THREADED ACCURATE MATRIX-MATRIX MULTIPLICATIONS WITH SPARSE MATRIX-VECTOR MULTIPLICATIONS

Shuntaro Ichimura(1), Takahiro Katagiri(2), Katsuhisa Ozaki(3), Takeshi Ogita(4), Toru Nagai(2)

(1) Graduate School of Information Science, Nagoya University
(2) Information Technology Center, Nagoya University
(3) College of System Engineering and Science, Shibaura Institute of Technology
(4) Division of Mathematical Science, Tokyo Woman's Christian University

The Thirteenth International Workshop on Automatic Performance Tuning (iWAPT2018), May 25, 2018, JW Marriott Parq Vancouver, Vancouver, British Columbia CANADA, AT Techniques, 10:30 - 12:30, May 25, 2018

Outline

- Background
- Accurate precision MMM (Ozaki Method)
- Parallel Implementation for Multi-core CPUs and Its Evaluation
- Conclusion

Outline

- Background
- Accurate precision MMM (Ozaki Method)
- Parallel Implementation for Multi-core CPUs and Its Evaluation
- Conclusion

Background

- Libraries for basic linear algebra operations, such as **BLAS (Basic Linear Algebra Subprograms), are one of crucial tools** for numerical computations.
- Generally speaking, **accuracy assurance for numerical linear algebra libraries**, such as LAPACK, **is still under research**.
- On the other hand, study on **accuracy assurance for BLAS operations** is performing by Prof. Oishi group (Waseda University), including Prof. Ogita, and Prof. Ozaki .
- We forces on the research, in particular, **high precision matrix-matrix multiplication (MMM)**.
- Here after, we call the method ***Ozaki Method***.

Outline

- Background
- Accurate precision MMM (Ozaki Method)
- Parallel Implementation for Multi-core CPUs and Its Evaluation
- Conclusion

Overview of High Precision Matrix-Matrix Multiplications (MMM) Algorithm (Ozaki Method †1)（1/3）

A Matrix-Matrix Multiplications A B

→

Summation of Decomposed Matrices with Floating Point Operations

Error-Free Transformation

$$C = AB = \sum_{q=1}^{r} C_q$$

$$C_q \in F^{m \times p}$$

F : A Set of Floating Point Numbers.
A : A Matrix with *m* * *n*.
B : A Matrix with *n* * *p*.
C : *A* * *B*

†1 K. Ozaki, T. Ogita, S. Oishi, S.M. Rump: Error-Free Transformation of Matrix Multiplication by Using Fast Routines of Matrix Multiplication and its Applications, Numerical Algorithms, Vol. 59, No.1, pp.95-118, 2012.

Other Part of The Error Free Transformation in Ozaki Method

- **Part of MMMs**

n_A : The number of decomposed matrices from matrix A.

n_B : The number of decomposed matrices from matrix B.

```
Function EF = EFT_Mul(A, B )
    [A, n_A] := Split_A;  [B, n_B] := Split_B;
      k := 1;
    for i =1: n_A
        for j =1: n_B
            EF{ k } := A{ i } * B{ j };    k := k + 1;
    end;  end;
end
```

Multiple BLAS implementation

- A High Precision Summation: $AB = \sum_{k=1}^{n_A \cdot n_B} EF^{(k)}$ ⇒ Faithful Algorithm

Faithful Algorithm†

Round-off the true answer to
the nearest left or right floating number.

Ozaki Method for MMM

True Answer (Real Number)

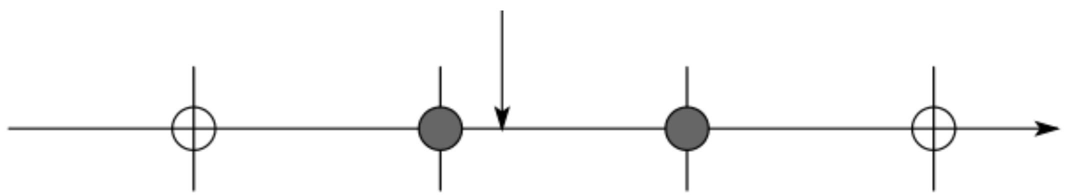


Figure 1: faithful rounding

Accuracy Assured

†Siegfried M. Rump, Takeshi Ogita, Shin'ichi Oishi: *Accurate Floating-Point Summation Part I: faithful Rounding*, SIAM Journal on Scientific Computing, **31**:1 (2008), 189-224.

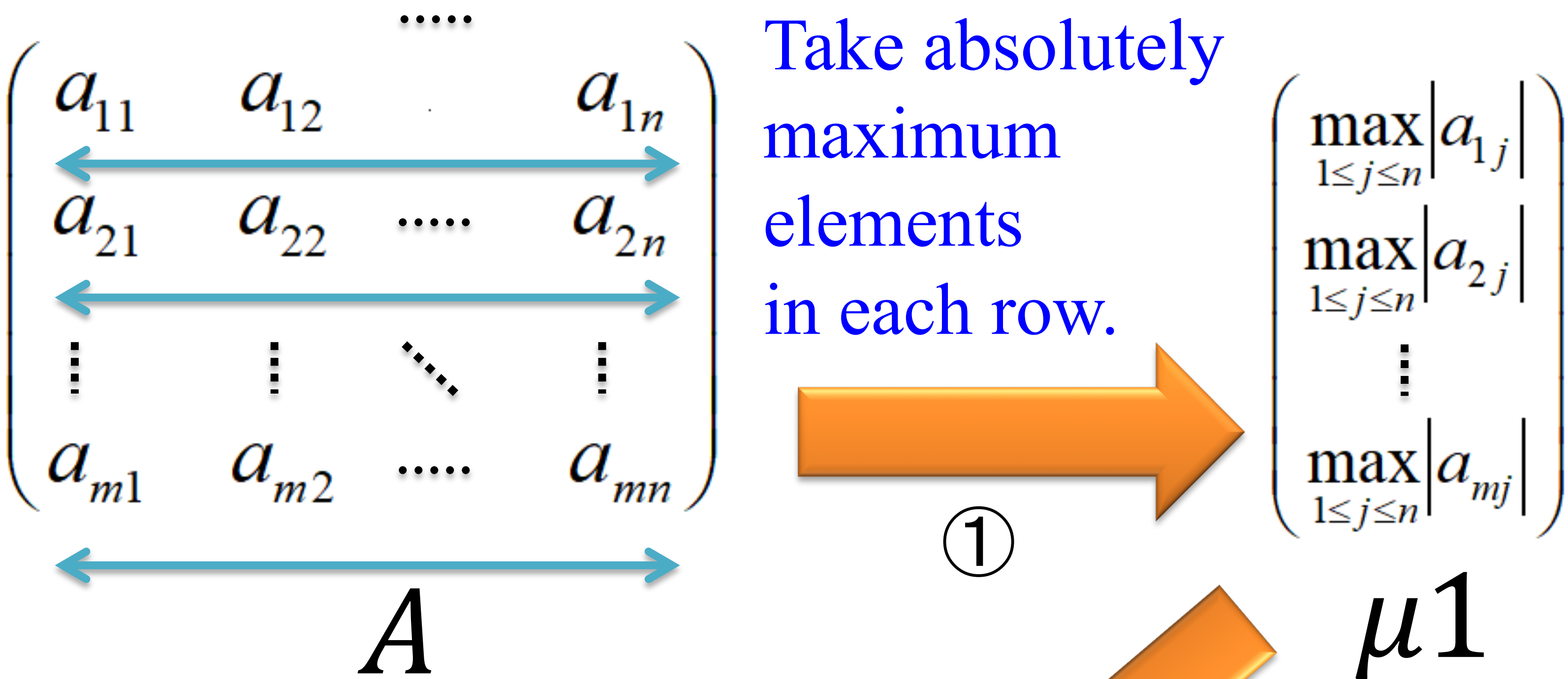
Characteristics of Ozaki Method

- Ozaki method can establish high precision for MMM with extremely dispersed elements.
- Computational complexity of Ozaki method depends on range of input elements.

(1) If dispersion of elements of matrix is large:
Sparse matrix can be utilized after error free translation to reduce computational complexity.

(2) If dispersion of elements of matrix is small:
Cannot reduce computational complexity.
But, Conventional high performance implementations (BLAS dgemm) of dense MMM can be utilized.

Error-Free Transformation (1/3)



- $\mu 1 \;=\; \max(abs(A), [\,], 2)\,;$
- $\tau \;=\; 2^{ceil(\frac{(\log_2 u^{-1} + \log_2 (n+1))}{2})};$

$$t_A = 2^{ceil(\log_2(\mu 1))}\;\tau$$

Take maximum elements of products in each column.

* $ceil()$: Compute minimum integer number

Error-Free Transformation (2/3)

③

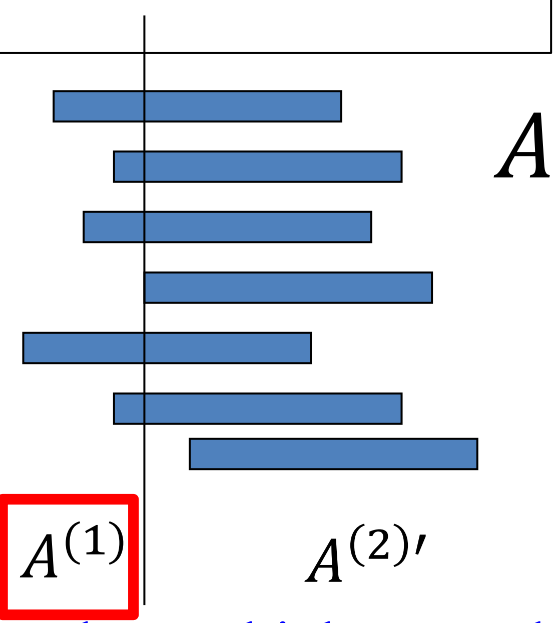
- Make T as:

$$T = [t_A, t_A, \ldots, t_A],$$

where, $T_{ij} > A_{ij}$.

Maximum number of products in each column.

$$T = \begin{pmatrix} t_A & t_A & \cdots & t_A \end{pmatrix}$$



fl (*) : A Floating Point Computation

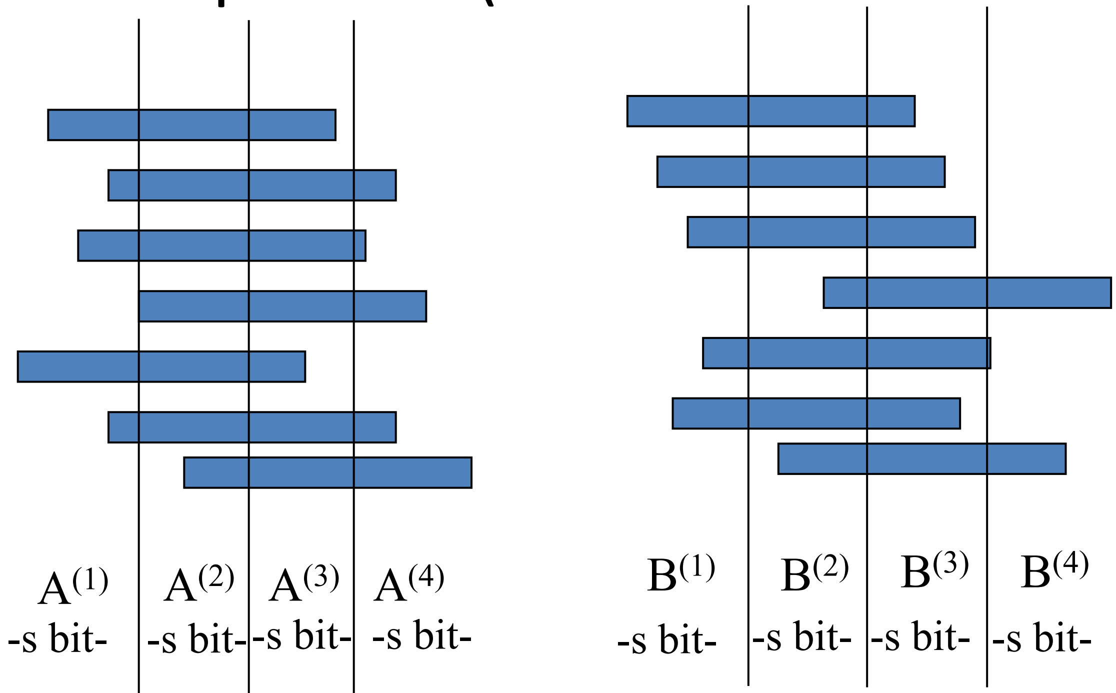
④

- $A^{(1)} = fl((A + T) - T);$
- $A^{(2)\prime} = fl(A - A^{(1)});$

Extract values which exceed range of expression of products with respect to round-off error.

