Acceleration of Structural Analysis Software FrontISTR on NEC SX-Aurora TSUBASA

Toshiaki Hishinuma[1], Ryo Igarashi[1], Naoki Morita[1], Yu Ihara[1]
Moriyuki Takamura[2], Hiroshi Hirano[2],
Takashi Hagiwara[3], Naoki Iwata[3], Hiroshi Okuda[4]

[1] Research Institute for Computational Science Co. Ltd
[2] Industry SuperComputer Promotion Center
[3] NEC Corporation
[4] The University of Tokyo

Introduction

- Structural Analysis using finite element method (FEM) consists of:
  1. Generation of stiffness matrix A from mesh data
  2. Solve $A\boldsymbol{x} = \boldsymbol{b}$, A is sparse matrix
     - In many cases, a krylov subspace method is used to solve linear equation
- NEC SX-Aurora TSUBASA (SXAT)
  - x86_64 host computer called Vector Host (VH)
  - Vector accelerator board called Vector Engine (VE) connected by PCIe
- The VE has 2.15 TFLOPS DP Peak and 1,228 GB/s memory B/W
  - 8 high-performance vector cores
  - 32 DP floating point element executions simultaneously

Problems of FEM on VE

1. The stiffness matrix generation is not suitable for VE
   - Involves many integer operations

2. Acceleration of sparse matrix and vector multiplication (SpMV)
   - SpMV occupies most of the execution time to Solve $\boldsymbol{Ax} = \boldsymbol{b}$
   - SpMV needs indirect memory access
     - Indirect memory access cause performance degradation in vectorization
   - Requires a sparse matrix storage format suitable for vectorization

Objective

GOAL
Development of High-Performance FEM Software on SXAT
based on Open-source FEM Software FrontISTR†

Today's Topic

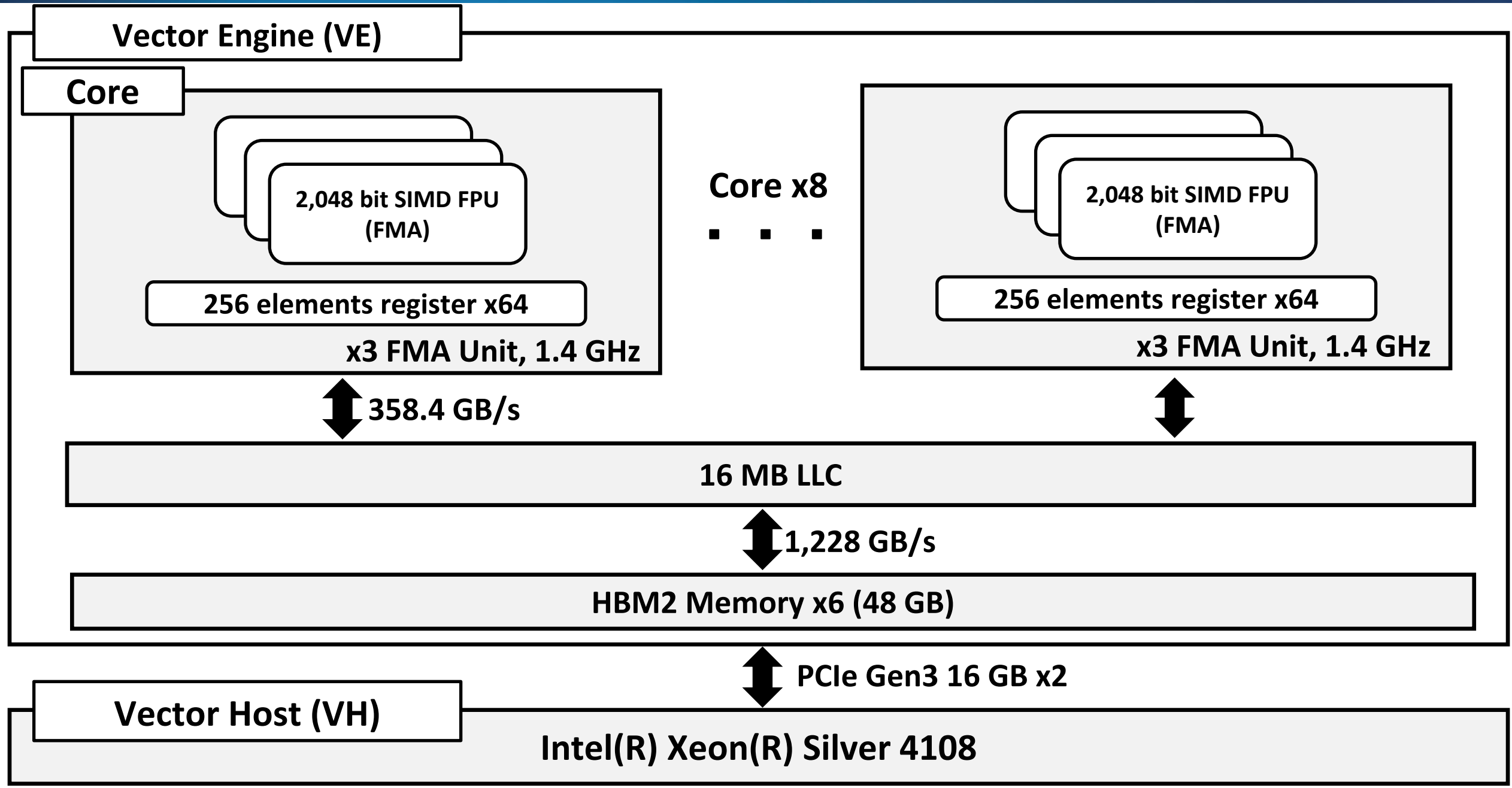
1. Performance evaluation of conjugate gradient (CG) method
   - CG method is one of krylov subspace methods
   - Evaluate the perf. of SpMV stored in BCRS and JAD formats
2. Acceleration of the overall FEM program using AVEO††
   - AVEO is VH-VE data transfer API
   - AVEO enables VH and VE hybrid computing

