Fast Computation of double precision sparse matrix in BCRS and DD vector product using AVX2

Toshiaki Hishinuma†, Teruo Tanaka†, Akihiro Fujii†, and Hidehiko Hasegawa††
†Kogakuin University ††University of Tsukuba

- High precision arithmetics can improve the convergence of Krylov subspace methods.
- We have accelerated "Double-Double"(DD) precision arithmetics using AVX2. DD arithmetics is one of high precision arithmetics.
- DD-SpMV ($y_{DD}=A_D x_{DD}$) in CRS (compressed row storage format) using AVX2 needs "processing for the remainder" and "sum. of elements in the SIMD register" for each rows. They are factors that affect performance.
- BCRS (Block CRS) can reduce these factors that affect performance . However it may result in increased computation.

"Double-Double" precision

- Double-Double precision(DD) arithmetic uses two double precision variables to implement one quadruple precision variable.
- DD mult. and add. consists of 19 double precision operations.

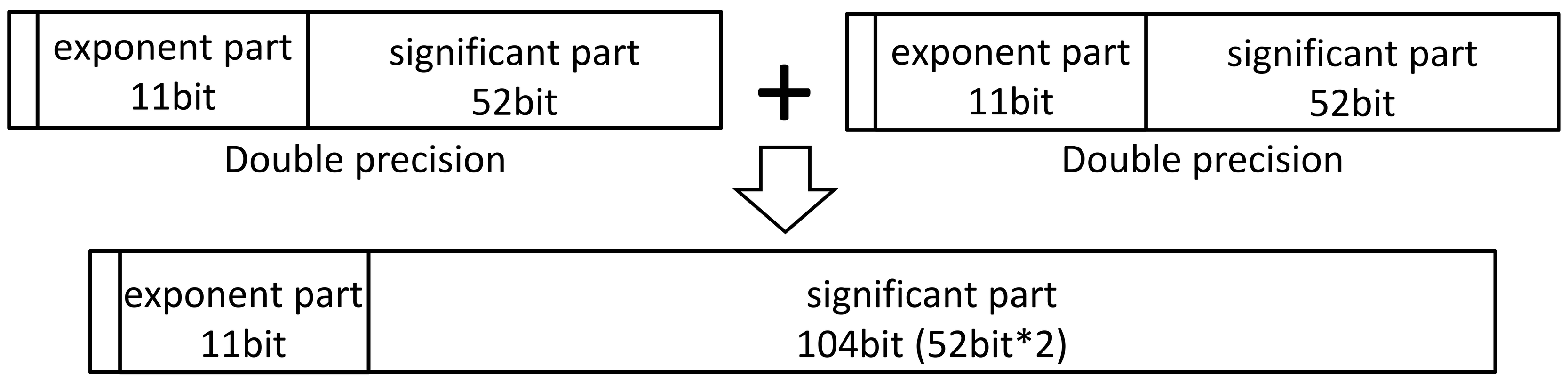


Fig.1 Double-Double precision format

Table 1 bytes/flops of ***y*** = A***x***

	bytes / flops
$y_D = A_D x_D$	14 (28 bytes / 2 flops)
$y_{DD} = A_{DD} x_{DD}$	2.48 (52 bytes / 21 flops)
◎ $y_{DD} = A_D x_{DD}$	2.32 (44 bytes / 19 flops)

- We accelerate double precision sparse matrix and DD vector product.
- This allowed the matrix data size to half and bytes/flops is 94% compared to DD matrix and DD vector product.

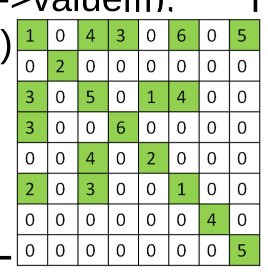
Double precision sparse matrix and DD vector product using AVX2

- AVX2 must process four double precision data simultaneously.
  - DD-SpMV in CRS needs "**processing for the remainder**" for each rows (1, 2, 3).
- When storing ***y***, DD-SpMV in CRS needs **"sum. of elements in the SIMD register"**.
- BCRS can reduce "processing for the remainder" and "sum. of elements".