Error-Free Transformation (3/3)

- An image of decomposition (Error free transformation)



Matrix size = n x n

$$s = floor((\log_2(u^{-1}) - \log_2(n))/2)$$ [bit]

Ex.) If *double precision*, then it should take:

$floor((53 - \log_2(n))/2)$ [bit],

And if matrix size is $n = 1024$, then

it should take $floor\left(\frac{53-10}{2}\right) = 21$ [bit].

floor : under rounding for the first digit of floating point number.

An Example

We take the following floating point numbers.

$$A = \begin{bmatrix} 2^0 + 2^{-30} & 2^{-20} + 2^{-40} & 2^{-10} + 2^{-15} \\ 2^{-10} + 2^{-30} & 2^{-15} + 2^{-35} & 2^1 + 2^{-15} \\ 2^{-5} + 2^{-25} & 2^0 + 2^{-30} & 2^5 + 2^{-30} \end{bmatrix}$$

$$B = \begin{bmatrix} 2^0 + 2^{-30} & 2^{-35} + 2^{-60} & 2^5 + 2^{-15} \\ 2^{-5} + 2^{-10} & 2^{-30} + 2^{-40} & 2^{-10} + 2^{-30} \\ 2^{-3} + 2^{-15} & 2^{-40} + 2^{-50} & 2^{-8} + 2^{-17} \end{bmatrix}$$

- Considering an inner product for 1st row of A and 1st column of B. We compute products of $A(1,1)$ and $B(1,1)$ as follows.

$$\underline{(\mathbf{2^0}+\mathbf{2^{-30}})(\mathbf{2^0}+\mathbf{2^{-30}}) = \mathbf{2^0} + \mathbf{2^{-29}} + \mathbf{2^{-60}}}$$

A rounding error is occurred in the red part with 53-bit.

- Information over 53-bit is dropped in the other part as same as the above.

Error-free Transformations for A

We separate the matrix with error-free transformations:

- $A = \begin{bmatrix} 2^{0} + 2^{-30} & 2^{-20} + 2^{-40} & 2^{-10} + 2^{-15} \\ 2^{-10} + 2^{-30} & 2^{-15} + 2^{-35} & 2^{1} + 2^{-15} \\ 2^{-5} + 2^{-25} & 2^{0} + 2^{-30} & 2^{5} + 2^{-30} \end{bmatrix}$

to the following two matrices.

- $A^{(1)} = \begin{bmatrix} 2^{0} & 2^{-20} & 2^{-10} + 2^{-15} \\ 2^{-10} & 2^{-15} & 2^{1} + 2^{-15} \\ 2^{-5} & 2^{0} & 2^{5} \end{bmatrix}$
- $A^{(2)} = \begin{bmatrix} 2^{-30} & 2^{-40} & 0 \\ 2^{-30} & 2^{-35} & 0 \\ 2^{-25} & 2^{-30} & 2^{-30} \end{bmatrix}$

Error-free Transformations for B

We separate with error-free transformations:

- $B = \begin{bmatrix} 2^{0} + 2^{-30} & 2^{-35} + 2^{-60} & 2^{5} + 2^{-15} \\ 2^{-5} + 2^{-10} & 2^{-30} + 2^{-40} & 2^{-10} + 2^{-30} \\ 2^{-3} + 2^{-15} & 2^{-40} + 2^{-50} & 2^{-8} + 2^{-17} \end{bmatrix}$

to the following two matrices.

- $B^{(1)} = \begin{bmatrix} 2^{0} & 2^{-35} & 2^{5} + 2^{-15} \\ 2^{-5} + 2^{-10} & 2^{-30} + 2^{-40} & 2^{-10} \\ 2^{-3} + 2^{-15} & 2^{-40} + 2^{-50} & 2^{-8} + 2^{-17} \end{bmatrix}$
- $B^{(2)} = \begin{bmatrix} 2^{-30} & 2^{-60} & 0 \\ 0 & 0 & 2^{-30} \\ 0 & 0 & 0 \end{bmatrix}$

Multiplication with error-free translated matrices for $A^{(1)}$ and $B^{(1)}$

We consider the following MMM:

- $A^{(1)} = \begin{bmatrix} 2^0 & 2^{-20} & 2^{-10}+2^{-15} \\ 2^{-10} & 2^{-15} & 2^1+2^{-15} \\ 2^{-5} & 2^0 & 2^5 \end{bmatrix}$

- $B^{(1)} = \begin{bmatrix} 2^0 & 2^{-35} & 2^5+2^{-15} \\ 2^{-5}+2^{-10} & 2^{-30}+2^{-40} & 2^{-10} \\ 2^{-3}+2^{-15} & 2^{-40}+2^{-50} & 2^{-8}+2^{-17} \end{bmatrix}$

Products of the above MMM, such as products of the third row of $A^{(1)}$ and the first column of $B^{(1)}$ is:

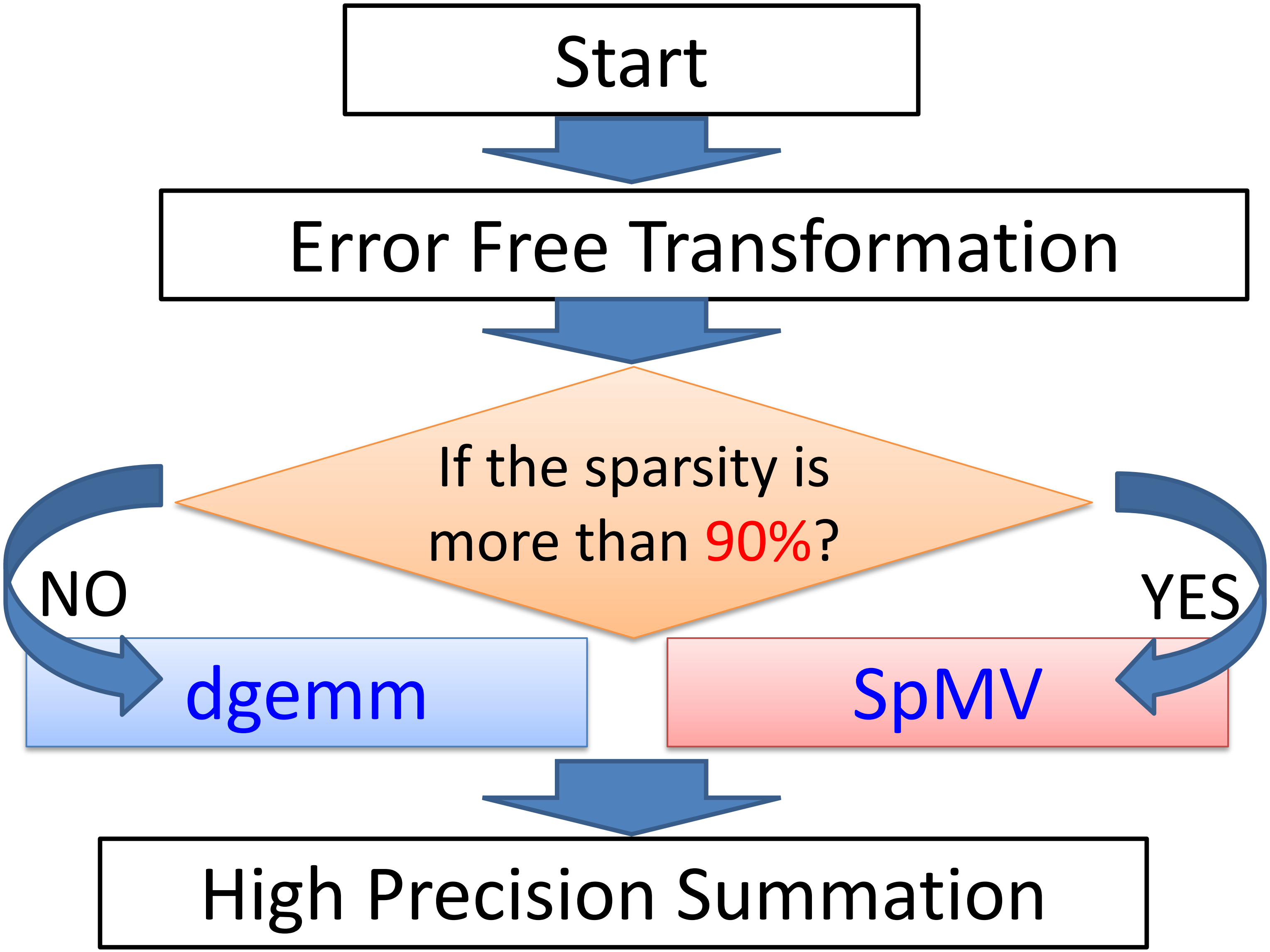
$$\underline{2^{-5} * 2^0 + 2^0(2^{-5}+2^{-10}) + 2^5(2^{-3}+2^{-15}) = 2^2 + 2^{-4} + 2^{-9}.}$$

This is all inside 53-bit computations. Hence there is no rounding error.

Outline

- Background
- Accurate precision MMM (Ozaki Method)
- Parallel Implementation for Multi-core CPUs and Its Evaluation
- Conclusion

Strategy of Using Sparse Matrix for Our Implementation



THREAD PARALLEL IMPLEMENTATIONS

Sparse Matrix Formats for SpMV (1/2)

- CRS Format

Scan non-zero elements for row-wise, and non-zero elements are stored.

$$A = \begin{pmatrix} a_1^1 & b_2^2 & c_3^3 & 0 \\ 0 & 0 & 0 & d_4^4 \\ e_1^5 & 0 & 0 & f_4^6 \\ 0 & 0 & g_3^7 & 0 \end{pmatrix}$$

$$data = [a, b, c, d, e, f, g]$$
$$I(A) = [1, 4, 5, 7, 8]$$
$$J(A) = [1,2,3,4,1,4,3]$$

$data$: An array for store of non-zero elements.
$I(A)$: An array for store of the first position of each row in array $data$.
$J(A)$: An array for store of column numbers corresponding to elements of array $data$.

Sparse Matrix Formats for SpMV (2/2)

- ELL Format

$$A = \begin{pmatrix} a_1^1 & b_2^2 & c_3^3 & 0 \\ 0 & 0 & 0 & d_4^4 \\ e_1^5 & 0 & 0 & f_4^6 \\ 0 & 0 & g_3^7 & 0 \end{pmatrix}$$

Store non-zero elements from left-side without zero elements.

The column number is set to maximum number of columns of non-zero elements. If there is no non-zero elements, then "0" is patted.

$$data = \begin{bmatrix} a & b & c \\ d & 0 & 0 \\ e & f & 0 \\ g & 0 & 0 \end{bmatrix} \quad indices = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 1 & 1 \\ 1 & 4 & 1 \\ 3 & 1 & 1 \end{bmatrix}$$

Proposal method is using ELL format for Ozaki Method.

$$data = [a, b, c, d, 0,0, e, f, 0, g, 0,0]$$

$$num\_col = 3$$

$$J(A) = [1,2,3,4,0,0,1,4,0,3,0,0]$$

Thread-level Parallelization of SpMV in Ozaki Method

1. Inner Parallelization

Thread-level parallelization inside SpMV.

```
for (i = 0; i < n; i++) {
#pragma omp parallel for
  for (j = 0; j < n; j++) {
      (c_i)_j = Sp(A)_j b_i
  }
}
```

$(c_i)\_j$: j-th element of vector c_i.
$Sp(A)\_j$: j-th vector of $Sp(A)$.

Fig1. Inner Parallelized code with OpenMP

2. Outer Parallelization

Thread-level parallelization in multiple calling level of SpMV.
(Using parallelism of columns of *B*)

```
#pragma omp parallel for
for (i = 0; i < n; i++) {
      c_i = Sp(A) b_i
}
```

Fig2. Outer Parallelized code with OpenMP

3. Using Multiple Right-hand-sides

Dedicated inside thread-level parallelism of SpMV with multiple Right-Hand-Sides (RHS).

```
#pragma omp parallel for
for (j=0; j<n; j++) {
  for (i=0; i<n; i+=m) {
      (c_i)_j = Sp(A)_j B_{i:i+m-1}
  }
}
```

$B_{i:i+m-1}$: A Matrix with $b_i, b_{i+1}, \ldots, b_{i+m-1}$

Fig3. Paralleled Code with Multiple Right-hand-sides with OpenMP

$$c_i = \mathrm{Sp}(A) b_i, (i = 1, \ldots, n)$$

c_i: i-th vector of Matrix *C*.
b_i : *i-th* vector of Matrix *B*.
$\mathrm{Sp}(A)$: A sparse matrix from Matrix *A*.

