analysis, a sparse matrix is a matrix populated primarily with zeros [1]. Most of sparse matrix storage format do not store the explicit 0's, such as COO, CSR [9, 10], CSC [11], VBR [12], SSS[24], etc. CSR format has become the main storage format of sparse matrix because of its high compression ratio. Fig. 1 shows an illustration of CSR format sparse matrix. Sparse matrix-vector multiplication (SpMV) is widely used in scientific and engineering fields and most commonly in the numerical solutions of Partial Differential Equations(PDE), which frequently involve large sparse systems. Methods like Conjugate Gradient (CG) and Generalized Minimum Residual (GMRES) [2] that are employed to solve such problems use the SpMV as their basic operation. Therefore, optimization of CSR format SpMV help to improve the efficiency of the algorithm then can better solve related problems.

The current works on CSR format SpMV can be divided into three categories:

- ***Underlying Optimization:*** The optimization strategy such as SIMD optimization, loop unrolling, software prefetching are described by Samuel Williams et.al. [3]. These optimization strategy uses computing hardware features and its instruction set to improve operational efficiency and has a good performance. However, for different matrix structure and different hardware, different configuration should be carried out.
- ***Working Set Optimization:*** Kornilios Kourtis et.al. [5] discuss compression method for CSR index and the values of element and design the CSR-VI and CSR-DU which are two structures use the improve compression technology. CSR-VI performs well when sparse matrix with large repeated elements, but has bad performs with less repeated elements. CSR-DU applies for sparse matrix with large distance of non-zero elements, however, frequent type conversion becomes the main bottleneck of optimization.
- ***Paralleling Optimization:*** The multi-threading optimization of SpMV called pOSKI and the distributed optimization of SpMV named MPI-pOSKI are implemented by Ankit Jain et.al.[4]. However, pOSKI is based on the load balancing strategy of software threads, and may lead to the frequent visits of data between CPU. Moreover, its expression in the maintenance of the thread boundaries is also a considerable overhead. Therefore, it can not achieve good optimization results. The MPI-pOSKI is just a simple distributed implementation, without taking communication overhead into account, so the optimizing performance is not outstanding.

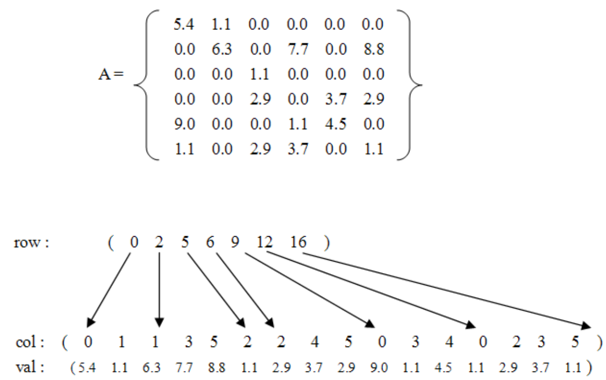

Fig. 1. An illustration of CSR Format.

Moreover, a new format called Block Compressed Sparse Row (BCSR) which is inspired by CSR format has been presented by E.Im et.al. [13, 23]. This format records only one index for each sub-block of the matrix to reduce the index data and makes the block fit into the cache size. [16] continue to optimizing the BCSR with register level optimizations, and get a good performance. Since the size of block in BCSR format affects its optimization result, the auto-tuning performance method is proposed in [25-28]. R.Vuduc et.al. [15] develop the OSKI library which is the first distributed auto-tuning library that incorporates register blocking optimization and cache blocking optimization together. On the other hand, the compression methods are widely used in the optimization of SpMV. Kornilios Kourtis in [14] shows the CSX format which uses run-length encoding to reduce the size of index array. Goumas et.al. [32-34] describe the compression optimization under paralleling. All of these optimizations depend on the distribution of non-zeros in matrix, can not get a stable optimization result. In addition, several code-level optimization techniques in SpMV are presented in [29-31], and the basic study for SpMV are reported in [17-22]. In short, the SpMV kernel, although very simple in its essence, is difficult to optimize and has attracted much attention from researchers due to its importance.

