Parallel Blocked Sparse Matrix-Vector Multiplication with Dynamic Parameter Selection Method

Makoto Kudo[1], Hisayasu Kuroda[2], and Yasumasa Kanada[2]

[1] Department of Computer Science
Graduate School of Information Science and Technology
The University of Tokyo
[2] Super Computing Division, Information Technology Center
The University of Tokyo
{mkudoh,kuroda,kanada}@pi.cc.u-tokyo.ac.jp

Abstract. A blocking method is a popular optimization technique for sparse matrix-vector multiplication (SpMxV). In this paper, a new blocking method which generalizes the conventional two blocking methods and its application to the parallel environment are proposed. This paper also proposes a dynamic parameter selection method for blocked parallel SpMxV which automatically selects the parameter set according to the characteristics of the target matrix and machine in order to achieve high performance on various computational environments. The performance with dynamically selected parameter set is compared with the performance with generally-used fixed parameter sets for 12 types of sparse matrices on four parallel machines: including PentiumIII, Sparc II, MIPS R12000 and Itanium. The result shows that the performance with dynamically selected parameter set is the best in most cases.

1 Introduction

Sparse matrix-vector multiplication (SpMxV), $y = y + Ax$ where A is a sparse $(N \times N)$-matrix, x and y are dense N-vector, is an important kernel which appears in many scientific fields. Since parallel computers have recently become popular, most SpMxV computations have been executed on parallel environments. The computation time of SpMxV often accounts for significant part of the total application time. Therefore, fast SpMxV routine is important for high performance computing field. This paper proposes a parallel SpMxV routine in which aggressive optimizations are exploited. The target machines range over distributed memory parallel machines including clusters. SMPs are also considered via SPMD programming model. Each node is assumed to have memory hierarchy of cache and memory.

In local computation part on each node, our routine uses blocking method in order to efficiently exploit higher level memory. In addition to conventional two blocking methods for SpMxV, a new blocking method which generalizes

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2659, pp. 581–591, 2003.

those two methods is proposed. This proposed blocking method is applied to the parallel environment.

There are several parameters that affect the performance of blocked parallel SpMxV computation. The optimal parameter set highly depends on the characteristics of target matrix and machine. Therefore, selecting the optimal parameter set according to these characteristics is important. The optimal parameter set is automatically selected at the run time according to the characteristics of target matrix and machine. Therefore, our routine can achieve high performance on various computational environments.

2 Experimental Environments

The features of 4 parallel machines used in the experiment are shown in Table 1. C language is used as a programming language. For communication library, MPI library is used. The implementation of MPI library is MPICH [1] ver 1.2.1.

12 test sparse matrices are shown in Table 2. They are collected from Tim Davis' Sparse Matrix Collection [2] except the dense matrix of No. 12. Column 'Size' is the dimension of the matrix. All matrices are square matrix. Column 'Nonzeros' is the number of non-zero elements in the matrix. Column 'Density' is the percentage of existence of non-zero elements.

3 Local Computation Optimization

A blocking method is a popular optimization technique for SpMxV on machines that have memory hierarchy. Non-zero elements in sparse matrices from real-world scientific fields are not randomly scattered, but often exist in dense blocks [3]. By blocking such dense blocks, high performance of SpMxV can be achieved. This blocking method is called as 'Register Blocking' [4], because its main focus is efficient usage of the register memory. The blocking shape is generally rectangular. Since most of dense blocks in sparse matrices are small, a block size

Table 1. Machine's architectures

Processor	Intel Pentium III	SUN Ultra Sparc II	SGI MIPS R12000	Intel Itanium
Clock	800 MHz	333 MHz	350 MHz	800 MHz
# of PEs	8 (dual × 4)	8	8	8
Network	100 base-T Ethernet	SMP	DSM	SMP
Compiler	Portland Group's C compiler	SUN WorkShop Compiler	MIPSpro Compiler	Intel Itanium
Compiler Version	3.2-3	5.0	7.30	5.0.1
Compiler Option	-fast -tp p6 -Mdalign -Mvect= assoc,prefetch,nosse	-xO5 -dalign -xtarget=native -xarch=v8plusa -xrestrict=all	-Ofast=ip27 -TARG:proc=r10k -TARG:isa=mips4 -r12000 -64	-O3