PERFORMANCE EVALUATION

Performance Evaluation

Fujitsu PRIMEHPC FX100 (FX100)@ITC, Nagoya U.	
CPU	SPARC64 XIfx, 2.2 GHz, 32 Cores (+2 Assistant Cores)
Memory Amount	32 GB
Cache Organization	L1: 64KB L2: 24MB
Compiler	Fujitsu C/C++ Compiler Driver Version 2.0.0 P-id: T01776-01
Compiler Options	Sparse Kernel Part: -Kfast –Kopenmp
	The others : -O0 –Kopenmp
Memory Performance	480 GB/s

Test Matrices

1. **(Random)** Elements of matrices *A* and *B* were generated with a pseudo-random generator from the standard uniform distribution on the open interval (0, 1).
2. **(Random * Inverse)** Elements of matrix *A* were generated with a pseudo-random generator from the standard uniform distribution on the open interval (0, 1).
   $B = A^{-1}$ using the dgetrf and dgetri routines in LAPACK.
3. **(Sparse + Dispersed elements)** Elements of matrices *A* and *B* were generated with a pseudo-random generator from the standard uniform distribution on the open interval (0, 1). Then, the elements were selected with a specified ratio (sp_num % to total number of elements), and **we added the selected values with pow(10, rand()%Φ).**
4. **(Sparse + Dispersed elements * Inverse)** Elements of matrix A were generated with 1 for the first row, and then they are added with a pseudo-random generator from the standard uniform distribution on the open interval (0, 1).
   $B = A^{-1}$ using the dgetrf and dgetri routines in LAPACK.

* Φ determines dispersion of elements of matrix.

Condition of Experiments

Implementation Methods

Conventional Ozaki Implementation

Conventional BLAS

	Details	Notation
1	Simple dgemm routine call. This is not accurate MMM.	**simple dgemm**
2	dgemm implementation in the Ozaki method. Thread parallelization is performed inside dgemm.	**Ozaki (dgemm)**
3	Sparse matrix is generated when its sparsity is more than 90%, and SpMV is performed with CRS format. The parallel implementation is inner parallelization.	Ozaki (CRS, Inner)
4	The implementation is the same as that in 3. The parallel implementation is outer parallelization.	Ozaki (CRS, Outer)
5	The implementation is the same as that in 3. The parallel implementation is inner parallelization with multiple vectors of RHS.	Ozaki (CRS, Multi RHS)
6	The implementation is the same as that in 3. The parallel implementation is inner parallelization with multiple vectors of RHS and the blocking factor is 100.	Ozaki (CRS, Multi RHS(100))
7	Sparse matrix is generated when its sparsity is more than 90%, and SpMV is performed with ELL format. The parallel implementation is inner parallelization.	Ozaki (ELL, Inner)
8	The implementation is the same as that in 7. The parallel implementation is outer parallelization.	Ozaki (ELL, Outer)
9	The implementation is the same as that in 7. The parallel implementation is inner parallelization with multiple vectors of RHS.	Ozaki (ELL, Multi RHS)
10	The implementation is the same as that in 3. The parallel implementation is inner parallelization with multiple vectors of RHS and the blocking factor is 100.	Ozaki (ELL, Multi RHS(100))
11	The implementation is with accurate sum for MMM (dot2 [10]). Only dense matrix optimization is performed in the current version.	Inner Products

CRS

ELL

Condition of Experiments

- Matrix Sizes: N=500, and N=1000.
- Φ = 5. (a constant value.)
- Using 1 node. (32 threads)
- **Result is verified with MPFR Library,** which has binary 212 digits in this experiment (almost as same as quad double precision).
- Maximum relative error is calculated with:

$$\max_{1<i,j<n} \left|C^*_{i,j} - C_{i,j}\right| / \left|C^*_{i,j}\right|,$$

$C^*_{i,j}$ means elements of matrix with i –th row and j − th column.

RESULTS

NUMBER OF NON-ZERO ELEMENTS FOR ERROR-FREE TRANSFORMED MATRICES AND NUMBER OF SPARSE MATRICES

Number of Non-zero Elements for Error-free Transformed Matrices for *A* (N=1000)

Let a decomposed matrix of *A* be $A^{(i)}$.

(i =1, 2,…, *p*, where *p* is the number of decompositions for *A*.)

The yellow parts show sparse matrices after error free translation.

Test #	1		2		3		4	
$A^{(i)}$	Max	Min	Max	Min	Max	Min	Max	Min
$A^{(1)}$	1000	1000	1000	1000	1000	996	92	91
$A^{(2)}$	1000	998	1000	998	1000	998	91	90
$A^{(3)}$	86	34	87	40	1000	993	91	85
$A^{(4)}$	-	-	-	-	7	0	-	-

Number of non-zero elements for sparse matrix by error free transformation is small.

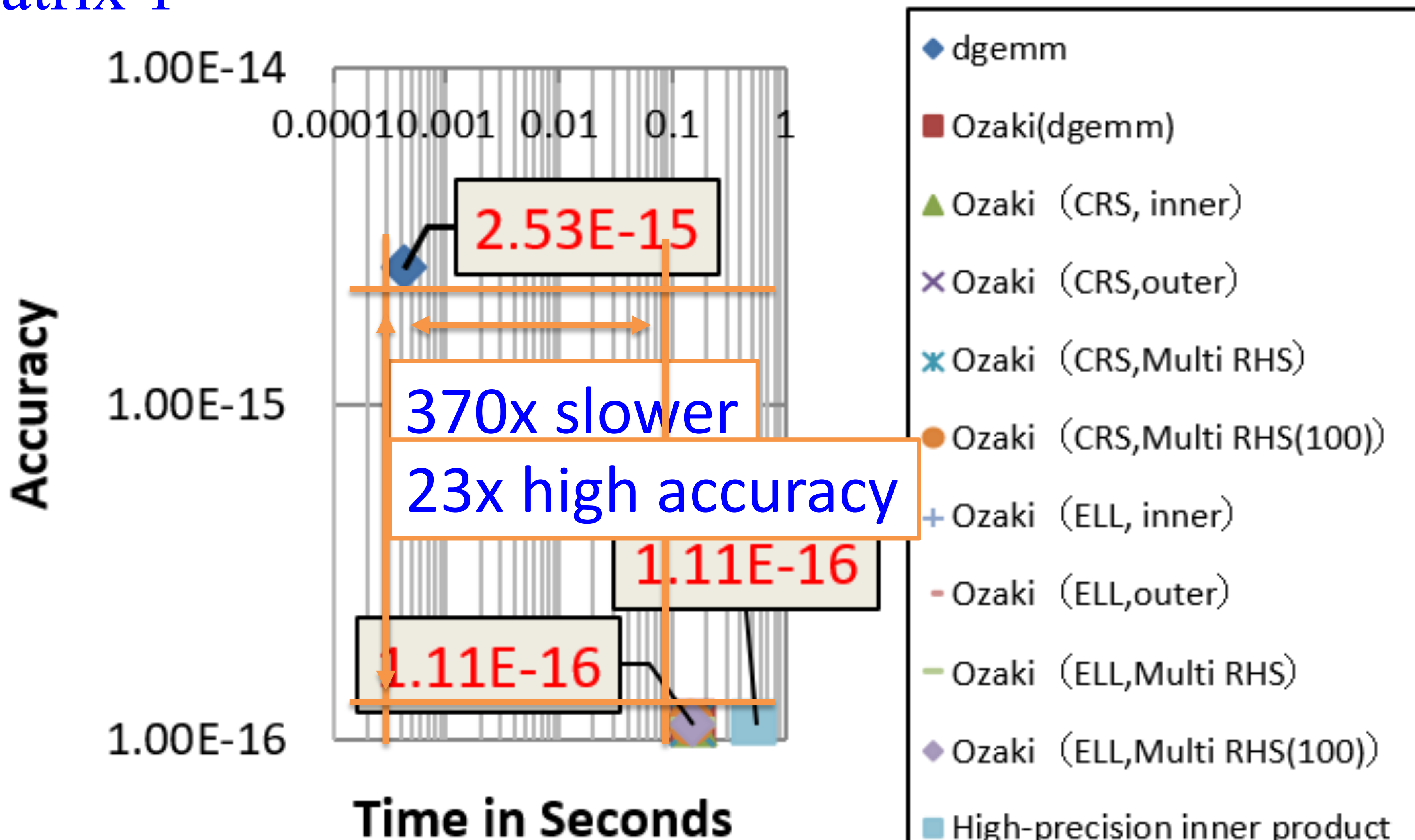
The number of non-zero elements per column is almost constant.

⇒ Computation efficiency is getting high by ELL.

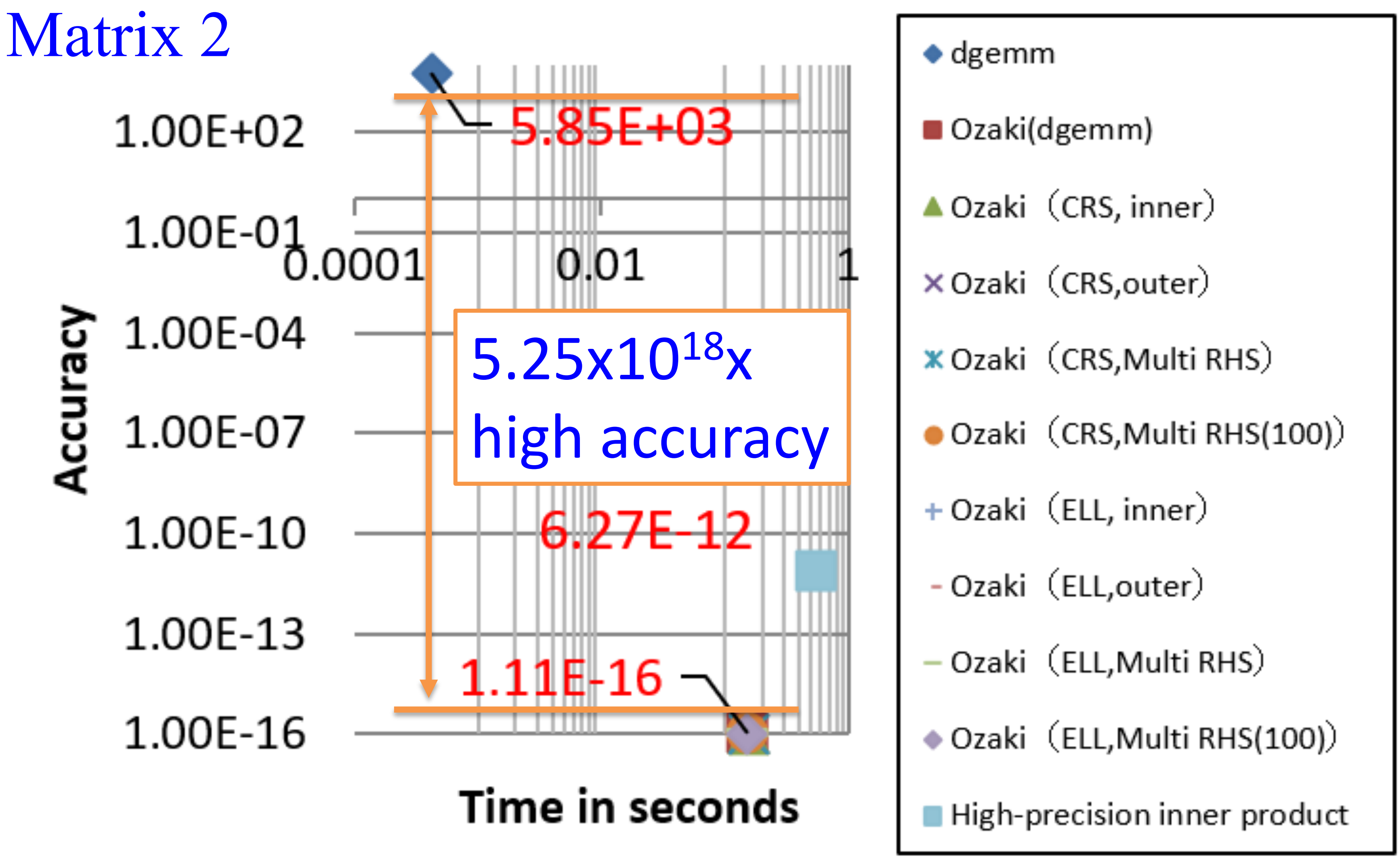
EXECUTION SPEED AND COMPUTATION ACCURACY

Execution speed and computation accuracy (N=500)