Table 2 Feature of each matrix format

	Loading ***x***	Processing for the remainder	Sum. of elements in SIMD reg.	Storing ***y***
CRS	set	each row	each row	each row
BCRS1x4	load	none	each row	each row
× BCRS2x2	set	none	sets of two	sets of two
◎ BCRS4x1	broadcast	none	none	sets of four

```
for(i=0; i<N; i++){
  for(j=A->ptr[i]; j<A->ptr[i+1]-3; j+=4){
    xv = _mm256_set_pd( x[A->index[j]],
                        x[A->index[j+1]],
                        x[A->index[j+2]],
                        x[A->index[j+3]] );
    av = _mm256_load_pd(&A->value[j]);
    DD_ADD_MULT (av, xv, yv);
  }
  processing remainder();
  summation of elements();
```



factors that affect performance

Fig.2 SpMV in CRS

```
for(i=0; i<N; i++){
  for(j=A->ptr[i]; j<A->ptr[i+1]; j+=4){
    av = _mm256_load_pd(&A->value[j*4]);
    xv = _mm256_broadcast_sd(&x[A->index[j]*4]);
    DD_ADD_MULT (av, xv, yv);
  }
  _mm256_store_pd(&y[i], yv);
}
```

BCRS consists zero elements
→ It increases computations

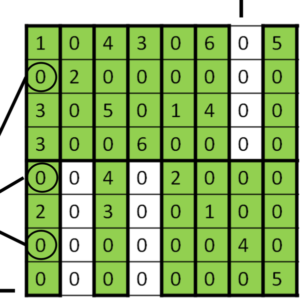


Fig.3 SpMV in BCRS4x1

Performances of DD-SpMV in BCRS format

(Intel Core i7 4770K 3.4GHz 4core 16GB, CentOS 6.4, intel C/C++ Compiler 13.1.0)