Table 2. Test matrix set

No.	Name	Description	Size	Nonzeros	Density
1	pre2	Harmonic balance method	659033	5,959,282	0.00137
2	cfd2	CFD, symmetric pressure matrix	123440	3,087,898	0.0203
3	pwtk	Stiff matrix	217918	11,634,424	0.0245
4	gearbox	Aircraft flap actuator	153746	9,080,404	0.0384
5	venkat01	Flow simulation	62424	1,717,792	0.0441
6	nasasrb	Structural engineering	54870	2,677,324	0.0889
7	ct20stif	CT20 engine block	52329	2,698,463	0.0985
8	bcsstk32	Stiff matrix for automobile	44609	2,014,701	0.101
9	gupta2	Linear programming matrix	62064	4,248,286	0.110
10	3dtube	3D pressure tube	45330	3,213,618	0.156
11	gupta3	Linear programming matrix	16783	9,323,427	3.31
12	dense	Dense matrix	3162	9,998,244	100

should also be small, such as 2×2 or 3×3. By blocking in such a small size block, a usage of the highest level memory, register memory, can be optimized, and the number of access to the column index array is reduced.

The access order to matrix data in blocked SpMxV computation changes from non-blocked SpMxV. Therefore, the data format of sparse matrix is reconstructed from Compressed Row Storage (CRS) format to Block Compressed Row Storage (BCRS) format [5]. Array elements of non-zero values are reordered to the accessing order, and column index array is also reconstructed.

There have been used two types of register blocking method so far. The difference of these two methods is that whether zero elements in the original sparse matrix can enter the block of the blocked matrix or they can not.

One of the conventional methods which does not allow zero elements entering into the blocked matrix (we call this method as type 1 method) splits the original matrix into two matrices, blocked matrix A_{block} and remaining matrix A_{remain} [6,7,8] (See the left of Fig. 1). We start searching blocks from the top-left of the matrix to the right-bottom direction. If a dense block larger than the block size $r \times c$ is found, it is stored into the blocked matrix A_{block}. The dense block smaller than the block size is not blocked and stored into the remaining matrix A_{remain}. The data storage format of A_{remain} is CRS format. In this blocking method, the

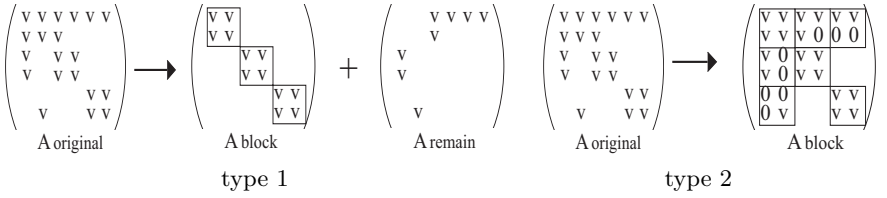


Fig. 1. The example of conventional two blocking methods (2×2)

total number of the non-zero elements is the same before and after the blocking. Therefore, the number of floating-point number operations is also the same. However, since the original matrix is split into two matrices, SpMxV operation is computed twice. The performance may decline by overhead of two matrices computation. Since blocks can not contain zero elements, an opportunity of blocking many non-zero elements may be missed.

The other method of conventional blocking (type 2 blocking) is that even if there are dense blocks in the original matrix whose size is smaller than block size $r \times c$, the elements are stored in the blocked matrix with padding zero elements [4,9]. The original matrix is split into logical $r \times c$ blocks. If there is at least one element in a logical block, that block is stored into the blocked matrix with padding zero elements. If zero elements in the original matrix are blocked, these zero elements are treated as non-zero elements in the blocked matrix (See the right of Fig. 1). All the non-zero elements are blocked in this manner. The original matrix is not split unlike the type 1 blocking. However, the number of non-zero elements may increase. Therefore, the number of floating-point operations may also increase, and it can cause performance decline.