Matrix 1

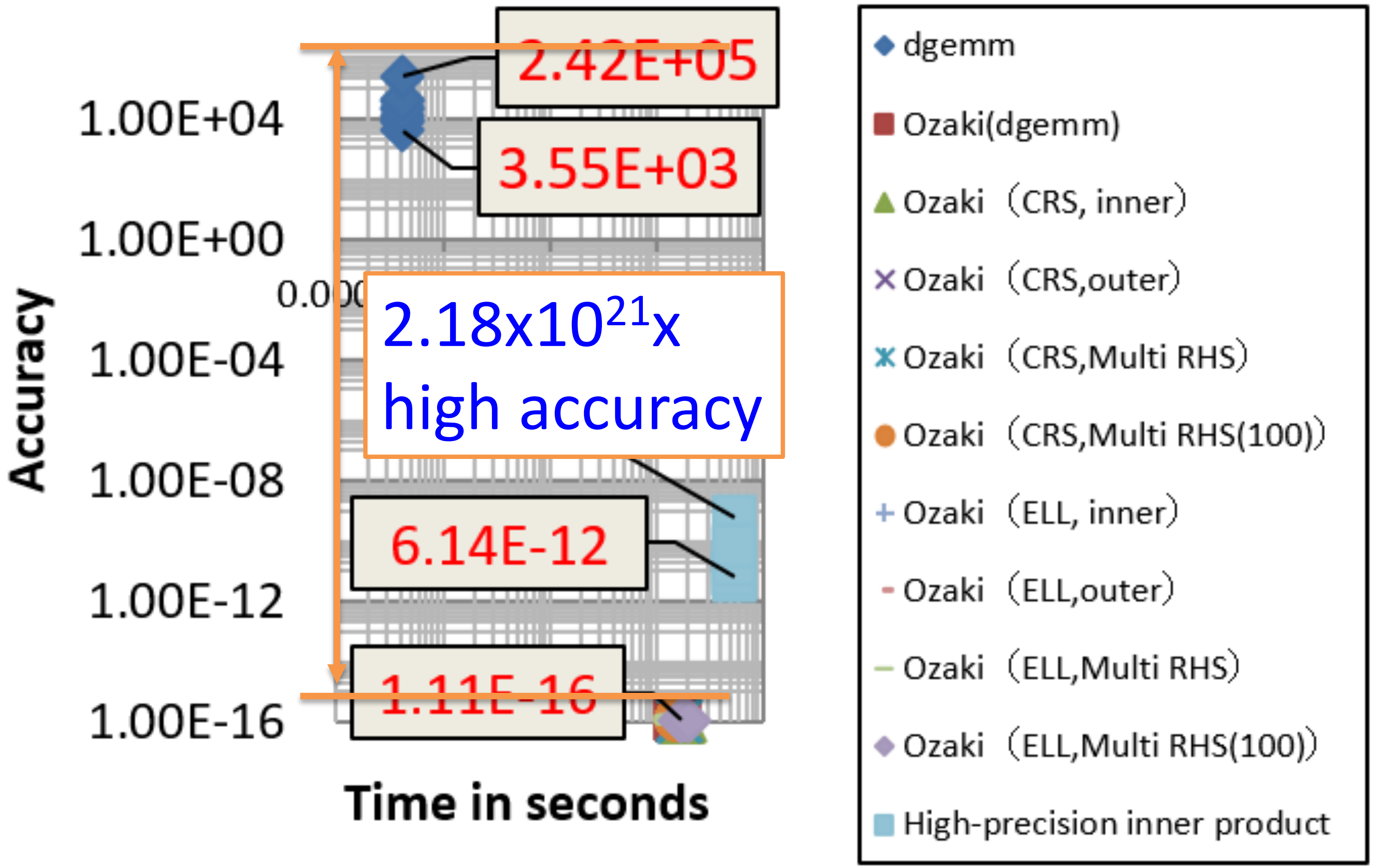


Execution speed and computation accuracy (N=500)



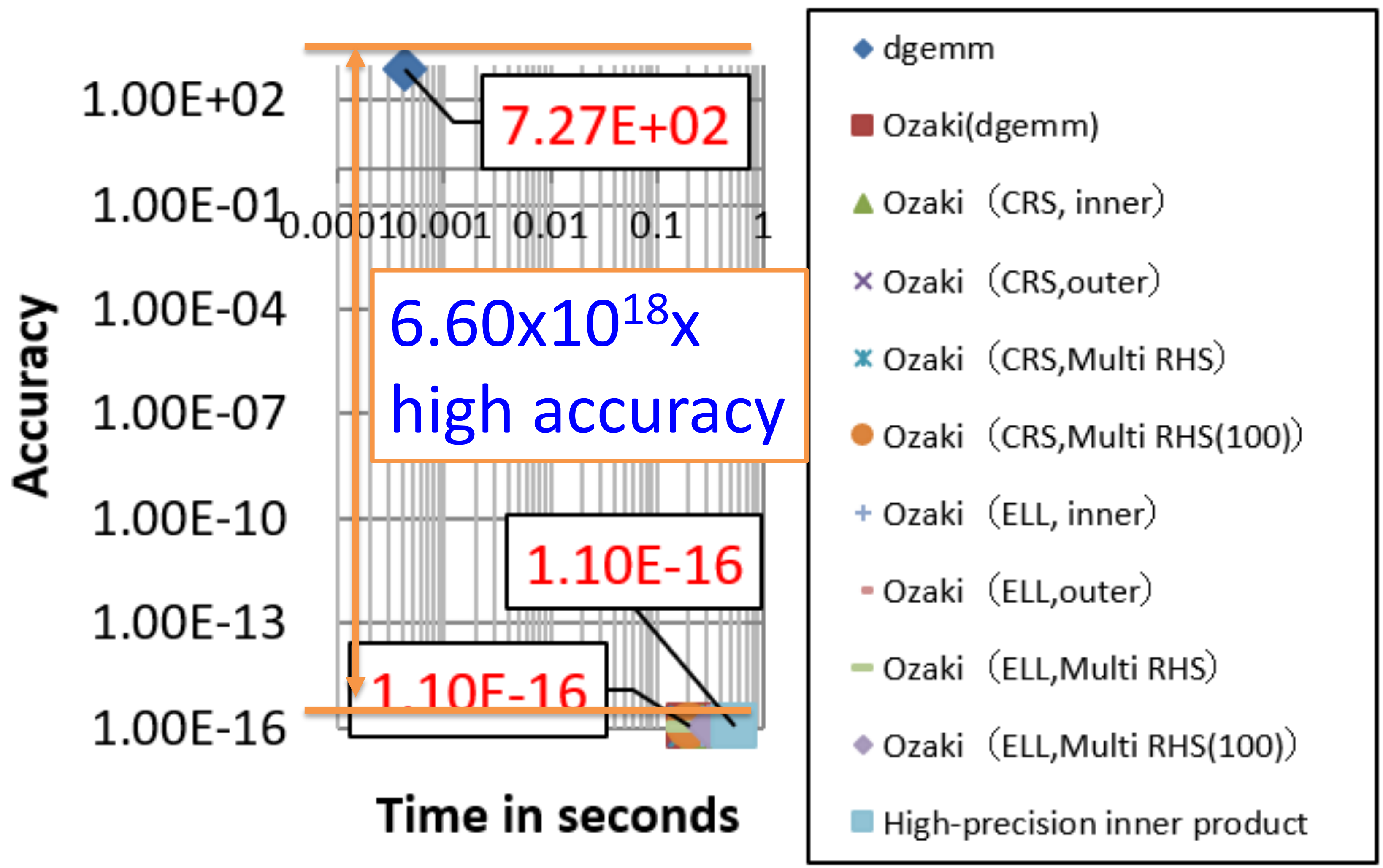
Execution speed and computation accuracy (N=500)

Matrix 3



Execution speed and computation accuracy (N=500)

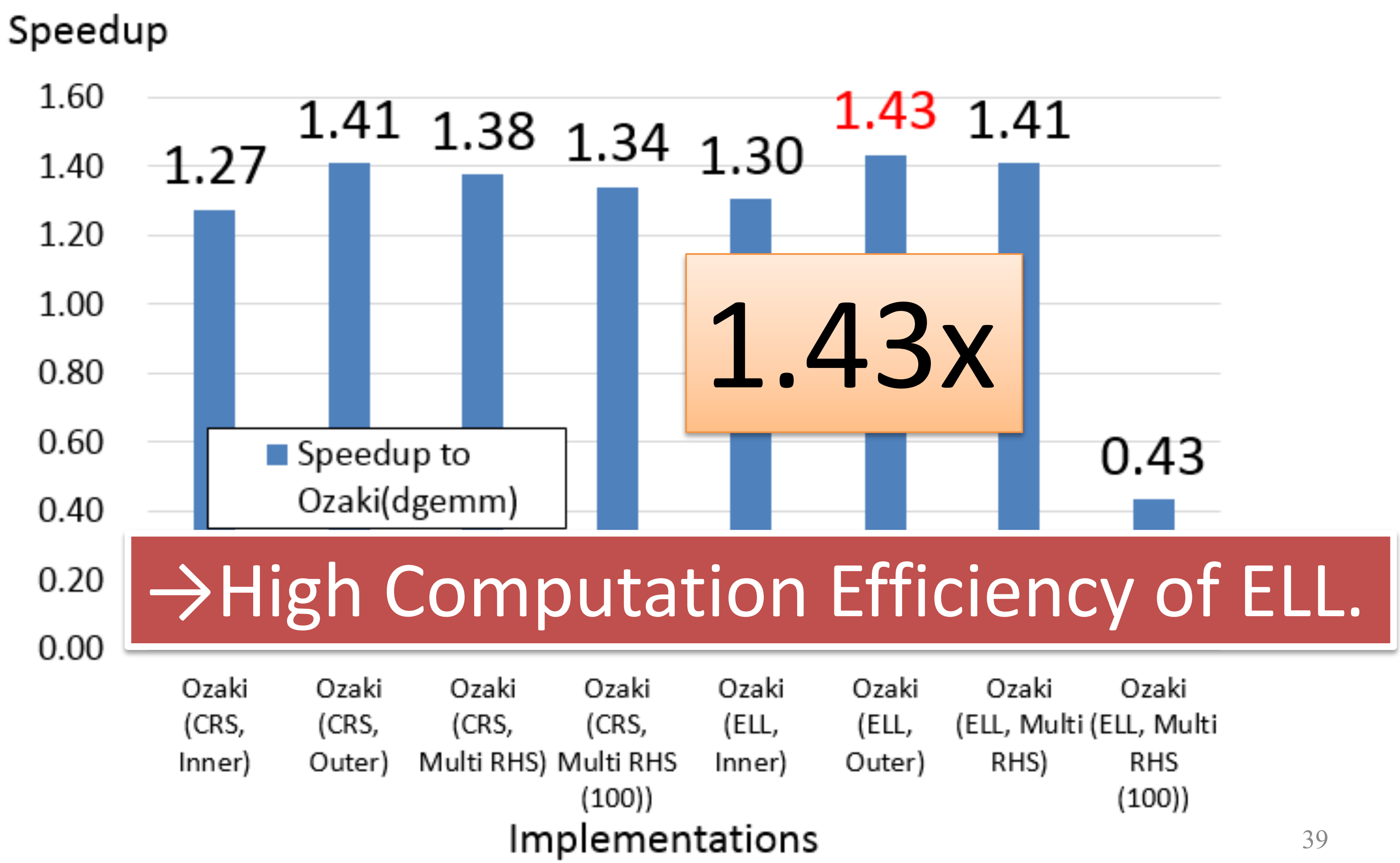
Matrix 4



KERNEL SPEEDUPS BASED ON OZAKI (DGEMM)