† https://www.frontistr.com/
†† https://sx-aurora.github.io/posts/AVEO/

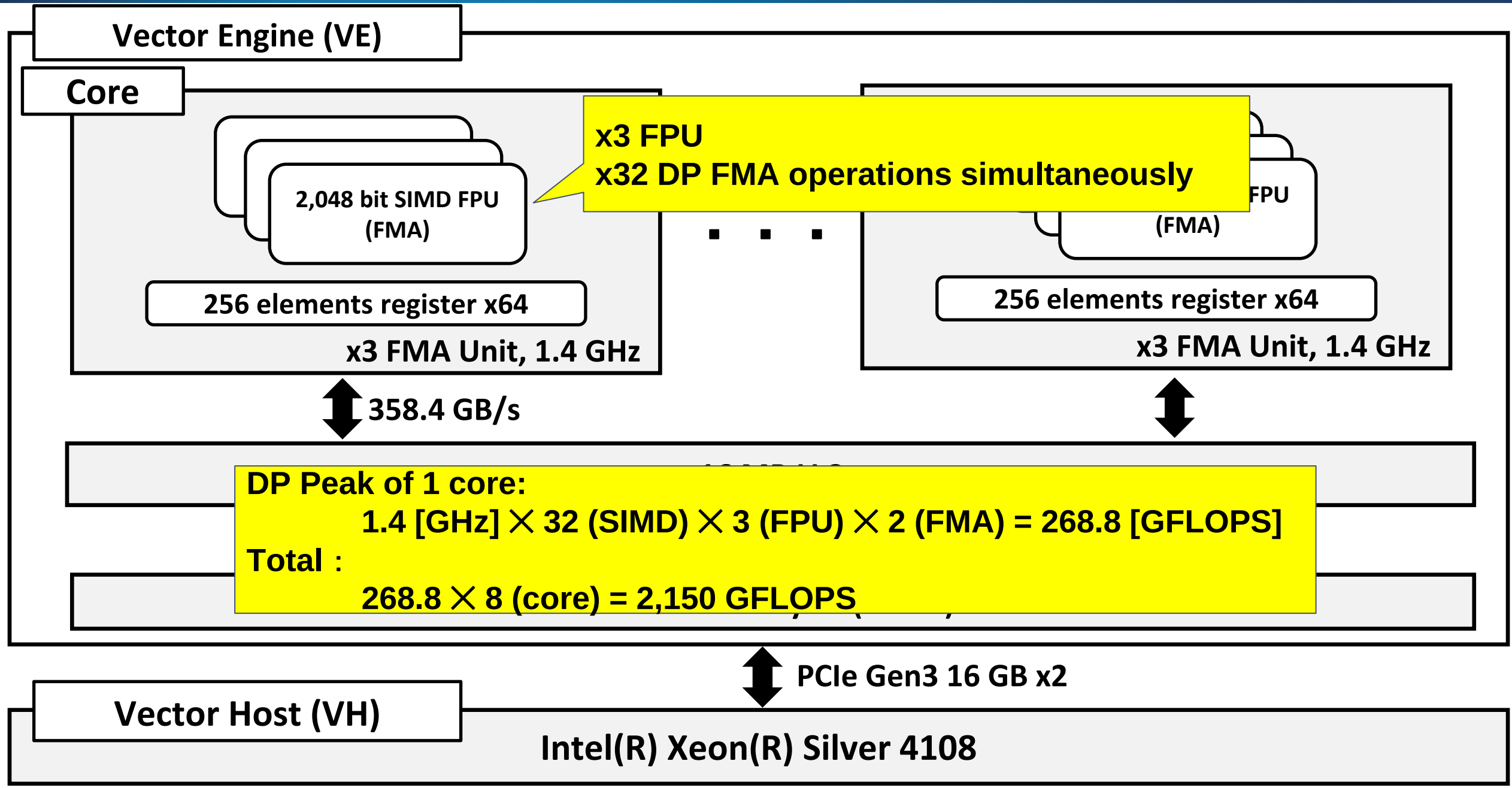
Agenda

- Introduction
- NEC SX-Aurora TSUBASA
- FrontISTR structural analysis on SXAT
  - SpMV and matrix storage format
  - Hybrid computing using AVEO
- Performance evaluation
- Conclusion