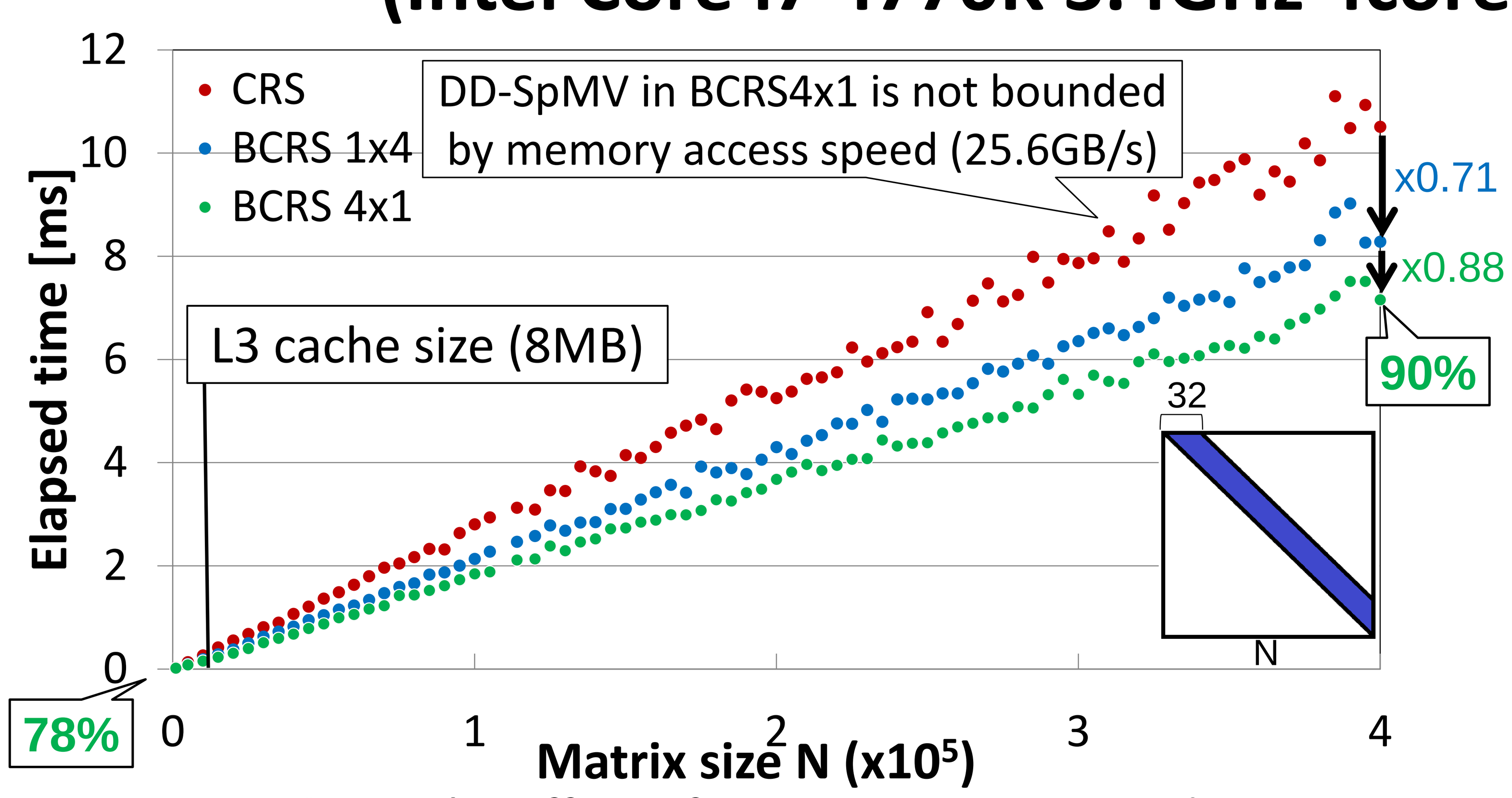


Fig.4 The effect of memory access speed
(band matrix, nnz/row=32, It does not occur processing for the remainder)