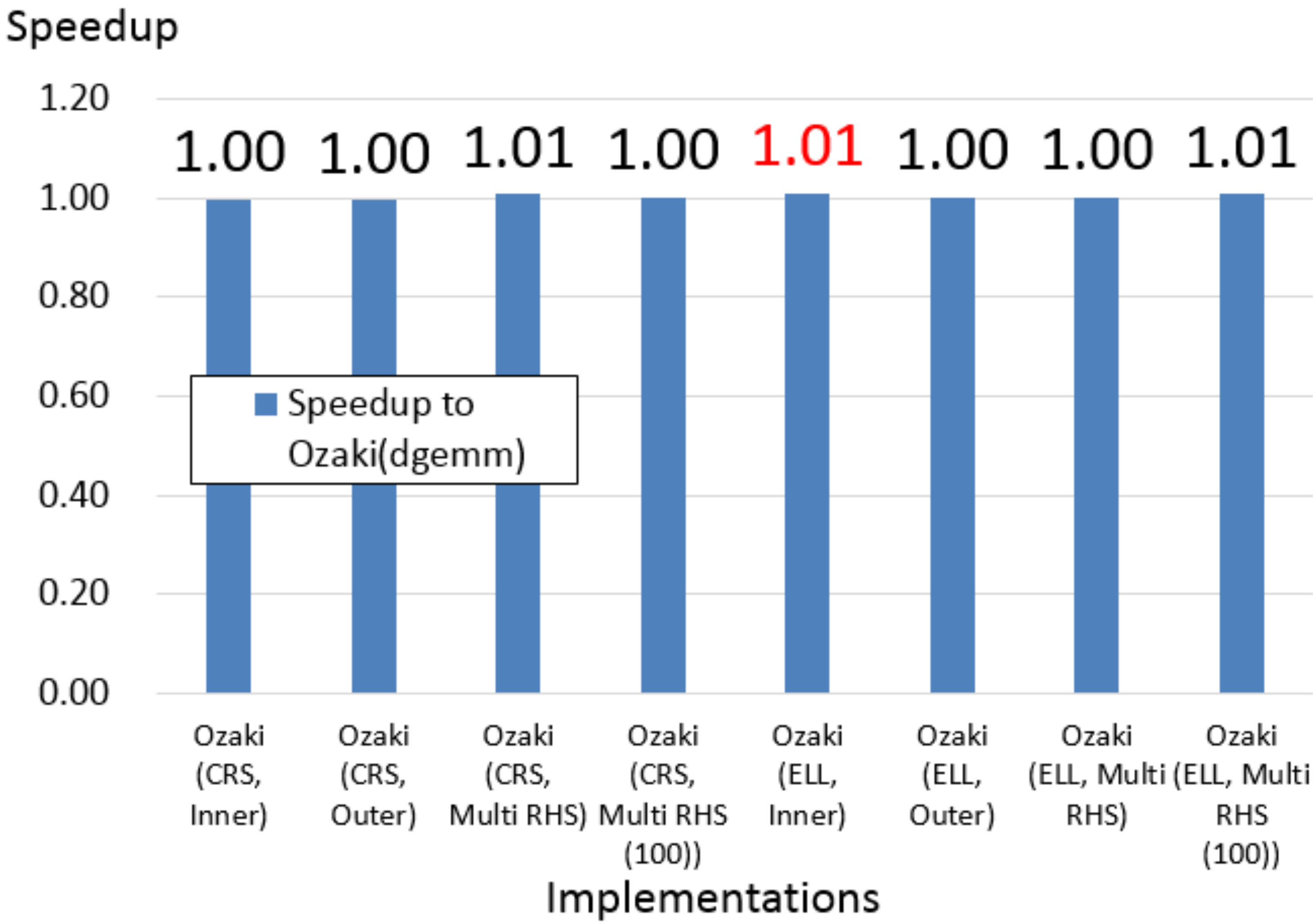
Kernel speedups
based on Ozaki (dgemm) (N=1000)

Matrix 1



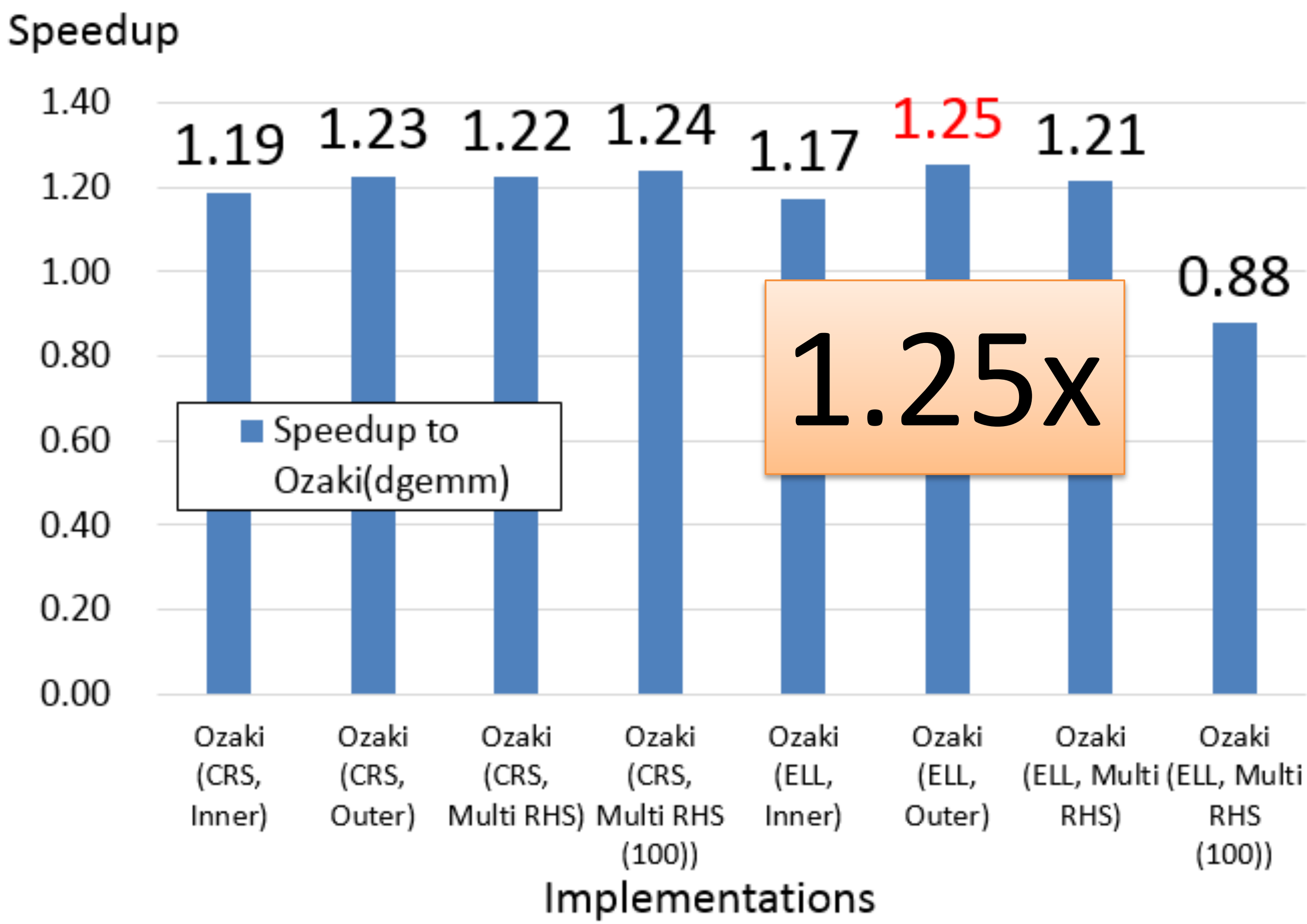
Kernel speedups based on Ozaki (dgemm) (N=1000)

Matrix 2



Kernel speedups based on Ozaki (dgemm) (N=1000)

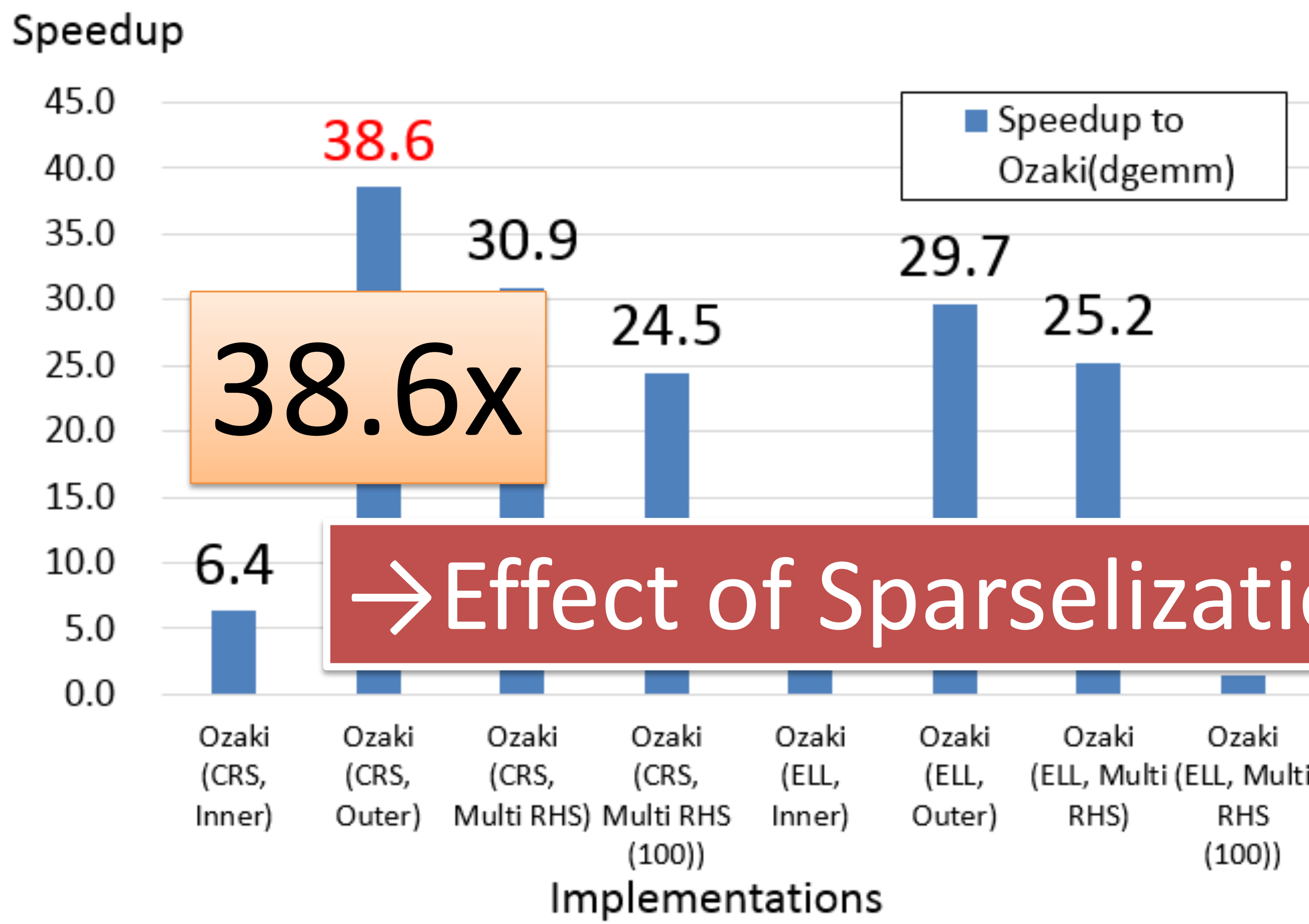
Matrix 3



Kernel speedups

based on Ozaki (dgemm) (N=1000)

Matrix 4



38.6x

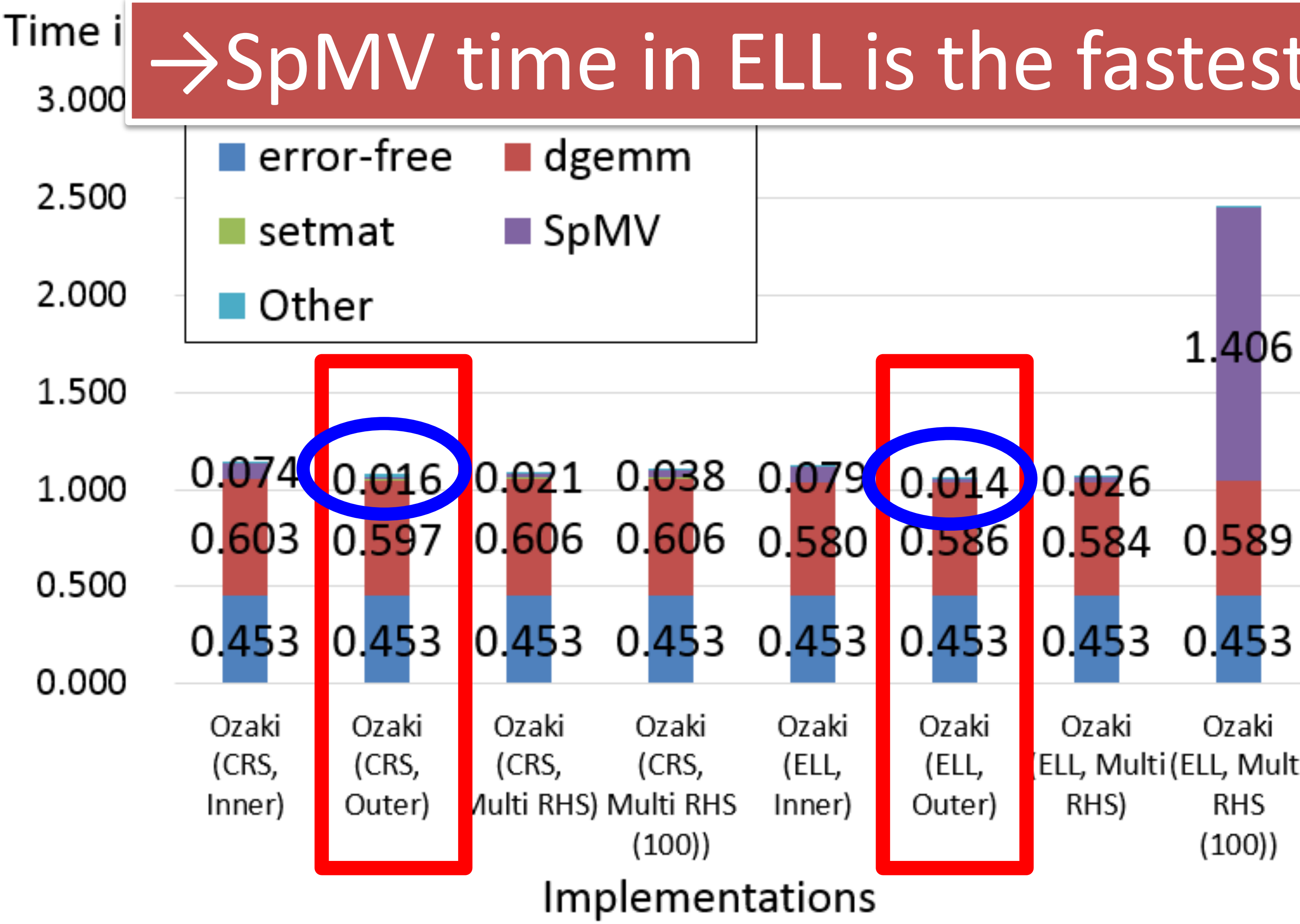
→Effect of Sparselization.