3.1 The New Blocking Method

Assume that the original matrix is blocked with block size $r \times c$. The type 1 blocking method makes block if there are at least rc elements in a block. And the type 2 blocking method makes block if there is at least 1 element in a block. We can easily find that the both methods are the parts of a general method that if there are at least t elements in a block, that block is stored into the blocked matrix. t is the threshold integer number. This generalized method is our proposal method. This method uses the threshold number t, if the number of non-zero elements in a block is larger than t, the block is stored into the blocked matrix with padding zero elements if necessary. And if the number of non-zero elements in a block is smaller than t, the block is stored into remaining matrix in CRS format. This method does not miss the non-zero elements to be blocked unlike the type 1 blocking, and does not increase a lot of non-zero elements unlike the type 2 blocking. The example of blocking by this method with block size 2×2 and threshold 2 is shown in Fig. 2.

Next, we show the effect of our new blocking method. The performance of non-blocked SpMxV, that of conventional two methods and that of our new

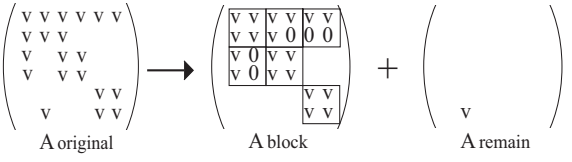


Fig. 2. The example of proposed blocking method $(2 \times 2, 2)$

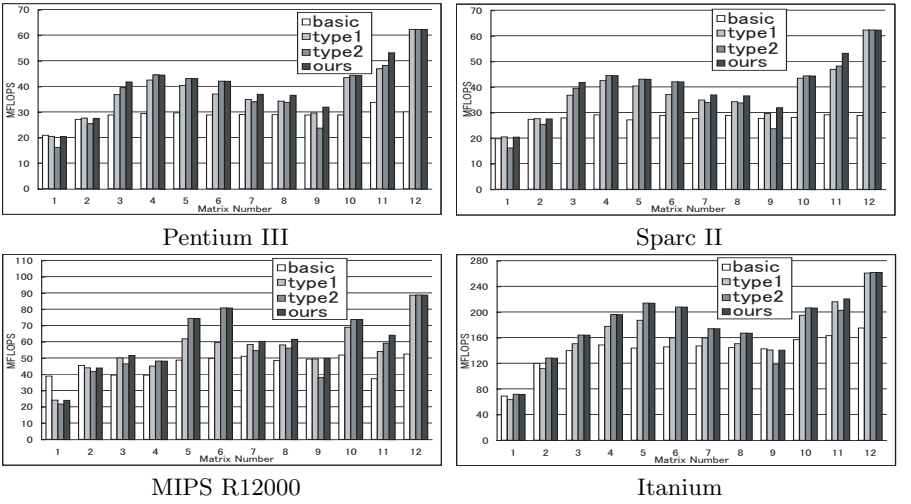


Fig. 3. The performance of four types of SpMxV

method are shown in Fig. 3. The block sizes used in the figure are the best block sizes that are selected by exhaustive search. The block sizes used in this paper are from 1×2 to 8×8. From the result, it can not be said that one of the conventional two methods are better for all cases. There are cases that the difference of performance of type 1 and type 2 is more than 20 %. Therefore, it can be said that fixing the blocking method to either of conventional two methods is not a good idea. On the other hand, since our new method includes type 1 and type 2, the performance of our new method is at least the same performance of conventional two methods. In some cases, its performance is better than both of conventional methods.

3.2 Selection of the Optimal Blocking Parameter Set

There are 3 parameters for our new blocking method, row size r, column size c and threshold t. To achieve high performance, the optimal parameter set (r, c, t) should be used. t can take from 1 to rc for the block size $r \times c$. The number of combinations which the parameter set (r, c, t) can take is 1926 for the block sizes from 1×2 to 8×8. One of the selecting methods to find the best parameter set is measuring the performance exhaustively for all parameter sets. As stated before, the matrix data have to be reconstructed from CRS format to BCRS format before the series of computation of blocked SpMxV.

The reconstruction operation takes about 10 times of non-blocked SpMxV computation time in this study. Therefore, it takes more than 10,000 times of non-blocked SpMxV computation time for exhaustive search. Since it is not practical, the performance of the blocked SpMxV for a given parameter has to be estimated without matrix data reconstruction.

