LETTER

Performance Optimization for Sparse A^tAx in Parallel on Multicore CPU

Yuan TAO[†,††,†††a)], **Yangdong DENG**[††††b)], ***Nonmembers***, **Shuai MU**[††††c)], ***Member***, **Zhenzhong ZHANG**[††d)], **Mingfa ZHU**[††e)], **Limin XIAO**[††f)], ***and*** **Li RUAN**[††g)], ***Nonmembers***

SUMMARY The sparse matrix operation, $y \leftarrow y+A^tAx$, where A is a sparse matrix and x and y are dense vectors, is a widely used computing pattern in High Performance Computing (HPC) applications. The pattern poses challenge to efficient solutions because both a matrix and its transposed version are involved. An efficient sparse matrix format, Compressed Sparse Blocks (CSB), has been proposed to provide nearly the same performance for both Ax and A^tx. We develop a multithreaded implementation for the CSB format and apply it to solve $y \leftarrow y+A^tAx$. Experiments show that our technique outperforms the Compressed Sparse Row (CSR) based solution in POSKI by up to 2.5 fold on over 70% of benchmarking matrices.

key words: *sparse A^tAx, compressed sparse block, compressed sparse rows, multicore platform*

1. Introduction

Sparse matrices are extensively used in HPC applications because they properly capture the natural properties of engineering and scientific problems [1]. Among various computation patterns, $y \leftarrow y+A^tAx$ is an essential one due to its usage in singular value decomposition [2], [3] and other applications. Vuduc et al. [3] provide optimization strategies for the memory hierarchy as well as performance bounds of computing $y \leftarrow y+A^tAx$. With multicore platforms becoming prevalent in the HPC community, the exploitation of increasing parallelism by designing efficient parallel algorithms will be the dominant means to attack the increasing problem size. However, there has been few works on developing parallel solution for $y \leftarrow y+A^tAx$ due to the lack of effective data structures supporting the computing of both Ax and A^tx. Recently, an efficient sparse matrix format, Compressed Sparse Blocks (CSB) [4], has been proposed to provide nearly the same performance on the above mentioned problems.

In this work, we develop an efficient parallel solution for the CSB data structure. Our techniques are tested on a multicore platform and compared with the Compressed Sparse Row (CSR) based techniques used in the latest Parallel Optimized Sparse Kernel Interface Library (POSKI) [5]. Our multicore parallel solution outperforms POSKI on over 70% of benchmark sparse matrices up to 2.5 fold.

2. Preliminaries

In this section, we briefly review the preliminaries of this work. First, the CSB data structure is explained. Then we explain the software and hardware configuration for the experiment as well as the benchmarking matrices used in this work.

2.1 CSB

Buluc et al. [4] introduced the CSB to store a sparse matrix to enable efficient computations of both Ax and A^tx. As illustrated in Fig. 1, a sparse matrix A is partitioned into $\beta \times \beta$ blocks designated as $A_{i,j}$. Such a row (column) of blocks is coined as blockrow (blockcolumn). Inside each block of sub-matrix, the non-zeros are stored with their intra-block row and column coordinates. Because CSB provides better locality for vectors x and y, it offers roughly identical performance on both Sparse Matrix Vector Product (SMVP) and Sparse Matrix Transpose Vector Product (SMTVP) problems.

$$A = \begin{pmatrix} A_{0,0} & A_{0,1} & \dots & A_{0,\frac{n}{\beta}-1} \\ A_{1,0} & A_{1,1} & \dots & A_{1,\frac{n}{\beta}-1} \\ \vdots & \vdots & \ddots & \vdots \\ A_{\frac{n}{\beta}-1,0} & A_{\frac{n}{\beta}-1,1} & \dots & A_{\frac{n}{\beta}-1,\frac{n}{\beta}-1} \end{pmatrix}$$

2.2 Experimental Setup

Table 1 lists the software and hardware platform for our experiments. Our parallel implementation is built in Cilk multithreaded language [6]. Table 2 lists the benchmarking matrices, which were all used in [3] and downloaded from the University of Florida sparse matrices collection [7]. We compare our implementations with POSKI.