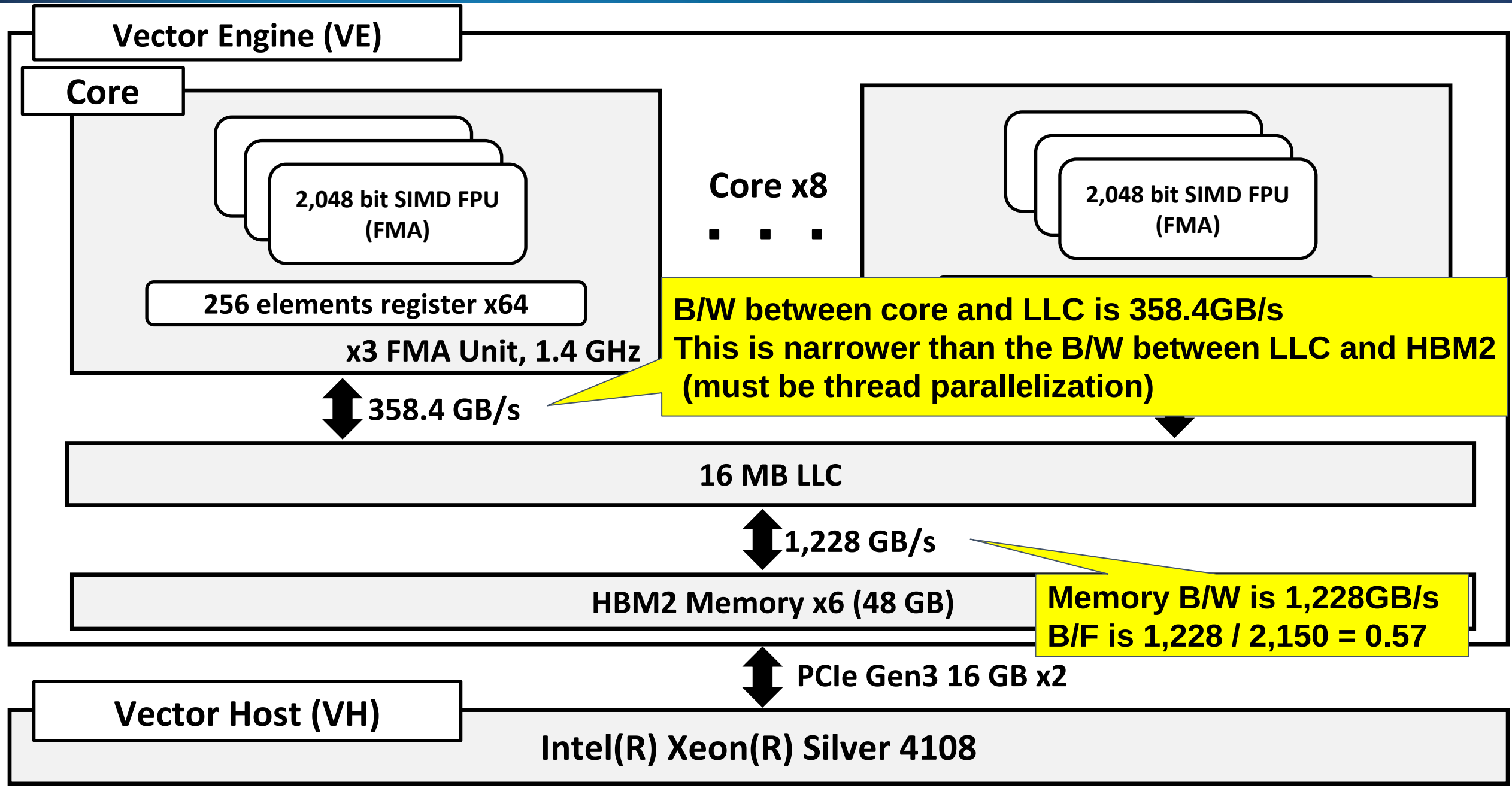
SX-Aurora TSUBASA Archtecture (VE model Type-10B)



SX-Aurora TSUBASA Archtecture (VE model Type-10B)