To improving SpMV and optimizing its performance, we implement an arithmetic library under CSR format, named as Combine Optimized SpMV for CSR format (COSC). It's the first SpMV library for CSR format which supports to incorporate the optimization strategies together to get the best performance. The remainder of this paper is organized as follows: Section 2 describes three types of optimization technology in the implementation of COSC; Section 3 presents the experimental evaluation of the optimization effect of COSC by tested with the set of matrices from [6]; Section 4 draws our conclusion.

II. Optimization Technology

The COSC library implements three types of optimization strategy, such as underlying, paralleling and working set. The underlying optimization mainly involves SIMDization and pipelining while paralleling optimization includes thread-level paralleling and distributed paralleling. As mentioned above, three types of optimization discussed in this sectoin are all based on CSR format SpMV operations. The SpMV operation is easily implemented for the matrix stored in CSR format as Algorithm 1 shows. X represents the vector need to be calculated and Y represents vector after SpMV operation. if note nnz as the number of non-zero elements of sparse matrix, the time complexity of matrix vector multiplication is $O(nnz)$ which is noted as CSR-Naive in this paper for comparison.

Algorithm 1: MULTIPLY($\mathcal{A}$)
The CSR Format Sparse Matrix-Vector Multiplication Algorithm

Require: N : the total number of rows in matrix $\mathcal{A}$
val : nonzero values in matrix $\mathcal{A}$
col : column indices of values in matrix $\mathcal{A}$
row : pointers to row starts in matrix $\mathcal{A}$
Input : X : the vector need to calculate
Output : Y : the vector after calculate

for $i \leftarrow 0$ **to** $N-1$ **do**
  for $j \leftarrow row_i$ **to** row_{i+1}-*1* **do**
    $Y_i \leftarrow Y_i + val_j * X_{col_j}$
  end
end

A. Underlying Optimization

1) SIMDization: It's better to use SIMD which goals to implement the data parallel. And the SSE [7][8] which short for Streaming SIMD Extensions is the classic implementation of SIMD. Samuel Williams et. al.[3] point out that the implementations of SpMV that used the SSE intrinsics performed significantly better than straight C code on the Intel Clovertown, but they have not given the specific implementation.

It is observable that on $N \times M$ matrix, SpMV can be treated as M-dimensional vector dot product of N times. In view of the high paralleling of vector dot product, COSC applies SSE intrinsics on SpMV optimization. By filling zero, the number of bytes of non-zero elements taken is supplemented a multiple of 16. Hence the vectors to participate in the operation are divided into several groups to 16 bytes of atomic unit vector set. Then we apply the arithmetic intrinsics of SSE to rapid vector dot product. Since SpMV under CSR format using col array for positioning elements in X, prefetching the col elements can also improves the hit rate of CPU Cache. Algorithm 2 presents its detail implementation.

Algorithm 2: SIMDIZATION($\mathcal{A}$)
SpMV with SIMDization optimization

Require: N : the total number of rows in matrix $\mathcal{A}$
val : nonzero values in matrix $\mathcal{A}$
col : column indices of values in matrix $\mathcal{A}$
row : pointers to row starts in matrix $\mathcal{A}$
det : the distance of prefetching
Input : X : the vector need to calculate
Output : Y : the vector after calculate

for $i \leftarrow 0$ **to** $N-1$ **do**
  for $j \leftarrow row_i$ **to** row_{i+1}-*1* **do**
    $vals \leftarrow \{val_j, val_{j+1}, val_{j+2}, val_{j+3}\}$
    $xs \leftarrow \{X_{col_j}, X_{col_{j+1}}, X_{col_{j+2}}, X_{col_{j+3}}\}$
    $z \leftarrow \_mm\_mul\_ps(vals, xs)$
    $Y_i \leftarrow sum\{z\}$
    $\_mm\_prefetch(col_{j+det})$
    $j \leftarrow j + 4$
  end
end

2) Pipelining: Pipelining achieves effectively schedule by overlapping continuous loop entities. Since indirect memory access operations of inner loops in CSR-Naive cost much, Ankit Jain at [4] shows the Software Pipelined CSR Algorithm which re-organizes the operation sequences to improving the performance of SpMV. And he also point out this optimization does not help SpMV performance significantly since the bottleneck on most architectures is the memory-to-cache bandwidth and not the cache-to-register bandwidth. So this optimization has not been implemented in pOSKI.