Manuscript received August 30, 2013.

[†]The author is with State Key Laboratory of Software Development Environment, Beihang University, Beijing 100191, China.

[††]The authors are with School of Computer Science and Engineering, Beihang University, Beijing 100191, China.

[†††]The author is with College of Mathematics, Jilin Normal University, Jilin 136000, China.

[††††]The authors are with Institute of Microelectronics, Tsinghua University, China.

a) E-mail: taoyuan@cse.buaa.edu.cn

b) E-mail: dengyd@tsinghua.edu.cn

c) E-mail: mus04ster@gmail.com

d) E-mail: zzzhang0118@gmail.com

e) E-mail: zhumf@buaa.edu.cn (Corresponding author)

f) E-mail: xiaolm@buaa.edu.cn

g) E-mail: ruanli@buaa.edu.cn

DOI: 10.1587/transinf.E97.D.315

Table 1 Test-bed used in this work.

CPU	Memory	OS	Kernel	Compiler	Runtime Version	
Intel*R* Core*T M* i7-3770	32G	Centos 6.3 x86_64	3.4.56	Red Hat 4.4.6-4	POSKI 1.0.0	cilk_8503-x86_64

$y \leftarrow y+A^tAx$, A is sparse matrix with CSB	$y \leftarrow y+A^tAx$, A is sparse matrix with CSR

Input: Sparse matrix A in CSB format;
Vector x
Output:Vector y
rowmax(colmax) = maximum number of non-zeros among blockrow (blockcolumn)
rowavg(colavg) = average number of non-zeros over blockrows(blockcolumns)
ratio=rowmax (colmax) / rowavg (colavg)
 	
// $y \leftarrow y+Ax$
if (threads>1 && ratio>2){
parallel SMVP
}else{
sequential SMVP
}
// $y \leftarrow y+A^tx$
if (threads>1 && ratio>2){
parallel SMTVP
}else{
sequential SMTVP
}
	Input: Sparse matrix A in CSR format of POSKI;
	Vector x
	Output: Vector y
	
	Vector z
	poski_Init();
	//Creating default thread object
	poski_threadarg_t *poski_thread =
	poski_InitThreads();
	convert A to format of POSKI;
	double alpha=1.0,beta=0.0;
	// $z=Ax$
	poski_MatMult(A_tunable,
	OP_NORMAL,alpha,x,beta,z);
	//$y=A^tz$
	poski_MatMult(A_tunable,
	OP_TRANS,alpha,z,beta,y);

Fig. 1 Algorithms for computing $y \leftarrow y+A^tAx$ with CSB (our work) and CSR (POSKI) formats.

```
Parallel SMVP (SMTVP) using CSB
//threshold: a value to identify a block is dense or sparse

for each blockrow (blockcolumn) in parallel
{
    for each block along blockrow (blockcolumn)
    {
        if (the dense degree of block > threshold){
            recursively divide the dense block into sub-blocks
            do SMVP (SMTVP) for each sub-block in parallel
        }else{
            packing multiple blocks into one block
            do SMVP (SMTVP) for this block in parallel with other block
        }
    }
}
```

Fig. 2 Pseudo code for parallel computing of SMVP and SMTVP.

3. Implementation

In this section, we provide implementation details of the algorithms for computing CSB based $y \leftarrow y+A^tAx$ in Cilk and CSR based $y \leftarrow y+A^tAx$ in POSKI. Both algorithms are designed by first performing SMVP and then computing SMTVP which are supported by CSB [4]. Figure 1 lists the CSB based algorithm of ours in details and the CSR based algorithm used by POSKI. Figure 2 gives details of parallel computing SMVP or SMTVP.