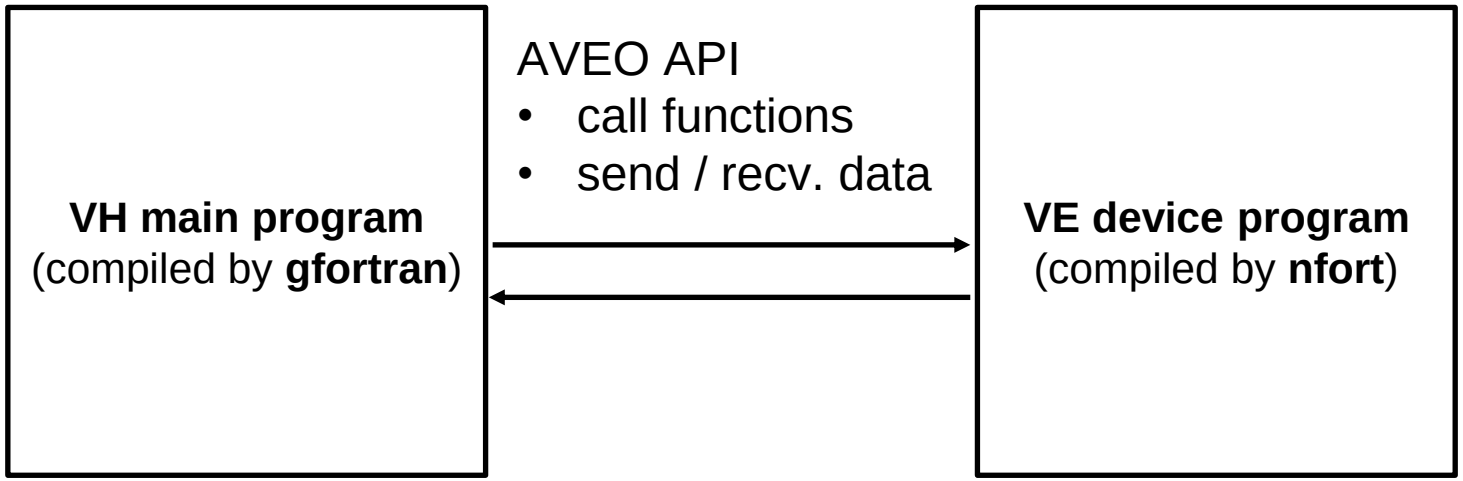


SX-Aurora TSUBASA Archtecture (VE model Type-10B)



AVEO (Another/Alternative/Awesome VE Offloading) API

- AVEO (Another/Alternative/Awesome VE Offloading) is data transfer between VH and VE API
- VE offloading; accelerator-style programming model for VE
  - AVEO can execute code on VE
    and can control the execution from VH main program.
- In this talk, we use only one VE



Agenda

- Introduction
- NEC SX-Aurora TSUBASA
- FrontISTR structural analysis on SXAT
  - SpMV and matrix storage format
  - Hybrid computing using AVEO
- Performance evaluation
- Conclusion

Block CRS (BCRS) format of FrontISTR (block size is 2x2)

11	12	13	14			17	18
21	22		24			27	28
31		33	34	35	36		
41	42	43	44		46		
		53		55	45	57	58
		63	64	65	66		68
71	72			75		77	78
81	82			85	86	87	88

Divide the matrix into three
to speed up preprocessing and stiffness matrix generation

- **U** Upper triangle block matrix
- **D** Diagonal block matrix
- **L** Lower triangle block matrix

Diagonal block

Value of non-zero blocks

D

11	12	21	22	33	34	43	44	. . .	77	78	87	88

VAL_U

13	14	0	24	17	18	27	28	. . .	57	58	0	68

INDEX_U

1				3				. . .	3			

PTR_U

0	2	3	4

Column block indices

Value indices where each row starts

※ Create the lower matrix in the same way as the upper matrix

SpMV stored in BCRS (block size is 2x2)

r and c are # of row and col of blocks

```
for(bi=0;bi<block_row;bi++){

    //diag. block matrix
    y[r*bi+0] += D[bi+0] * x[bi+0];
    y[r*bi+1] += D[bi+1] * x[bi+0];
    y[r*bi+0] += D[bi+2] * x[bi+1];
    y[r*bi+1] += D[bi+3] * x[bi+1];

    //Upper triangular block matrix
    for(bc=PTR_U[bi];bc<PTR_U[bi+1];bc++){
        bj   = INDEX_U[bc] * bnc;
        k = r * c * j;
        y[r*bi+0] += VAL_U[k+0] * x[bj+0];
        y[r*bi+1] += VAL_U[k+1] * x[bj+0];
        y[r*bi+0] += VAL_U[k+2] * x[bj+1];
        y[r*bi+1] += VAL_U[k+3] * x[bj+1];
    }

    //Lower triangular block matrix
    ...
}
```

Multithreading is possible for block row loops

Calculation for diagonal block matrix does not require indirect memory access

Calculation for Upper / Lower block matrix is:
- Requires indirect memory access for vector x
- The vector length cannot be long (only 4)

Multithreading can be expected, however it is not suitable for vectorization

JAD format of FrontISTR (block size is 2x2)

- It has only the off-diagonal block matrix as JAD.
- The diagonal block matrix is the same as the BCRS format.
- Sort row with # of non-zero elements contained in the row, and stored in the col. direction.