Based on Ankit Jain's work, COSC continues focus to pipelining. We use loop unrolling to merge several iteration calculation into one calculation. We also use the rich registers resources to pipeline address and calculate operation by putting the similarity operations together. Comparing to Ankit Jain [4], the implementation of our strategy has better optimization effect.

B. Working Set Optimization

In CSR format, we typically use 4 bytes to store the index. Attention to the general situation that the distance between non-zero elements in one row of sparse matrix is not large, Kornilios Kourtis [5] propose compression technology by means of reducing the working set size for improving memory bandwidth utilization to optimize the performance of SpMV. They propose two optimization methods named CSR-DU and CSR-VI. CSR-DU use run-length to compress the CSR index data by bit while CSR-VI only simply compress the repeated value. Meanwhile they also point out that the performance of these two methods depends on the distribution and value of non-zero elements in matrix. Based on [5], we simply implement CSR-DU and CSR-VI optimization strategies in COSC.

C. Paralleling Optimization

1) Thread-Level Paralleling: The loading balancing is the key issue in thread-level paralleling. pOSKI uses the simple distribution strategy which makes itself hard to rising the performance. To break the bottleneck in pOSKI, the implementation of thread-level paralleling in COSC improving the distribution strategy.

We use pure hardware thread, binding the thread with CPU through built-in functions, divide matrix into several pieces with equally number of non-zero elements and separately deliver matrix pieces to CPU processes. So each thread just need to maintain the starting and ending rows of the regional block, avoid the maintenance of the starting and ending columns in [4], reduce unnecessary costs and also guarantee load balance among various threads. Algorithm 3 shows the detail implementation of distribution strategy.

Algorithm 3: DISTRIBUTION($\mathcal{A}$)
The distribution strategy for Thread-Level Paralleling

Require: N : the total number of rows in matrix $\mathcal{A}$
nnz : the number of non-zeros in matrix $\mathcal{A}$
row : pointers to row starts in matrix $\mathcal{A}$
$maxcore$: the number of cores
Output : $thread$: the array of distribution result

$avg \leftarrow (nnz + N)/maxcore$
$det \leftarrow avg$
$now \leftarrow 1$
$thread[0] \leftarrow 0$
for $i \leftarrow 1$ **to** $maxcore$ **do**
  while $now <= N$ **do**
    if $row[now] + now >= det$ **then**
      $det \leftarrow det + avg$
      $break$
    end
    $thread[i] \leftarrow now$
    $now \leftarrow now$ + 1
  end
end

2) Distributed Paralleling: Effective communication between the nodes and organization high performance computing network is becoming a trend in computing today. COSC is first distributed paralleling CSR SpMV library which can be deployed in the real-world. Although the expansion of pOSKI — MPI-pOSKI [4] is a distributed paralleling SpMV based on MPI. But it is only a simple implementation of SpMV with MPI, never consider the communication cost between the nodes and can not be applied in real-world.

Therefore, the implementation of distributed paralleling optimization of COSC focuses on the balance between the allocation of calculation of nodes and communication overhead. We distributes calculation of nodes by rows. Set row_i represents the non-zero elements until i_{th} row. $[r_{i-1}, r_i)$ represents the calculation interval of i_{th} node by dividing rows. S_1 represents calculation cost of one float data while S_2 represents transport cost of one float data. Then the distribution strategy of calculation of SpMV can be converted to the following problem, when $2 \le i \le n$,

$$\begin{aligned} Z = \min((row_{r1} - row_{r0}) \times S_1 + (r_n - r_1) \times S_2, \\ \max((row_{r_i} - row_{r_{i-1}}) \times S_1)), \end{aligned} \tag{1}$$

where n is the total number of nodes, master node is numbered as 1 and r_0 is 0.

To simplify the model, we use the strategy that average allocating calculation except the master node. So we only need to consider the calculation on the master node. From equation 1, we can simply emulate the value of r_1. Algorithm 4 presents the detail implementation of distribution strategy.