3.3 The Blocking Parameter Selection Method

In this section, we propose a parameter selection method with performance estimation for our new blocking method. First, we define the performance estimation formula. The formula calculates the estimated performance with parameter set (r, c, t), $P(r, c, t)$. On the installation time of our routine, three types of performance information have to be measured, performance of dense matrix vector multiplication for each block size ($Pd_{r,c}$ (MHz)), performance of non-blocked dense matrix vector multiplication (Pd_{nb} (MHz)) and the sum of read and write time to one element of floating-point array (Tac). Next, when the target sparse matrix is given, the number of elements stored into the blocked matrix ($Nb_{r,c,t}$) and the number of elements stored into the remaining matrix ($Nr_{r,c,t}$) are counted.

If we use performance of blocked matrix as $Pd_{r,c}$ and performance of remaining matrix as Pd_{nb}, the whole performance of blocked SpMxV with parameter set (r, c, t) is estimated as follows,

$$P(r,c,t) = \begin{cases} \frac{nnz}{\frac{Nb_{r,c,t}}{Pd_{r,c}} + M \cdot Tac}, & t = 1 \\ \frac{nnz}{\frac{Nb_{r,c,t}}{Pd_{r,c}} + \frac{Nr_{r,c,t}}{Pd_{nb}} + 2M \cdot Tac}, & t > 1 \end{cases} \tag{1}$$

where M is row size of matrix and nnz is the number of non-zero elements in the original matrix. $M \cdot Tac$ means the access time to the destination vector, y. Three variables in (1), $Pd_{r,c}, Pd_{nb}$ and Tac have to be measured only once at the installation time of the routine. However, $Nb_{r,c,t}$ and $Nr_{r,c,t}$ have to be counted at the run-time after the target sparse matrix is given. Therefore, it is important to quickly count these two variables. Next, we explain the fast counting method of these two variables.

The original matrix is logically split into the $r \times c$ size blocks. Then, the number of non-zero elements in each block is counted. Let $nnzb$ be this value. $nnzb$ is counted for all the logical blocks, and then the number of blocks for each $nnzb$ is counted. That is, we collect the information that the number of blocks whose $nnzb = 1$ is a and the number of blocks whose $nnzb = 2$ is b, etc. Let $b_{r,c,i}$ be the number of blocks whose $nnzb = i$ (block size is $(r \times c)$). The two variables $Nb_{r,c,t}$ and $Nr_{r,c,t}$ in (1) are calculated by the following equation, $Nb_{r,c,t} = \sum_{i=t}^{rc} b_{r,c,i} rc, \quad Nr_{r,c,t} = \sum_{i=1}^{t-1} b_{r,c,i} i$. By using this counting method, if we count $b_{r,c,i}$ once for a block size, $Nb_{r,c,t}$ and $Nr_{r,c,t}$ of all the threshold values for the block size can be quickly calculated by small number of integer instruction.

$P(r, c, t)$ for all the combination of (r, c, t) is calculated in this manner, and the parameter set (r, c, t) with which $P(r, c, t)$ is best is selected as the optimal parameter set.

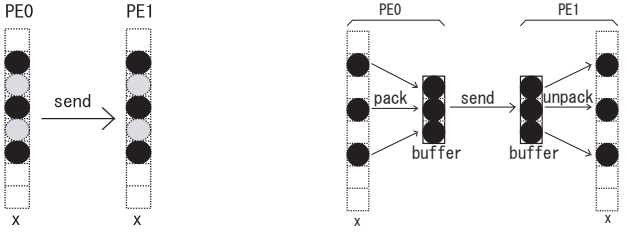
4 Data Communication Optimization

The matrix and vector data are distributed in the row block distribution manner so that each processor has nearly the equal number of non-zero elements. On

parallel SpMxV computation, vector elements that reside on other processors may be required. In such case, communication of vector data between processors is necessary. Communication time becomes more dominant as the number of processors becomes larger. Therefore it is important for parallel SpMxV operation to reduce the communication time. In this paper, two communication methods are used; 'range limited communication' and 'minimum data size communication'. It is dependent on the non-zero structure of target matrix and machine's architecture which method is fast.