		13	14			17	18
			24			27	28
31				35	36		
41	42						
		53				57	58
		63	64				68
71	72			75			
81	82			85	86		

13	14	17	18				
81	82	85	86				
24	27	28					
31	35	36					
53	57	58					
66	64	68					
71	72	75					
41	42						

VALUE

13	81	24	31	53	36	71	41	14	82	27

▪ ▪ ▪ Val. of non-zero elements

JADORD

1	8	2	3	5	3	7	4

Row number before sorting

IAJAD

0	8	16	23	25

Value indices where each row starts

JAJAD

3	1	4	1	3	6	1	1	4	2	7

▪ ▪ ▪ Column indices

SpMV stored in JAD (block size is 2x2)

r and c are # of row and col of blocks

```
//diag. block matrix
for(ii=0;ii<N;ii++){
    int k = r*c*ii;
    y[k+0] += D[k+0] * x[k+0];
    y[k+0] += D[k+1] * x[k+1];
    y[k+1] += D[k+2] * x[k+0];
    y[k+1] += D[k+3] * x[k+1];
}
// Lower and Upper triangular block matrix
for(j=0;j<NZ;j++){
    for(i=IAJAD(j);i<IAJAD(j+1)-1;i++){
        int ixx = i - IAJAD(j) + 1;
        int k = r * c * i;
        w1[ixx] += VALUE[k+0]*x[JAJAD[k]*r+0]
        w1[ixx] += VALUE[k+1]*x[JAJAD[k]*r+1]
        w2[ixx] += VALUE[k+2]*x[JAJAD[k]*r+0]
        w2[ixx] += VALUE[k+3]*x[JAJAD[k]*r+1]
    }
}
for(ii=0;ii<N;ii++){
    y[r*ii+0] += w1[JADORD[ii]];
    y[r*ii+1] += w2[JADORD[ii]];
}
```

Calculation for diagonal block matrix does not require indirect memory access

Calculation for Upper / Lower block matrix is:

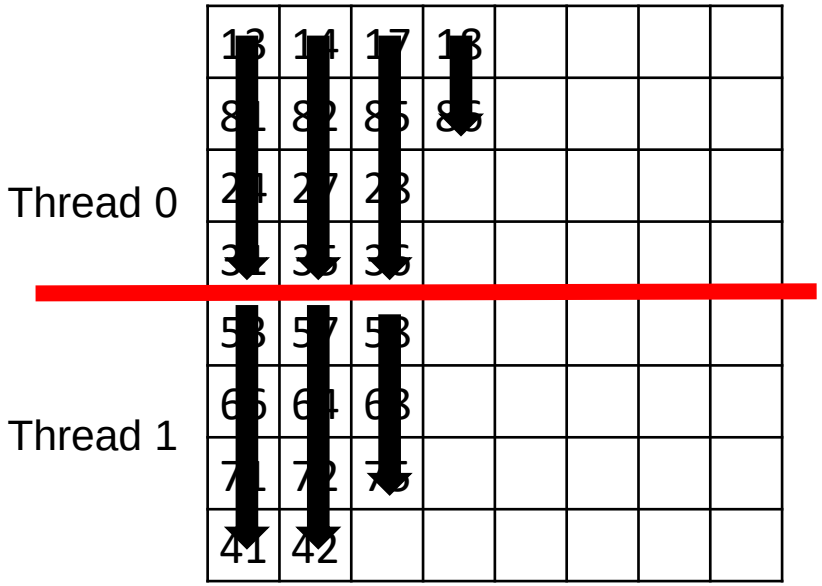
- The vector length can be increased in the col. direction.
- Indirect memory access is required for vectors x and y
- Difficult to multithread (described later)

Add the result to y before sorting using JADORD

Multi-threading of SpMV stored in JAD

- SpMV stored in JAD has a shorter vector length due to multi-threading

```
// Lower and Upper triangular block matrix
for(j=0;j<NZ;j++){
    # pragma omp parallel for ...
    for(i=IAJAD(j);i<IAJAD(j+1)-1;i++){
        int ixx = i - IAJAD(j) + 1;
        int k = r * c * i;
        w1[ixx] += VALUE[k+0]*x[JAJAD[k]*r+0]
        w1[ixx] += VALUE[k+1]*x[JAJAD[k]*r+1]
        w2[ixx] += VALUE[k+2]*x[JAJAD[k]*r+0]
        w2[ixx] += VALUE[k+3]*x[JAJAD[k]*r+1]
    }
}
```