III. EXPERIMENTS

In the experiments, we use cluster of four nodes with Intel Xeon 5150 (Woodcrest architecture), 2.33 GHZ dual-core processors, 3.5G memory, CentOS 5.4 with Linux kernel 2.6.18, GCC 4.1.2 and MPICH 2-1.0.2. Compilation options in experiments is shown in table 1. We use 58 matrices mentioned in [6] as test data and we apply 100 times SpMV on every matrix for get more accuracy results. The details of 58 matrices show in table 2.

Figure 2 presents the average SpMV speedup of test matrices through the single optimization strategy in COSC. The x-axis is the optimization strategy we have used while y-axis represents the average speedup. We can easy found the SIMDization optimization get the best performance which the speedup is 1.2569 while the speedup under pipelining

Algorithm 4: ALLOCATION($\mathcal{A}$)
The distribution strategy for Distributed Paralleling

Require: N : the total number of rows in matrix $\mathcal{A}$
nnz : the number of non-zeros in matrix $\mathcal{A}$
row : pointers to row starts in matrix $\mathcal{A}$
$maxnode$: the number of cluster nodes
$r1$: the calculation on the master node
Output : $node$: the array of allocation result

$now \leftarrow r_1$
$avg \leftarrow (nnz - row[now])/(maxnode - 1)$
$det \leftarrow avg + row[now]$
$node[0] \leftarrow 0$
$node[1] \leftarrow now$
for $i \leftarrow 2$ **to** $maxnode$ **do**
  $node[i] \leftarrow now$
  while $now <= N$ **do**
    if $row[now] >= det$ **then**
      $det \leftarrow det + avg$
      $break$
    end
    $node[i] \leftarrow now$
    $now \leftarrow now + 1$
  end
end

TABLE I
COMPILATION OPTIONS

Optimization Technology	Compilation Options
Thread-level Paralleling	gcc -O0 -lpthread
Distributed-level Paralleling	mpicxx -O0
SIMDization	gcc -O0 -msse3 -march=x86-64
Others	gcc -O0

optimization is not well which the speedup is only 1.052. This result correspond with our description in Section 2.

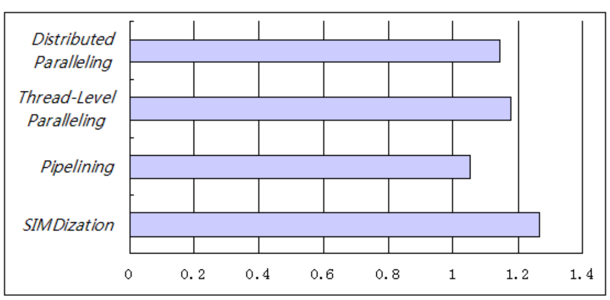

Fig. 2. The average speedup through the single Optimization strategy of COSC.

Figure 3 shows the SpMV speedup of test matrices through the best combination of optimization strategies in COSC. The x-axis is the name of matrix while y-axis represents speedup. The highest speedup is 3.7149 with av41092, the lowest speedup is 1.1579 with lns_3937 and the average speedup is 2.11016. Combining table 2, optimization and distribution of non-zero matrix elements are relative. For example, average number of non-zero elements per row of ship_001, bundle1 and sme3Db is more than 70, so the speedup is relatively high, while average number of non-zero elements per row of lns_3937, bayer02, orsreg_1 is small and its speedup relatively low.