Range Limited Communication: By this communication method, processors send only minimum contiguous required block (See the left of Fig. 4). Communications between processors that are not required are not sent. Since unnecessary elements are often contained in the block, the communication data size is not minimum on most cases. However, since this communication method does not need local copy operation, the overhead time is small.

Minimum Data Size Communication: This method communicates only elements that have to be sent to other processors. (See the right of Fig. 4). Therefore the communication data size is minimum. In order to complete communication within two processors by one message passing, we need pack and unpack operations, because the elements to be sent are not placed consecutively in the source vector. These pack and unpack operations require a little overhead time.



Range limited communication Minimum data size communication

Fig. 4. The range limited communication and the minimum data size communication The black circles in PE0 represents the source vector elements that must be sent to PE1. The gray circles are the dummy data that does not need to be sent to PE1 originally, however, these elements also have to be sent in range limited communication.

4.1 Implementation of Communication Method

The range limited communication and minimum data size communication use one to one communication. In this study, three implementation types of one to one communication are used, 'Send-Recv', 'Isend-Irecv' and 'Irecv-Isend'. 'Send-Recv' uses MPI_Send and MPI_Recv functions. 'Isend-Irecv' uses non-blocking communication (MPI_Isend and MPI_Irecv). All the communication types implemented in this study are summarized in Table 3.

Table 3. The communication Methods

Method	Explanation
sendrecv R	Range limited comm. with MPI_Send and MPI_Recv
isendrecv R	Range limited comm. with non-blocking communication in send-receive order
irecvsend R	Range limited comm. with non-blocking communication in receive-send order
sendrecv M	Minimum data size comm. with MPI_Send and MPI_Recv
isendrecv M	Minimum data size comm. with non-blocking communication in send-receive order
irecvsend M	Minimum data size comm. with non-blocking communication in receive-send order

4.2 Application of New Blocking to the Parallel Environment

On single processor environment, blocking parameter set is selected by performance estimation formula described in Sect. 3.3. This parameter selection method is applied to parallel environment. The block size is selected by performance estimation concurrently on each processor. That is, a processor selects the block size with only information of the matrix data that is distributed to the processor. Therefore, data communication is unnecessary on the blocking parameter selection time. Because the estimated blocking parameter can differ on each processor, each processor may select different blocking parameter set.

5 Numerical Experiment

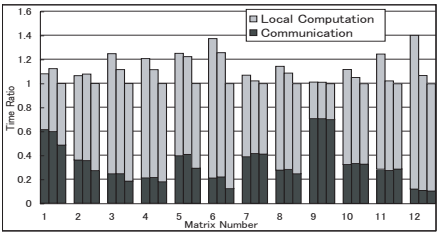
5.1 The Dynamic Parameter Selection Method

We have implemented parallel SpMxV routine which has optimization techniques on both local computation part and data communication part. It has four parameters that affect the performance of parallel SpMxV, *the row size of the block, the column size of the block, the threshold number and the data communication method.* The parameter set is selected automatically at the run time according to the characteristics of matrix and machine in the following order, **1.** Select the communication method, **2.** Select the blocking parameter set. First, all the communication methods in Table 3 are executed and their time is measured. The fastest communication method is selected as the optimal method. Next, the blocking parameter set is selected by the selection method described in Sect. 3.3.

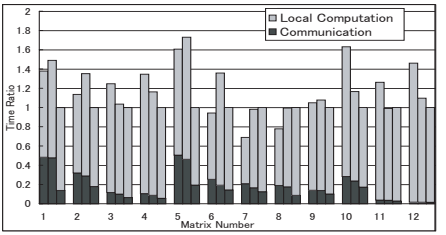
5.2 The Performance of Our Parallel SpMxV Library

To show the performance of our parallel SpMxV routine, the experimental results are shown in this section. The performance with dynamically selected parameter set is compared with the performance of fixed parameter sets that is generally used for all matrices and machines. The two fixed parameter sets used in this experiment are (No-blocking and isendrecv-M) and (2×2, 4 and isendrecv-M).