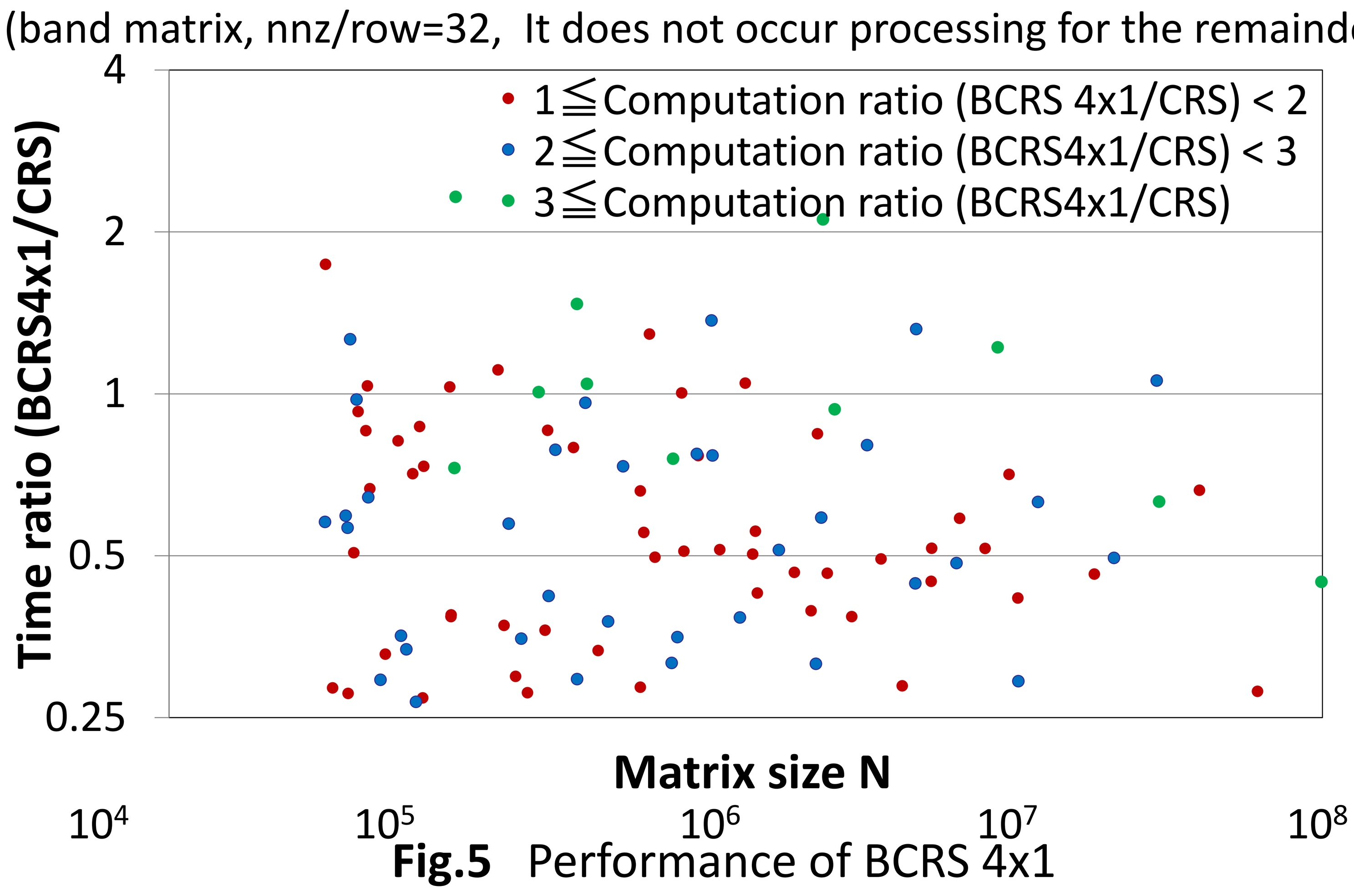


Fig.5 Performance of BCRS 4x1
(Univ. of Florida sparse matrix collection, 100 matrices)

- The effect of BCRS4x1 is effective for the large size matrices ($N>10^6$)
- In small size matrices, the effect of improving memory access is low.

Table3 The effect of DD-SpMV in BCRS4x1 [ms]
(band matrix, nnz/row = 33)

N = 10^4 (in the cache)	CRS		BCRS1x4		BCRS4x1
Computation time	0.16	-7 %	0.15		0.15
Processing for the remainder	0.04		0.04		none
Sum. of the elements in the SIMD reg.	0.05		none		none
Total	0.25	-24 %	0.19	-22%	0.15

N = 4.0 × 10^5 (out of the cache)	CRS		BCRS1x4		BCRS4x1
Computation time	9.07	-11 %	8.16		8.16
Processing for the remainder	1.12		1.12		none
Sum. of the elements in the SIMD reg.	1.32		none		none
Total	11.51	-29 %	9.28	-13 %	8.16

- "processing for the remainder" and "sum. of elements in SIMD reg." are 21% of elapsed time.
- The computation time of BCRS1x4 is 7-11% faster than that of CRS.
  - It is the effect of improving memory access by BCRS4x1.
  - The effect of improving memory access is large for the large size matrices.

Table 4 Total elapsed time of DD-SpMV [ms]
(relative performance)

	Total time (100 mat.)	The number of the best matrices
CRS	730 (1.35)	14
BCRS 1x4	880 (1.33)	4
BCRS 4x1	540 **(1.02)**	82
The best combinations	530 (1)	100

The case of optimally-combinations of CRS, BCRS1x4 BCRS4x1

- Total elapsed time of BCRS4x1 is 1.02 times of the best combinations.
- BCRS4x1 does not need combinations of other storage format.
- BCRS4x1 is the best storage format.

References

- Bailey, D, H.: *High Precision Floating-Point Arithmetic in Scientific Computation, Computing in Science and Engineering, pp. 54-61 (2005)*
- T.Hishinuma, et al.: *AVX acceleration of DD arithmetic between a sparse matrix and vector, Lecture Notes in Computer Science 8384, pp. 622-631 (2014)*

Conclusions

- We accelerate DD-SpMV in BCRS4x1 using AVX2. It does not need "processing for the remainder" and "sum. of element in SIMD reg.".
- Performances of DD-SpMV in BCRS4x1 are not bounded by memory access speed.
- BCRS4x1 is effective for the large size matrices ($N>10^6$). That of the effect of improving memory access is large.
- The best storage format is BCRS4x1. Total elapsed time of BCRS4x1 is 1.02 times of the best combinations.