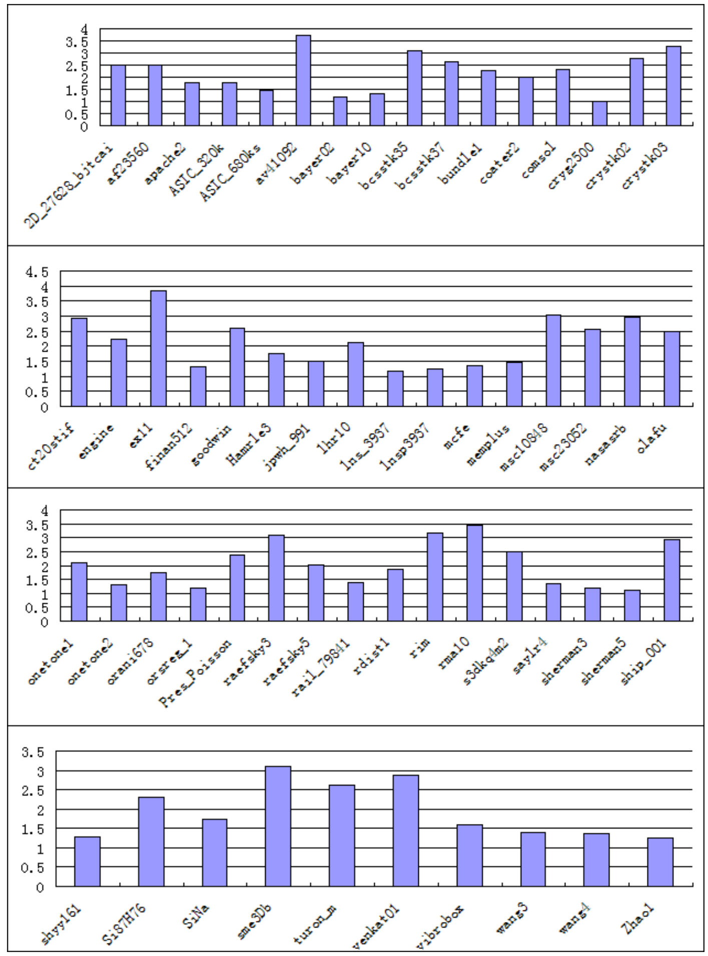

Fig. 3. The Speedup through the Optimization of COSC.

From figure 2 and figure 3, we can found the rising of performance with single strategy is limited. So it's a better way to applying these optimization strategies together. That's why the speedup of the best strategy is only 1.269 while the combination one can be 2.11016. But for the extension of COSC, the method of finding out the best optimization combination is need to researching. Since the number of strategy type in COSC are not large, we enumerate all of their combination to get the best one in our experiments.

IV. CONCLUSIONS

In this paper, we propose and implement a SpMV library named COSC based on CSR format and expriment optimization with set of matrices posted in [6]. Experiments show that the speedup of SpMV through COSC is average of 2.11, compared to SpMV of non-optimization. In the experiment, we found the optimization results and the distribution of non-zero

TABLE II
THE SET OF MATRICES

Name	Number of Row	Number of Columns	AVG of Non-zeros per Row	Number of Non-zeros
2D_27628_bjtcai	27628	27628	7.480454611	206670
af23560	23560	23560	19.55	460598
apache2	715176	715176	6.736621475	4817870
ASIC_320k	321821	321821	6.0028028	1931828
ASIC_680ks	682712	682712	2.480939254	1693767
av41092	41092	41092	40.978828	1683902
bayer02	13935	13935	4.54302117	63307
bayer10	13436	13436	5.328520393	71594
bcsstk35	30237	30237	47.95988359	1450163
bcsstk37	25503	25503	44.73893267	1140977
bundle1	10581	10581	72.84859654	770811
coater2	9540	9540	21.73039832	207308
comsol	1500	1500	65.09666667	97645
cryg2500	2500	2500	4.9396	12349
crystk02	13965	13965	69.35789474	968583
crystk03	24696	24696	70.90937804	1751178
ct20stif	52329	52329	49.69128017	2600295
engine	14357	14357	327.7894407	4706073
ex11	16614	16614	66.02552065	1096948
finan512	74752	74752	7.98630137	596992
goodwin	7320	7320	44.36775956	324772
Hamrle3	1447360	1447360	3.809862094	5514242
jpwh_991	991	991	6.081735621	6027
lhr10	10672	10672	21.40133058	228395
lns_3937	3937	3937	6.453390907	25407
lnsp3937	3937	3937	6.453390907	25407
mcfe	765	765	31.87189542	24382
memplus	17758	17758	5.583230093	99147
msc10848	10848	10848	113.3643068	1229776
msc23052	23052	23052	49.56992886	1142686
nasasrb	54870	54870	48.79394933	2677324
olafu	16146	16146	62.87352905	1015156
onetone1	36057	36057	9.306154145	335552
onetone2	36057	36057	6.173447597	222596
orani678	2529	2529	35.6496639	90158
orsreg_1	2205	2205	6.40952381	14133
Pres_Poisson	14822	14822	48.29334773	715804
raefsky3	21200	21200	70.22490566	1488768
raefsky5	6316	6316	26.4689677	167178
rail_79841	79841	79841	6.93780138	553921
rdist1	4134	4134	22.83696178	94408
rim	22560	22560	44.98896277	1014951
rma10	46835	46835	49.72973204	2329092
s3dkq4m2	90449	90449	48.95272474	4427725
saylr4	3564	3564	6.261503928	22316
sherman3	5005	5005	4.002597403	20033
sherman5	3312	3312	6.27807971	20793
ship_001	34920	34920	111.5835052	3896496
shyy161	76480	76480	4.311741632	329762
Si87H76	240369	240369	44.35526628	10661631
SiNa	5743	5743	34.6137907	198787
sme3Db	29067	29067	71.59538308	2081063
turon_m	189924	189924	8.902908532	1690876
venkat01	62424	62424	27.51813405	1717792
vibrobox	12328	12328	24.47274497	301700
wang3	26064	26064	6.797421731	177168
wang4	26068	26068	6.797452816	177196
Zhao1	33861	33861	4.915773309	166453