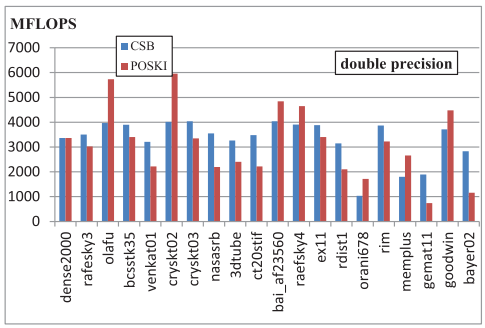

Fig. 3 Performance of CSB and POSKI on CPU for first twenty matrices (CSB is our implement for $y \leftarrow y+A^tAx$ in CSB format, and POSKI is in CSR format of POSKI for comparison).

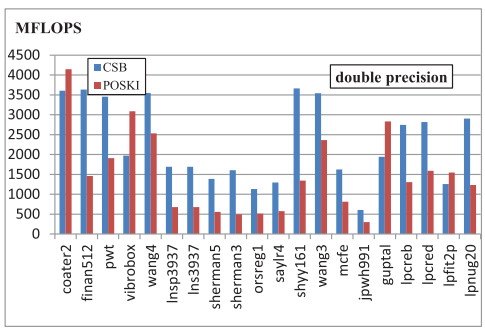

Fig. 4 Performance of CSB and POSKI on CPU for the remainder matrices.

4. Experimental result

We evaluate the proposed techniques on an Intel i7-3770 CPU running 8 threads on 4 cores. We compare the performance of our approach with that of the latest POSKI package. Figures 3 and 4 show the experimental results. We use double precision data, which is the only supported data type by POSKI. Our implementation outperforms POSKI on over 70% of the benchmarking matrices. In addition, our implementation delivers a higher level of performance than POSKI on all matrices with over 36,000 rows and/or 36,000 columns.

The performance advantage of our techniques are mainly due to the better locality made possible by the CSB data structure in both vector x and vector y. CSB also allows accessing the non-zeros of sparse matrices in the same storage format and data layout when computing SMVP and SMTVP. Such a feature significantly saves communicating demand between memory and processor on multi-core platforms. One remaining problem of our work is that matrix A has to be loaded twice to compute $y \leftarrow y+A^tAx$. In the future

Table 2 Benchmarking matrices used in this work.

Name	Description	Row/Column	Non-zeros	Non-zero per column Mean/Max
dense2000	Dense matrix	2000 / 2000	4000k	2000 / 2000
rafesky3	Fluid structure interaction turbulence	21200 / 21200	1488k	70 / 80
olafu	Accuracy problem on Y-MP	16146 / 16146	515k	31 / 60
bcsstk35	Stiffness matrix, automobile seat frame and body attachment	30237 / 30237	740k	24 / 166
venkat01	Unstructured 2D Euler solver	62424 / 62424	1717k	27 / 44
cryskt02	Fem crystal free vibration stiffness matrix	13965 / 13965	491k	35 / 42
cryskt03	Fem crystal free vibration stiffness matrix	24696 / 24696	887k	35 / 42
nasasrb	Structure from NASA langley, shuttle rocket booster	54870 / 54870	1366k	24 / 57
3dtube	3-D pressure tube	45330 / 45330	1629k	35 / 2358
ct20stif	CT20 engine block - stiffness matrix	52329 / 52329	1375k	26 / 207
bai_af23560	NACA airfoil eigenvalue calculation	23560 / 23560	484k	20 / 21
raefsky4	Buckling problem for container model	19779 / 19779	674k	34 / 72
ex11	Computational fluid dynamics problem	16614 / 16614	1096k	66 / 90
rdist1	Chemical process separation	4134 / 4134	94k	22 / 81
orani678	economic problem	2529 / 2529	90k	35 / 1110
rim	Fluid mechanics problem	22560 / 22560	1014k	44 / 112
memplus	Memory circuit	17758 / 17758	126k	7 / 574
gemat11	Power network problem sequence	4929 / 4929	33k	6 / 27
goodwin	Fluid mechanics problem	7320 / 7320	324k	44 / 112
bayer02	Chemical process simulation problem	13935 / 13935	63k	4 / 22
coater2	Tail end of a converging slot coater	9540 / 9540	207k	21 / 63
finan512	Portfolio optimization	74752 / 74752	335k	4 / 17
pwt	Structural problem	36519 / 36519	181k	4 / 15
vibrobox	Vibroacoustic problem	12328 / 12328	177k	14 / 78
wang4	Semiconductor device problem	26068 / 26068	177k	6 / 7
lnsp3937	Computational fluid dynamics problem	3937 / 3937	25k	6 / 11
lns3937	Computational fluid dynamics problem	3937 / 3937	25k	6 / 11
sherman5	Computational fluid dynamics problem	3312 / 3312	20k	6 / 21
sherman3	Computational fluid dynamics problem	5005 / 5005	20k	4 / 7
orsreg1	Oil reservoir simulation	2205 / 2205	14k	6 / 7
saylr4	Computational fluid dynamics problem	3564 / 3564	12k	3 / 4
shyy161	Computational fluid dynamics problem	76480 / 76480	329k	4 / 6
wang3	Semiconductor device problem	26064 / 26064	177k	6 / 7
mcfe	2D / 3D problem	765 / 765	24k	31 / 81
jpwh991	Semiconductor device problem	991 / 991	6027	6 / 16
guptal	Linear programming matrix	31802 / 31802	1098k	34 / 109
lpcreb	Linear programming problem	9648 / 77137	260k	27 / 844
lpcred	Linear programming problem	8926/ 73948	246k	27 / 808
lnfit2p	Linear programming problem	3000/ 13525	50k	16 / 22
lpnug20	Linear programming problem	15240 / 72600	304k	20 / 20