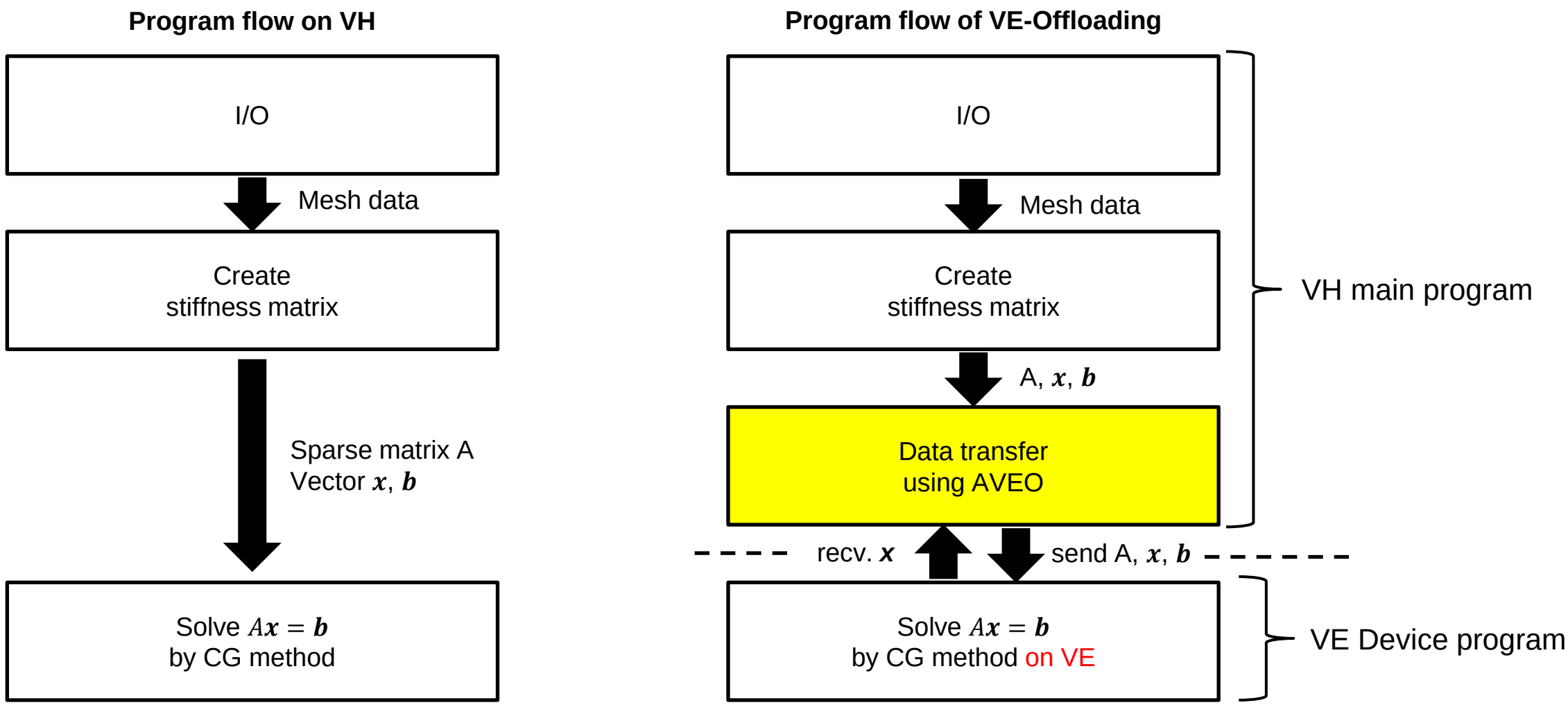
- **Multi-threading is possible without dependency**
- **As the number of threads increases, the vector length becomes shorter.**

VH and VE hybrid computing using AVEO

- The stiffness matrix generation is not suitable for VE
  - Involves many integer operations

- Compute only linear equation solver on the VE using the AVEO
- Achieve VE Offloading in the following 3 programs:
  1. The stiffness matrix A generation runs on the VH
     - Original code with switching if “enable VE”, compiled by “gfortran”
  2. Transfer matrix A, $\boldsymbol{x}$, and $\boldsymbol{b}$ to the VE
     - Program with AVEO, compiled by “gfortran”
  3. Solving $A\boldsymbol{x} = \boldsymbol{b}$ by CG method runs on the VE
     - Original code compiled by “nfort”
     - main is included, but everything is compiled by nfort. And call only CG function
     - No program changes required to make VE program
       - VE program can use general Fortran program without changes

The simulation flow of FrontISTR



Agenda

- Introduction
- NEC SX-Aurora TSUBASA
- FrontISTR structural analysis on SXAT
  - SpMV and matrix storage format
  - Hybrid computing using AVEO
- Performance evaluation
- Conclusion

Test bed (NEC SX-Aurora TSUBASA A300-4)

VH

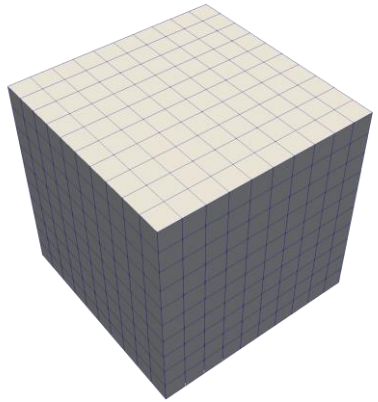
Model	Intel(R) Xeon(R) Silver 4108 CPU @ 1.80 GHz 8 core ╳ 2
Peak	460.8 GFLOPS ╳ 2
Memory	96 GB (B/W 127.8 GB/s)
OS	CentOS 7.7
Compiler (option)	gcc / gfortran 4.8.5 (-O3 -fopenmp)