BREAKDOWN OF EXECUTION (WHOLE)

Breakdown of Execution (Whole) (N=1000)

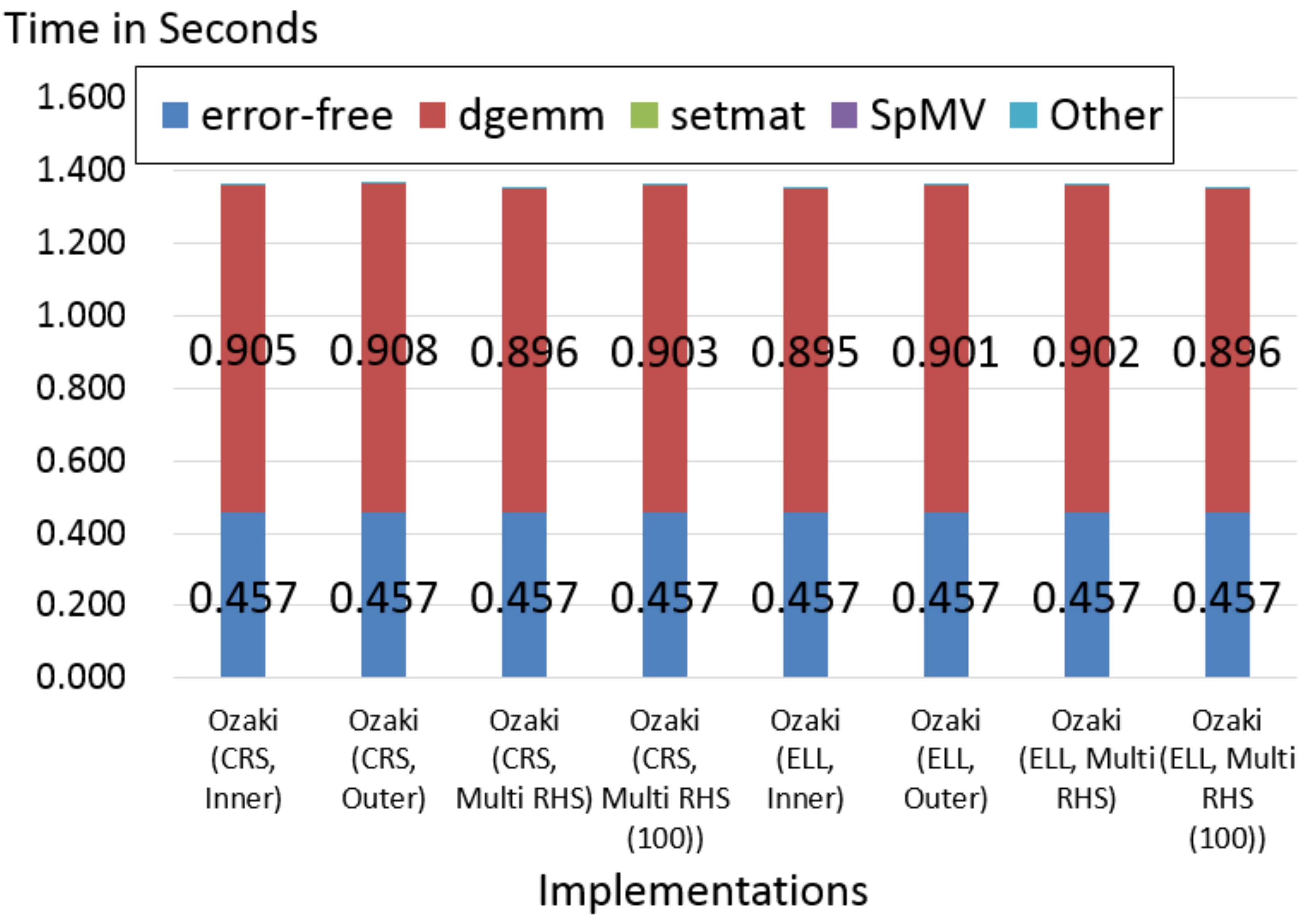
Matrix 1

→SpMV time in ELL is the fastest.



Breakdown of Execution (Whole) (N=1000)

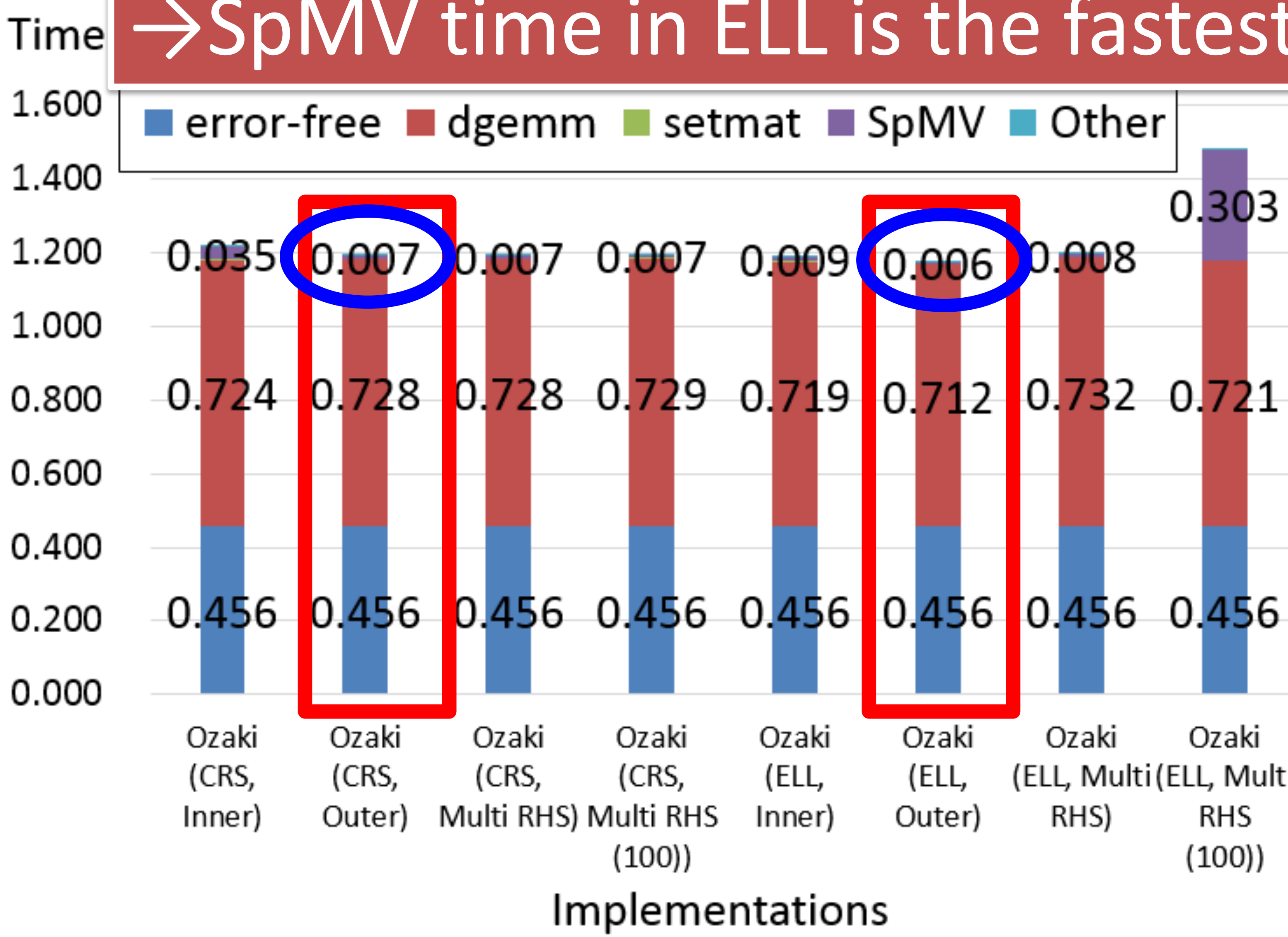
Matrix 2



Breakdown of Execution (Whole) (N=1000)

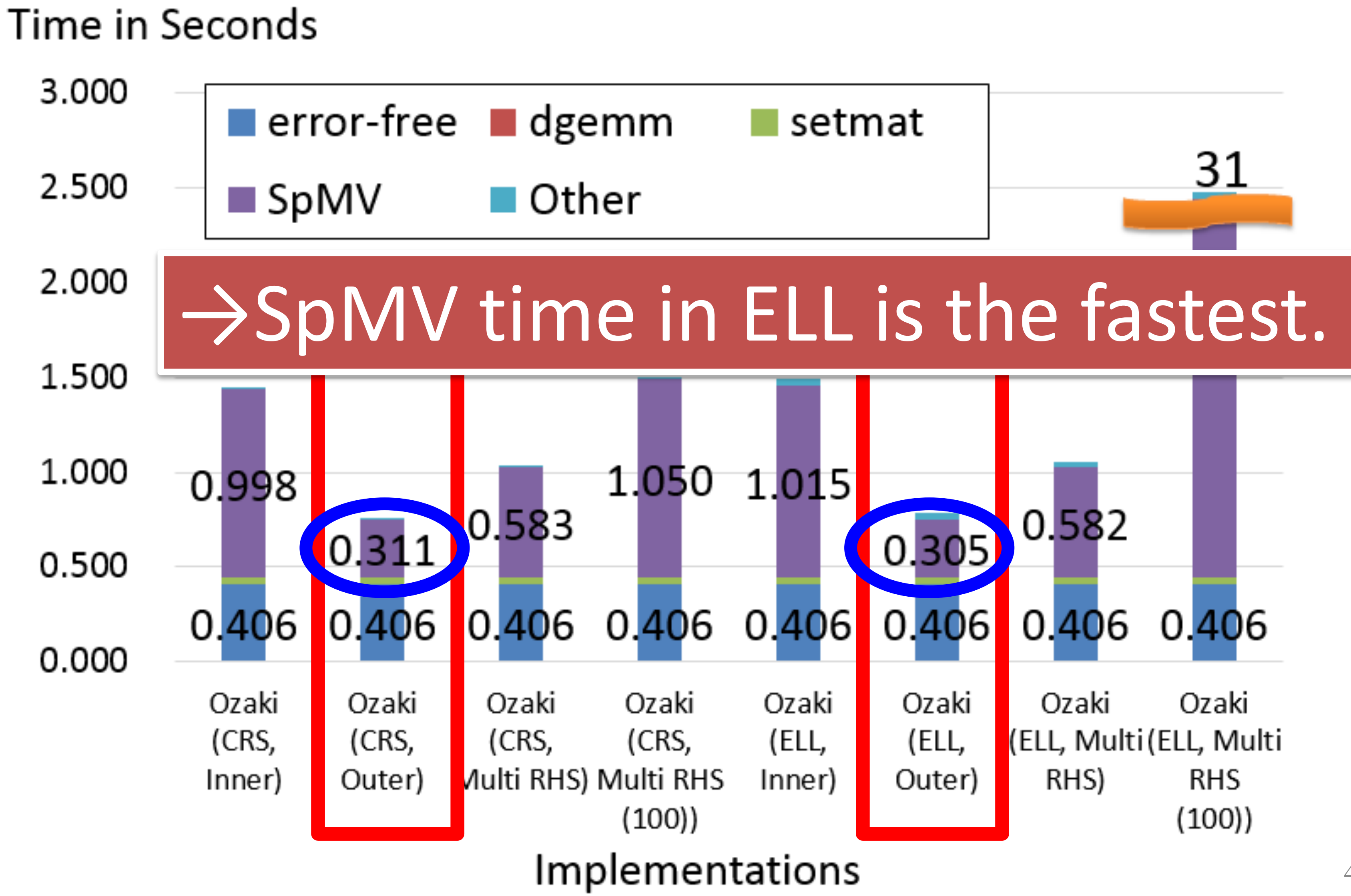
Matrix 3

→SpMV time in ELL is the fastest.