work, we are going to explore techniques to improve the I/O complexity.

5. Conclusion

CSB enables to compute both SMVP and SMTVP with a

similar level of performance. Such a capability makes CSB more suitable for $y \leftarrow y + A^t Ax$ on multi-core platforms than the CSR based approach used in POSKI. We developed multithreaded implementations for CSB on modern multi-core machines and observed significant performance advantage.

Acknowledgments

The work is partially supported by the Hi-tech Research and Development Program of China (863 Program) under Grant No.2011AA01A205, supported by the National Natural Science Foundation of China under Grant No.61370059, supported by the National Natural Science Foundation of China under Grant No.61003015, supported by the Doctoral Fund of Ministry of Education of China under Grant No.20101102110018, supported by Beijing Natural Science Foundation under Grant No.4122042, supported by the fund of the State Key Laboratory of Software Development Environment under Grant No.SKLSDE-2012ZX-07, supported by the National Natural Science Foundation of China under Grant No.61370059, and supported by the National Natural Science Foundation of China under Grant No.61232009.

References

[1] S.I. Duff, "A survey of sparse matrix research," Proc. IEEE, vol.65, no.4, pp.500–535, April 1977.
[2] J.W. Demmel, Applied Numerical Linear Algebra, SIAM, 1997.
[3] R. Vuduc, A. Gyulassy, J.W. Demmel, and K.A. Yelick, "Memory hierarchy optimizations and performance bounds for sparse ATAx," ICCS 2003, Melbourne and Petersburg, pp.705–714, June 2003.
[4] A. Buluc, J.T. Fineman, M. Frigo, J.R. Gilbert, and C.E. Leiserson, "Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks," 21st ACM Symposium on Parallelism in Algorithms and Architectures, pp.233–244, Calgary, Aug. 2009.
[5] J.H. Byun, R. Lin, J.W. Demmel, and K.A. Yelick, "pOSKI: Parallel Optimized Sparse Kernel Interface Library," http://bebop.cs.berkeley.edu/poski/, accessed Aug. 25, 2013.
[6] C.E. Leiserson, The Cilk Project, http://supertech.csail.mit.edu/cilk/, accessed Aug. 25, 2013.
[7] T. Davis and Y.F. Hu, The University of Florida Sparse Matrix Collection, http://www.cise.ufl.edu/research/sparse/matrices/, accessed Aug. 25, 2013.