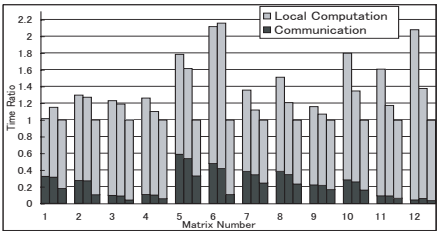
The computation time of parallel SpMxV is shown in Fig. 5.2. The time in the figure is an average time of 8 processors. There are 3 bars for each matrix. The left 2 bars are the time of fixed parameter sets. The right most bar is the time of our routine of which parameter set is dynamicaly selected. All bars are scaled so that the time of our routine becomes to 1.0. The light part of bar is the time



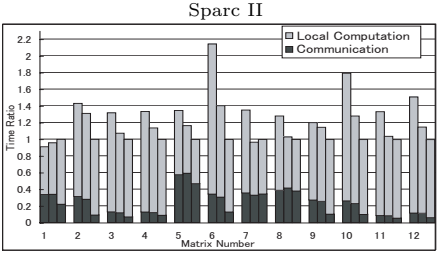
Pentium III



Sparc II



MIPS R12000



Itanium

Pentium III

No.	Comm. MFLOPS	Block sizes $r \times c\,(t)$
2	irecvsend M	2×1(2), 2×1(2), 2×1(2), 2×1(2)
	216.56	2×1(2), 2×1(2), 2×1(2), 2×1(2)
4	irecvsend M	3×3(1), 5×2(8), 5×2(8), 5×2(8)
	354.18	5×2(8), 3×3(1), 5×2(8), 5×2(8)
6	isendrecv R	6×6(1), 2×6(1), 3×6(14), 2×6(1)
	398.72	3×6(1), 3×6(14), 5×2(8), 2×6(1)
8	irecvsend M	6×4(17), 5×2(8), 5×2(8), 5×2(8)
	296.54	5×2(8), 5×2(8), 5×2(8), 5×2(8)
10	irecvsend M	3×6(14), 3×3(1), 4×1(4), 5×2(8)
	277.83	5×2(8), 3×3(1), 5×2(8), 5×2(8)

Sparc II

No.	Comm. MFLOPS	Block sizes $r \times c\,(t)$
2	irecvsend M	2×2(3), 2×1(2), 2×1(2), 2×2(3)
	133.60	2×2(3), 2×2(3), 2×2(3), 2×2(3)
4	irecvsend M	3×3(1), 2×3(4), 2×3(4), 2×3(4)
	103.37	2×3(4), 3×3(1), 2×3(4), 2×3(4)
6	irecvsend M	6×1(1), 2×3(4), 6×1(4), 2×3(4)
	235.25	3×6(1), 6×1(4), 6×1(4), 5×2(6)
8	irecvsend M	6×1(4), 6×1(4), 6×1(4), 6×1(4)
	207.35	8×1(4), 6×1(4), 8×1(4), 6×1(4)
10	irecvsend M	3×3(6), 3×3(1), 4×1(3), 6×1(4)
	272.05	6×1(4), 3×3(1), 6×1(4), 6×1(4)

MIPS R12000

No.	Comm. MFLOPS	Block sizes $r \times c\,(t)$
2	irecvsend R	2×2(4), 2×1(2), 2×1(2), 2×1(2)
	338.55	2×1(2), 2×1(2), 2×1(2), 2×2(4)
4	irecvsend M	3×3(1), 4×2(5), 4×2(5), 4×2(5)
	304.91	4×2(5), 3×3(1), 4×2(5), 4×2(5)
6	irecvsend R	6×3(1), 2×6(1), 6×3(11), 2×6(1)
	1115.22	3×6(1), 6×3(11), 6×3(11), 2×6(1)
8	irecvsend M	6×3(11), 4×2(5), 4×2(5), 4×2(5)
	791.08	4×2(5), 4×2(5), 4×2(5), 4×2(5)
10	isendrecv R	3×3(1), 3×3(1), 4×2(5), 4×2(5)
	636.33	4×2(5), 3×3(1), 4×2(5), 4×2(5)