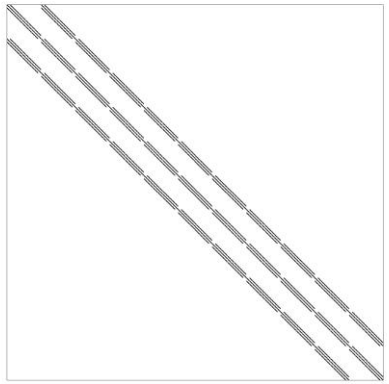
VE

Model	Type 10B (1.4 GHz, 8 core) ╳ 4 board
Peak	2.15 TFlops (DP) / 4.3 TFLOPS (SP) / board
Memory	48 GB (B/W 1,228 GB/s)
LLC	16 MB
Compiler (option)	ncc 3.1.0 / nfort 3.1.0 (-O3 -fopenmp -mvector)

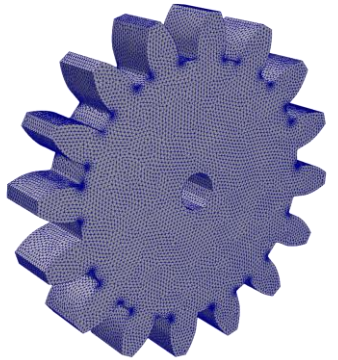
Test data

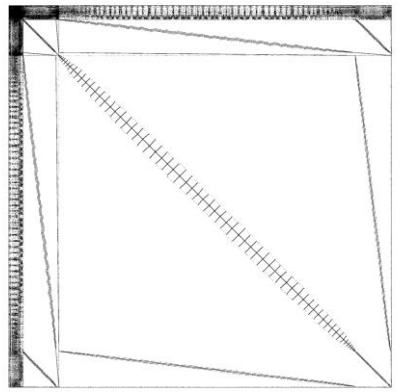
- Evaluate performance using the following two data





cube(N)
Cut the cube mesh into N x N x N
The matirx size can be changed arbitrarily





Gear16
1.9M dimensions
154M non-zero elements

Matrix conditions

- Evaluate SpMV execution time for the following 4 matrices
- SpMV stored in BCRS3x3 is the fastest in VH
- Compare the performance of SpMV stored in BCRS3x3 and JAD format

	N	nnz	nnz/row
cube(10)	3,993	268,119	67.1
cube(50)	397,953	30,986,559	77.9
cube(100)	3,090,903	245,438,109	79.4
Gear16	1,859,214	154,479,996	83.1

N # of Rows
nnz: # of non-zero elements

Elapsed time of SpMV [ms] (average vector length measured by profiler)

		cube(10)	cube(50)	cube(100)	Gear16
VH, BCRS3x3, 32 threads		3.4	26.9	202	165
VE BCRS3x3	1 thread	8.3 (10.7)	77.2 (12.5)	632.4 (12.7)	441.2 (11.2)
	8 threads	1.1 (10.7)	10.8 (12.5)	87.1 (12.7)	57.3 (11.2)
VE JAD	1 thread	0.6 (252.6)	4.2 (255.8)	38.2 (256.0)	19.9 (256.0)
	8 threads	0.5 (137.5)	1.2 (254.3)	11.2 (255.7)	7.5 (256.0)

Elapsed time of SpMV [ms] (average vector length measured by profiler)

		cube(10)	cube(50)	cube(100)	Gear16
VH, BCRS3x3, 32 threads		3.4	26.9	202	165
VE BCRS3x3	1 thread	8.3 (10.7)	77.2 (12.5)	632.4 (12.7)	441.2 (11.2)
	8 threads	1.1 (10.7)	10.8 (12.5)	87.1 (12.7)	57.3 (11.2)
VE JAD	1 thread	0.6 (252.6)	4.2 (255.8)	38.2 (256.0)	19.9 (256.0)
	8 threads	0.5 (137.5)	1.2 (254.3)	11.2 (255.7)	7.5 (256.0)

- SpMV stored in BCRS3x3 is 2-3x faster than VH for all cases in 8 threads
- The vector length of SpMV stored in BCRS3x3 is short (about 10)
  - The vectorization works only within blocks (3x3=9)

Elapsed time of SpMV [ms] (average vector length measured by profiler)

		cube(10)	cube(50)	cube(100)	Gear16
VH, BCRS3x3, 32 threads		3.4	26.9	202	165
VE BCRS3x3	1 thread	8.3 (10.7)	77.2 (12.5)	632.4 (12.7)	441.2 (11.2)
	8 threads	1.1 (10.7)	10.8 (12.5)	87.1 (12.7)	57.3 (11.2)
VE JAD	1 thread	0.6 (252.6)	4.2 (255.8)	38.2 (256.0)	19.9 (256.0)
	8 threads	0.5 (137.5)	1.2 (254.3)	11.2 (255.7)	7.5 (256.0)

- SpMV stored in JAD is **2.2-7.8x** faster than that stored in BCRS3x3
- SpMV on VE is **7-22x** faster than that on VH
  - About 43.8 GFLOPS (2% of Peak) for the cube(100)