matrix elements have close relations. The more of the number of non-zero elements per row, the more obvious optimization of COSC. The implementation of COSC can be settled in SpMV and improve efficiency of hot spot.

ACKNOWLEDGMENT

This paper is supported by State Key Development Program of Basic Research of China under grant No.2007CB310900, Natural Science Fund of China under grant No.61003077, 60873023, 60973029, Zhejiang Provincial Natural Science Foundation under grant No.Y1101104, Y1101092, Y1090940 and the priming scientific research foundation of Hangzhou Dianzi University under grant No.KYS055610004, KYS055608109.

REFERENCES

[1] Stoer, Josef, Bulirsch, Roland. Introduction to Numerical Analysis (3rd ed.), Berlin, New York: Springer-Verlag, pp.619, 2002.

[2] Y. Saad. Iterative Methods for Sparse Linear Systems. SIAM, Philadelphia, PA, USA, 2003.

[3] Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, and James Demmel. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. Parallel Computing, 35(3):178 - 194, 2009.

[4] Ankit Jain. pOSKI: An Extensible Autotuning Framework to Perform Optimized SpMVs on Multicore Architectures, Technical Report (pending), MS Report, EECS Department, University of California, Berkeley, 2008.

[5] Kornilios Kourtis, Georgios Goumas, Nectarios Koziris. Optimizing Sparse Matrix-Vector Multiplication Using Index and Value Compression, Computing Frontiers 2008.

[6] T. Davis. The University of Florida sparse matrix collection [EB/OL]. Nan Digest, vol.97, no.23 http://www.cise.ufl.edu/research/sparse/matrices, June 1997.

[7] Intel 64 and IA-32 Architectures Optimization Reference Manual, http://www.intel.com/Assets/PDF/manual/248966.pdf, January 2011.

[8] Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, http://www.intel.com/Assets/PDF/manual/253667.pdf, January 2011.

[9] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. M. 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, 1994.

[10] Y. Saad, Iterative Methods for Sparse Linear Systems. Philadelphia, PA, USA: SIAM, 2003.

[11] I. DUFF, R. GRIMES, AND J. LEWIS, Sparse matrix test problems, ACM Trans. Math. Soft, 15, pp. 1-14, 1989.

[12] Y. Saad. SPARSKIT: A basic tool kit for sparse matrix computations, 1994.

[13] E. Im, Optimizing the Performance of Sparse Matrix-Vector Multiplication, Ph.D.thesis, University of California, May 2000.

[14] Kornilios Kourtis. Software Development and Data Compression Techniques for Improving Performance Computing Applications on Parallel Shared Memory Architectures Ph.D.thesis, National Technical University of Athens, April 2010.