Breakdown of Execution (Whole) (N=1000)

Matrix 4



Outline

- Background
- Accurate precision MMM (Ozaki Method)
- Parallel Implementation for Multi-core CPUs and Its Evaluation
- Conclusion

Conclusion

- Accuracy assurance is required for numerical computations.
- Ozaki method, which is an algorithm for high precision matrix-matrix multiplication, is one of crucial approaches to do accuracy assurance for dense linear algebra libraries.
- We have implemented a method to convert dense matrices into sparse matrices by exploiting the nature of the target algorithm and adapting sparse-vector multiplication.
- Results with the FX100 supercomputer indicate that:
  - Implementation with the ELL format achieves 1.43x speedup.
  - A maximum of 38x speedup compared to conventional implementation for dense matrix operations with dgemm.
- Because of high efficiency of cache utilization of computations after error-free transformation in Ozaki method, ELL is better format to CRS format.

BACKUP

Observed Relative Errors

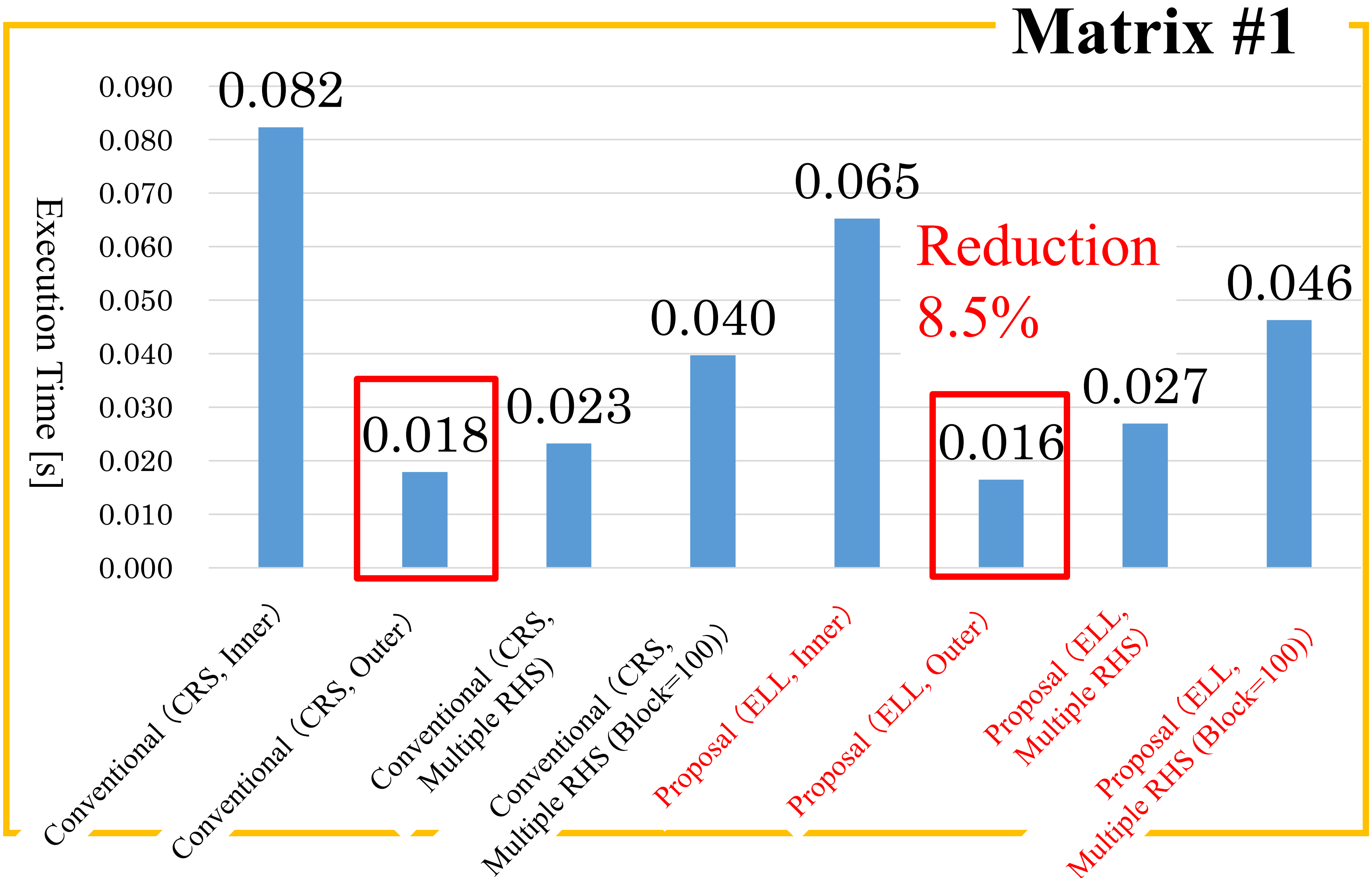
Observed Relative Errors between dgemm and Ozaki Method (maximum) (**double precision**)

Test Matrix No.	#1		#2		#3		#4	
Dimension	dgemm	Ozaki	dgemm	Ozaki	dgemm	Ozaki	dgemm	Ozaki
500	2.61 E-15	1.11 E-15	1.10 E-04	1.11 E-15	4.24 E-15	1.11 E-15	1.42 E-05	1.11 E-15
1000	3.77 E-15	1.11 E-15	4.31 E-05	1.11 E-15	7.55 E-15	1.11 E-15	3.23 E-05	1.11 E-15
2000	6.11 E-15	1.11 E-15	3.65 E-04	1.11 E-15	8.60 E-15	1.11 E-15	4.67 E-04	1.11 E-15

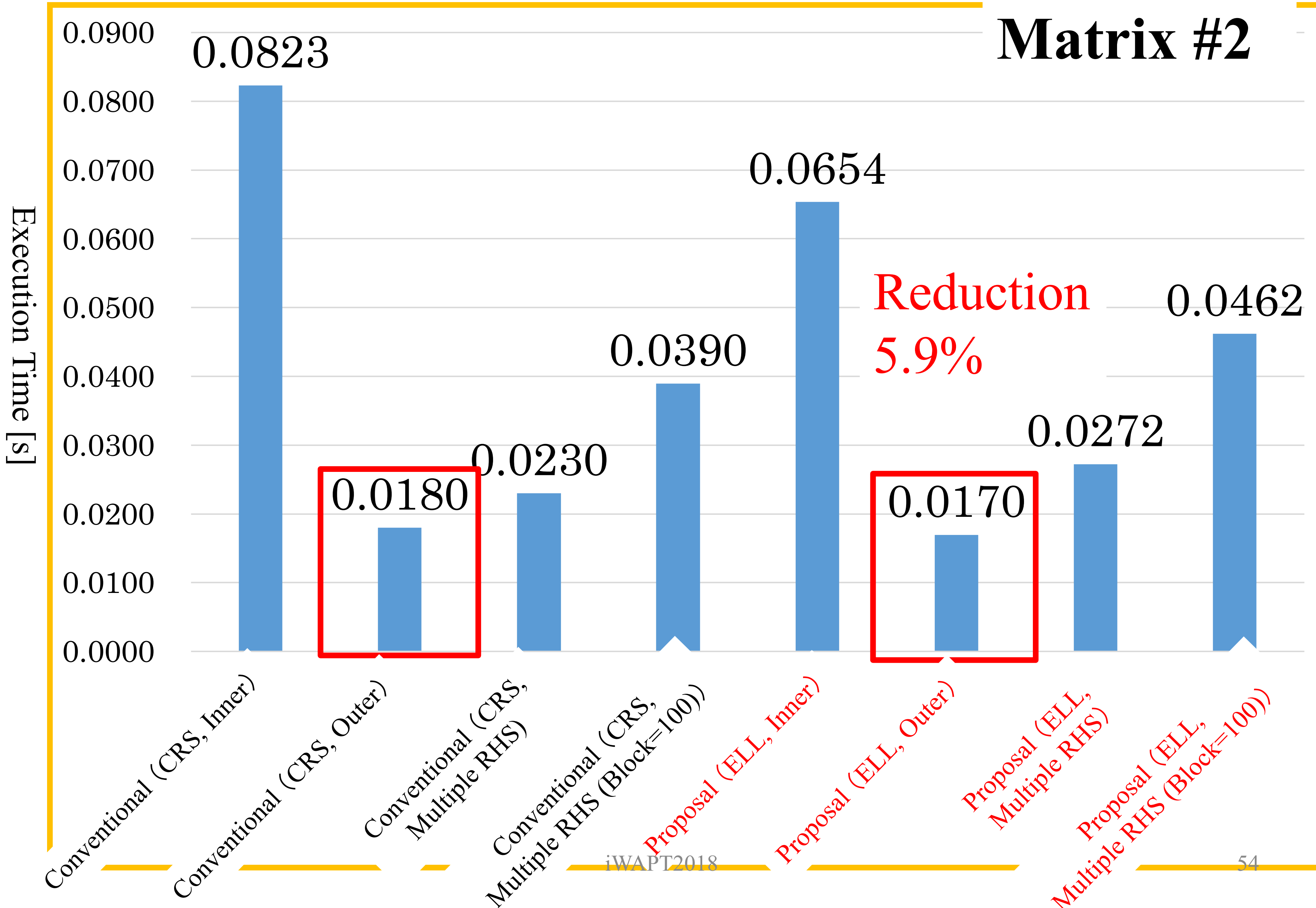
Dramatically errors occur in MMM with inverse matrix.

Ozaki Method → High precision
with comparison to dgemm.

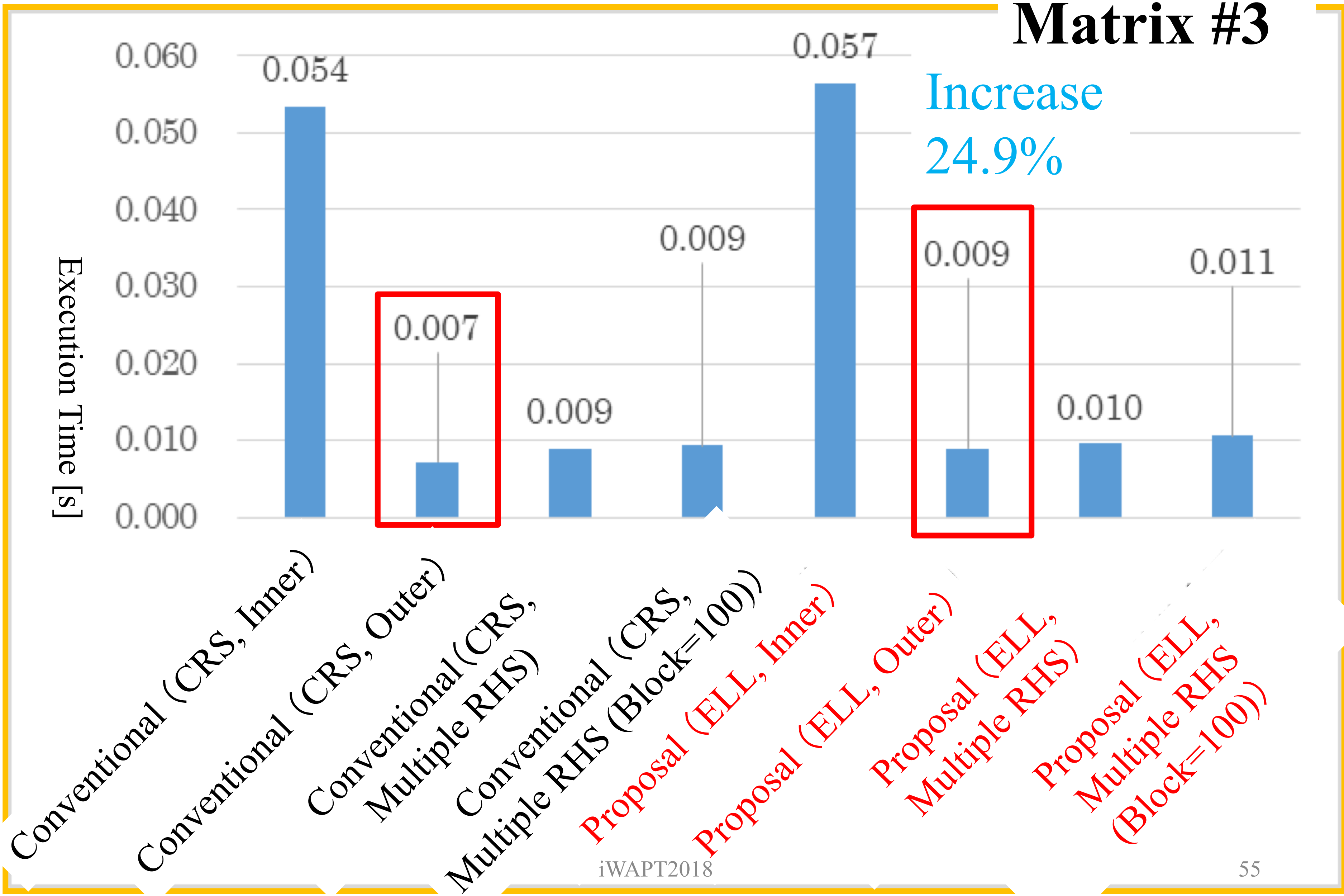
Comparison of Execution Speed (1/4)