Itanium

No.	Comm. MFLOPS	Block sizes $r \times c\,(t)$
2	isendrecv R	2×2(3), 1×2(2), 1×2(2), 2×1(2)
	470.03	2×1(2), 2×1(2), 2×1(2), 2×2(3)
4	sendrecv M	3×3(1), 1×3(1), 1×3(1), 1×3(1)
	393.46	1×3(1), 3×3(1), 1×3(1), 1×3(1)
6	sendrecv R	6×6(1), 2×6(1), 1×6(1), 2×6(1)
	947.61	3×6(1), 1×3(1), 1×6(1), 2×6(1)
8	sendrecv M	2×2(1), 4×2(6), 4×2(6), 4×2(6)
	735.68	3×4(8), 4×2(6), 4×2(6), 4×2(6)
10	isendrecv R	3×3(1), 3×3(1), 2×3(1), 1×3(1)
	699.75	1×3(1), 3×3(1), 1×3(1), 1×3(1)

Fig. 5. The performance of parallel SpMxV and selected parameters. The graph shows time of parallel SpMxV with each parameter set. All bars are scaled so that the time with auto-tuned parameter set becomes to 1.0. There are 3 bars for each matrix. The left most bar shows time with parameter set (No-blocking and isendrecv-M). The middle bar shows time with parameter set (2×2, 4 and isendrecv-M). The right most bar shows time with auto-tuned parameter set. The table shows the selected parameter set which is selected by auto-tuning system

of local computation. The dark part of bar is the time of data communication. Parameter sets selected by the dynamic parameter selection method for matrix nos. 2, 4, 6, 8 and 10 are also shown in Fig. 5. From the result, we can see that the performance of our library is the best in 41 cases among the 48 cases.

The cost of parameter selection is evaluated as the ratio of parameter selection time to the basic parallel SpMxV time that uses a non-optimized parameter set (MPI_Allgather communication and Non-blocking SpMxV kernel). The parameter selection cost is about 40-100 times of basic parallel SpMxV time in this experiment.

6 Concluding Remarks

We have developed a parallel sparse matrix-vector multiplication (SpMxV) routine. The local computation on each node uses a new blocking method which generalizes conventional two blocking methods in order to efficiently exploit higher level memory. From the experimental result, it has been shown that our new blocking method is equal or better compared with the conventional two methods. On data communication part, six types of data communication methods have been implemented. The parameter set such as blocking size and data communication type is selected automatically by dynamic parameter selection method at the run time according to the characteristics of matrix and machine.

The performance of our parallel SpMxV routine with dynamically selected parameter set has been compared with the performance with fixed parameter sets that are generally used for all machines and matrices. The experimental result has shown that the performance of our routine is the fastest in most cases. Therefore, it can be said that the dynamic parameter selection method works well and our parallel SpMxV routine is high performance for various kinds of sparse matrices and on many types of computers.

As future work, it would be important to develop the faster and more accurate parameter selection method.

References

1. William, G., Lusk, E.: (User's guide for mpich, a portable implementation of mpi)
2. Davis, T.: University of florida sparse matrix collection (1997) NA Digest, vol. 97., Jun 1997. `http://www.cise.ufl.edu/~davis/sparse/`.
3. White, J., Sadayappan, P.: On improving the performance of parallel sparse matrix-vector multiplication. In: Proceedings of Supercomputing'97, San Jose, CA (1997)
4. Im, E.: Optimizing the Performance of Sparse Matrix - Vector Multiplication. PhD thesis, University of California at Berkeley (2000)
5. Barrett, R., Berry, M., Chan, T.F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., der Vorst, H.V.: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (1994)

6. Geus, S.R.R.: Towards a fast parallel sparse matrix-vector multiplication. In: Parallel Computing: Fundamentals & Applications, Proceedings of the International Conference ParCo'99, 17–20 August 1999, Delft, The Netherlands. (2000) 308–315
7. Pinar, A., Heath, M.T.: Improving performance of sparse matrix-vector multiplication. In: Proceedings of Supercomputing'99. (1999)
8. Toledo, S.: Improving the memory-system performance of sparse-matrix vector multiplication. IBM Journal of Research and Development **41** (1997) 711–725
9. Vuduc, R., Demmel, J.W., Yelick, K.A., Kamil, S., Nishtala, R., Lee, B.: Performance optimizations and bounds for sparse matrix-vector multiply. In: Proceedings of Supercomputing 2002. (2002)