[15] R. Vuduc, J. Demmel, and K. Yelick. Oski: A library of automatically tuned sparse matrix kernels, 2005.

[16] E. Im, K. A. Yelick.Optimizing Sparse Matrix-Vector Multiplication for Register Reuse, International Conference on Computational Science, San Francisco, California, May 2001.

[17] R. C. Agarwal, F. G. Gustavson, and M. Zubair. A high performance algorithm using pre-processing for the sparse matrix-vector multiplication. In Supercomputing'92, pages 32-41, Minn, MN, November 1992.

[18] O. Temam and W. Jalby. Characterizing the behavior of sparse algorithms on caches. In Supercomputing'92, pages 578C587, Minnesota., MN, November 1992.

[19] U. V. Catalyuerek and C. Aykanat. Decomposing irregularly sparse matrices for parallel matrix-vector multiplication. Lecture Notes In Computer Science, 1117:75-86, 1996.

[20] S. Toledo. Improving the memory-system performance of sparse-matrix vector multiplication. IBM Journal of Research and Development, 41(6):711-725, 1997.

[21] J. White and P. Sadayappan. On improving the performance of sparse matrix-vector multiplication. In HiPC '97: 4th International Conference on High Performance Computing, 1997.

[22] A. Pinar and M. T. Heath. Improving performance of sparse matrix-vector multiplication. In Supercomputing'99, Portland, OR, November 1999.

[23] E. Im and K. Yelick. Optimizing sparse matrix-vector multiplication on SMPs. In 9th SIAM Conference on Parallel Processing for Scientific Computing. SIAM, March 1999.

[24] R. Geus and S. Rollin. Towards a fast parallel sparse matrix-vector multiplication. In Parallel Computing: Fundamentals and Applications, International Conference ParCo, pages 308-315. Imperial College Press, 1999.

[25] R. Vuduc, J. Demmel, K. Yelick, S. Kamil, R. Nishtala, and B. Lee. Performance optimizations and bounds for sparse matrix-vector multiply. In Supercomputing, Baltimore, MD, November 2002.

[26] Richard Vuduc. Automatic Performance Tuning of Sparse Matrix Kernels. PhD thesis, University of California, Berkeley, 2003.

[27] Alfredo Buttari, Victor Eijkhout, Julien Langou, and Salvatore Filippone. Performance optimization and modeling of blocked sparse kernels. IJHPCA, 21:467-484, 2007.

[28] R. W. Vuduc and H. Moon. Fast sparse matrix-vector multiplication by exploiting variable block structure. In High Performance Computing and Communications, volume 3726 of Lecture Notes in Computer Science, pages 807-816. Springer, 2005.

[29] J. Mellor-Crummey and J. Garvin. Optimizing sparse matrix-vector product computations using unroll and jam. International Journal of High Performance Computing Applications, 18(2):225, 2004.

[30] J. C. Pichel, D. B. Heras, J. C. Cabaleiro, and F. F. Rivera. Improving the locality of the sparse matrix-vector product on shared memory multiprocessors. In PDP, pages 66-71. IEEE Computer Society, 2004.

[31] J. C. Pichel, D. B. Heras, J. C. Cabaleiro, and F. F. Rivera. Performance optimization of irregular codes based on the combination of reordering and blocking techniques. Parallel Computing, 31(8-9):858-876, 2005.

[32] V. Karakasis, G. Goumas, and N. Koziris. Exploring the effect of block shapes on the performance of sparse kernels. In IEEE International Symposium on Parallel and Distributed Processing (Workshop on Parallel and Distributed Scientific and Engineering Computing with Applications), Rome, Italy, 2009.

[33] V. Karakasis, G. Goumas, and N. Koziris. A comparative study of blocking storage methods for sparse matrices on multicore architectures. In 12th IEEE International Conference on Computational Science and Engineering (CSE-09), Vancouver, Canada, 2009.

[34] Kornilios Kourtis, Georgios Goumas, Nectarios Koziris.Improving the Performance of Multithreaded Sparse Matrix-Vector Multiplication using Index and Value Compression, ICPP, 2008.