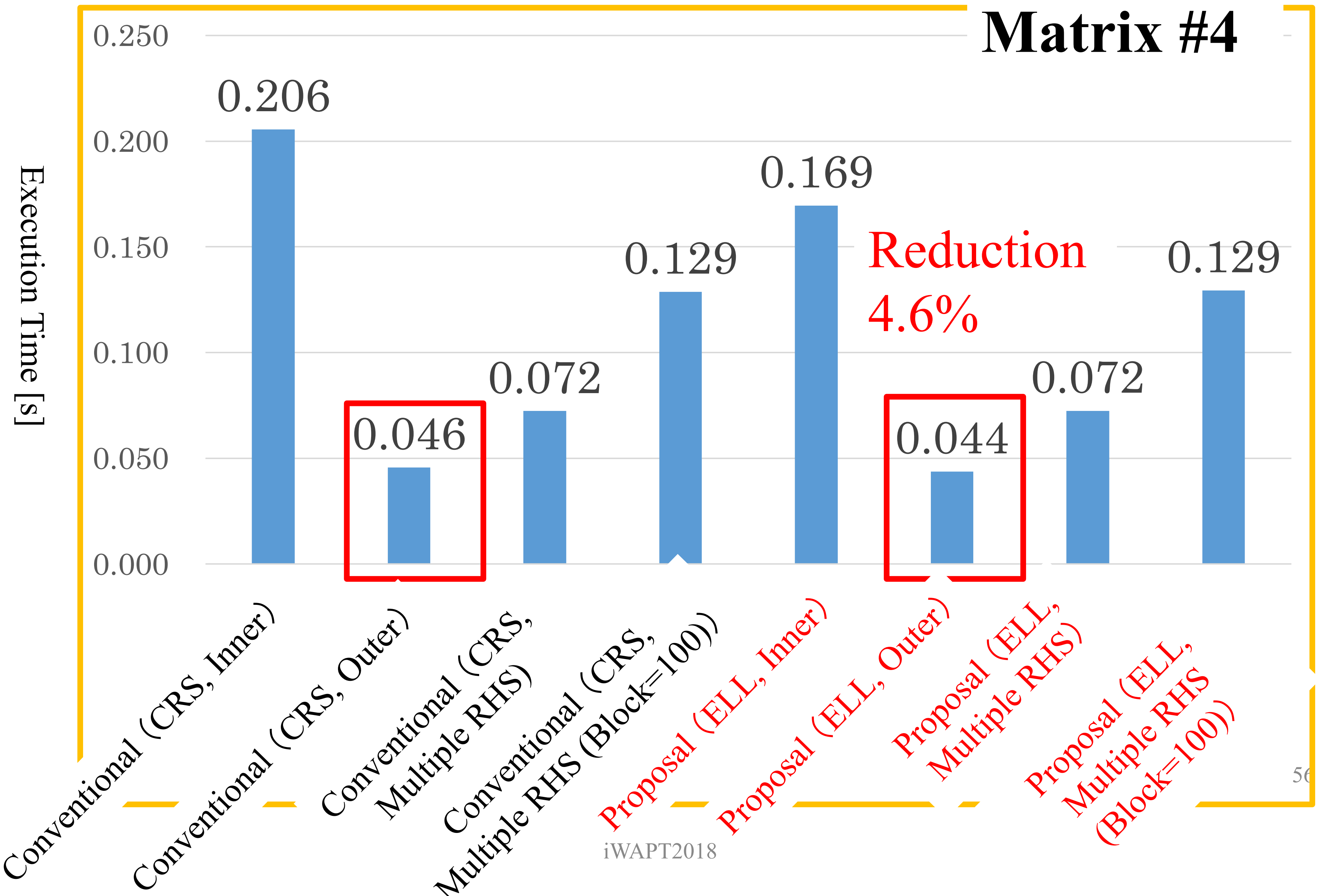
Comparison of Execution Speed (2/4)



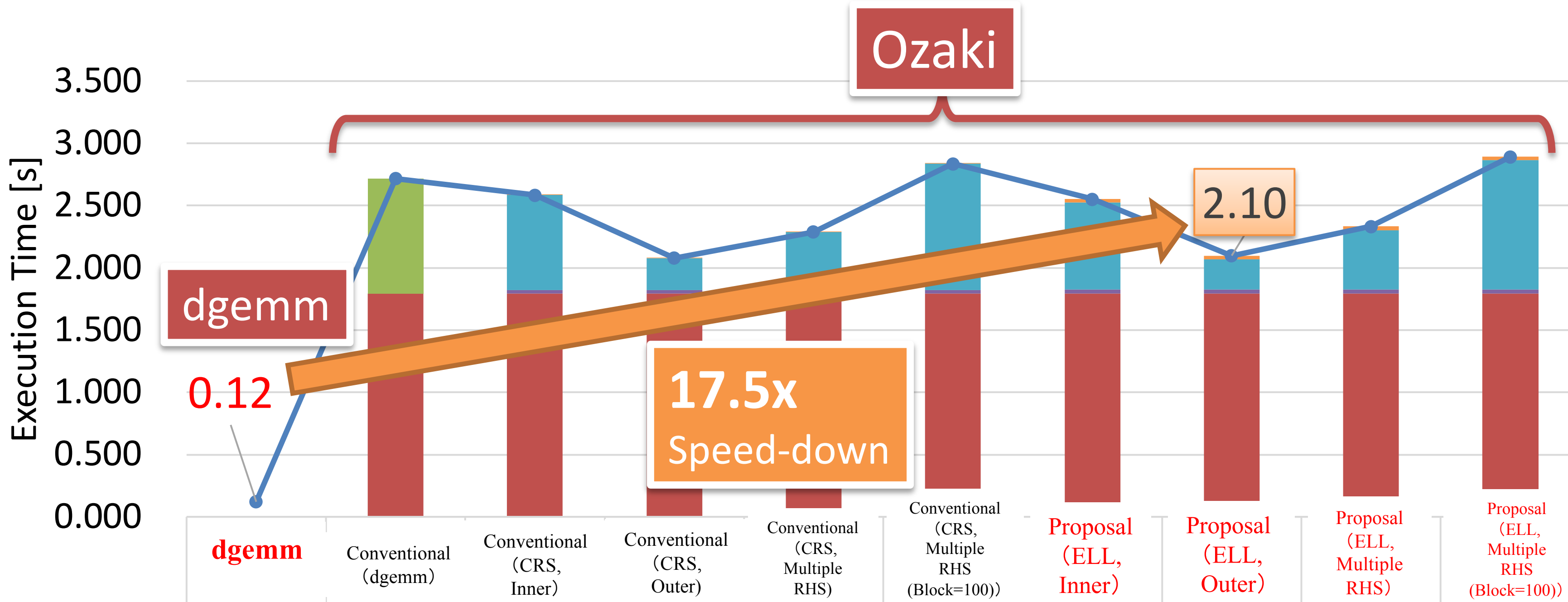
Comparison of Execution Speed (3/4)



Comparison of Execution Speed (4/4)



Breakdown of Whole Execution Time (FX100, N=2000, Test Matrix #4)



	dgemm	Conventional (dgemm)	Conventional (CRS, Inner)	Conventional (CRS, Outer)	Conventional (CRS, Multiple RHS)	Conventional (CRS, Multiple RHS (Block=100))	Proposal (ELL, Inner)	Proposal (ELL, Outer)	Proposal (ELL, Multiple RHS)	Proposal (ELL, Multiple RHS (Block=100))
Other	0.000	0.000	0.000	0.000	0.000	0.000	0.028	0.028	0.028	0.028
SpMV	0.000	0.000	0.761	0.256	0.465	1.012	0.700	0.246	0.480	1.039
Set SpMV	0.000	0.000	0.030	0.030	0.030	0.030	0.031	0.031	0.031	0.031
Dgemm	0.000	0.923444	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Error Free	0.000	1.793	1.793	1.793	1.793	1.793	1.793	1.793	1.793	1.793
Kernel	0.121084	2.717	2.584	2.079	2.288	2.836	2.553	2.098	2.332	2.892

Implementations

Error Free | Dgemm | Set SpMV | SpMV | Other | Kernel

All decomposed matrices are sparse matrices, after error free transformation.

入力行列4（N=1000）の場合のプロファイル結果

従来手法（CRS，外部並列）

	実行時間（sec）	浮動小数点演算ピーク比	MFLOPS	L1D ミス率（/ロード・ストア数）	L1D ミス数	L2 ミス率（/ロード・ストア数）	L2 ミス数
Thread 0	0.04	6.08%	2141	1.94%	2.12E+06	0.05%	5.28E+04
Thread 1	0.04	6.09%	2143	1.94%	2.12E+06	0.04%	4.90E+04
Thread 2	0.04	6.12%	2155	1.93%	2.12E+06	0.04%	4.90E+04
Thread 29	0.04	5.92%	2084	[illegible]	[illegible]	[illegible]	4.80E+04
Thread 30	0.04	5.93%	2086	[illegible]	[illegible]	[illegible]	4.80E+04
Thread 31	0.04	5.95%	2094	[illegible]	[illegible]	[illegible]	5.69E+04
Process	0.04	5.94%	66873	1.94%	6.62E+07	0.05%	1.57E+06

6.62E+07

提案手法（ELL，外部並列）

	実行時間（sec）	浮動小数点演算ピーク比	MFLOPS	L1D ミス率（/ロード・ストア数）	L1D ミス数	L2 ミス率（/ロード・ストア数）	L2 ミス数
Thread 0	0.04	6.51%	2291	1.94%	2.11E+06	0.05%	5.15E+04
Thread 1	0.04	6.53%	2297	1.94%	2.11E+06	0.05%	4.99E+04
Thread 2	0.04	6.54%	2302	1.94%	2.11E+06	0.04%	4.91E+04
Thread 29	0.04	6.30%	2217	[illegible]	[illegible]	[illegible]	4.79E+04
Thread 30	0.04	6.27%	2208	[illegible]	[illegible]	[illegible]	4.79E+04
Thread 31	0.04	6.31%	2220	[illegible]	[illegible]	[illegible]	5.68E+04
Process	0.04	6.31%	71070	1.94%	6.60E+07	0.05%	1.57E+06

6.60E+07

0.21%削減

黄色の部分は，Level 1データキャッシュにおける情報

キャッシュヒット率の向上

入力行列4（N=1000）の場合のプロファイル結果

従来手法（CRS，外部並列）

	実行時間（sec）	浮動小数点演算ピーク比	MFLOPS	L1D ミス率（/ロード・ストア数）	L1D ミス数	L2 ミス率（/ロード・ストア数）	L2 ミス数
Thread 0	0.04	6.08%	2141	1.94%	2.12E+06	0.05%	5.28E+04
Thread 1	0.04	6.09%	2143	1.94%	2.12E+06	0.04%	4.90E+04
Thread 2	0.04	6.12%	2155	1.93%	2.12E+06	0.04%	4.90E+04
Thread 29	0.0	[illegible]	2084	1.94%	2.05E+06	0.05%	4.80E+04
Thread 30	0.0	[illegible]	2086	1.94%	2.05E+06	0.05%	4.80E+04
Thread 31	0.04	[illegible]	2094	1.94%	2.05E+06	0.05%	5.69E+04
Process	0.04	5.94%	66873	1.94%	6.62E+07	0.05%	1.57E+06

5.94%

提案手法（ELL，外部並列）

	実行時間（sec）	浮動小数点演算ピーク比	MFLOPS	L1D ミス率（/ロード・ストア数）	L1D ミス数	L2 ミス率（/ロード・ストア数）	L2 ミス数
Thread 0	0.04	6.51%	2291	1.94%	2.11E+06	0.05%	5.15E+04
Thread 1	0.04	6.53%	2297	1.94%	2.11E+06	0.05%	4.99E+04
Thread 2	0.04	6.54%	2302	1.94%	2.11E+06	0.04%	4.91E+04
Thread 29	0.0	[illegible]	2217	1.94%	2.05E+06	0.05%	4.79E+04
Thread 30	0.0	[illegible]	2208	1.94%	2.05E+06	0.05%	4.79E+04
Thread 31	0.04	[illegible]	2220	1.94%	2.05E+06	0.05%	5.68E+04
Process	0.04	6.31%	71070	1.94%	6.60E+07	0.05%	1.57E+06

6.31%

0.37ポイント向上

黄色の部分は，Level 1データキャッシュにおける情報

キャッシュヒット率の向上