Elapsed time of SpMV [ms] (average vector length measured by profiler)

		cube(10)	cube(50)	cube(100)	Gear16
VH, BCRS3x3, 32 threads		3.4	26.9	202	165
VE BCRS3x3	1 thread	8.3 (10.7)	77.2 (12.5)	632.4 (12.7)	441.2 (11.2)
	8 threads	1.1 (10.7)	10.8 (12.5)	87.1 (12.7)	57.3 (11.2)
VE JAD	1 thread	0.6 (**252.6**)	4.2 (**255.8**)	38.2 (256.0)	19.9 (256.0)
	8 threads	0.5 (**137.5**)	1.2 (**254.3**)	11.2 (255.7)	7.5 (256.0)

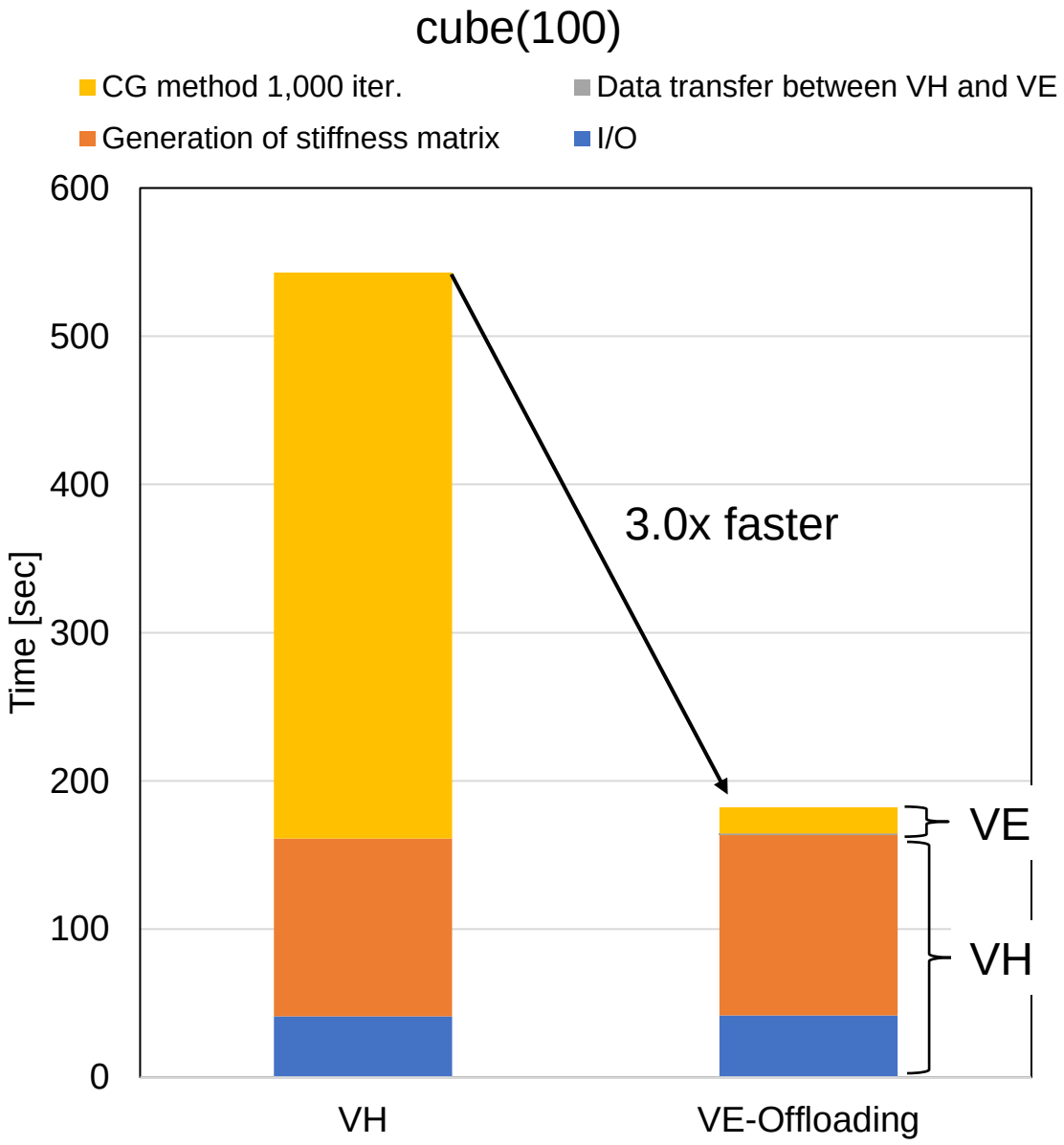
- As the number of threads increases, the vector length becomes shorter.
- Multithreading is not a problem for practically-sized matrices
  - In the cube (10), the vector length is decrease by half.
    - Still 2.2x faster than that stored in BCRS3x3
  - In the cube (50), Vector length does not decrease
    - The cube (50) has enough rows to vectorization
- The effect of VE acceleration is remarkable

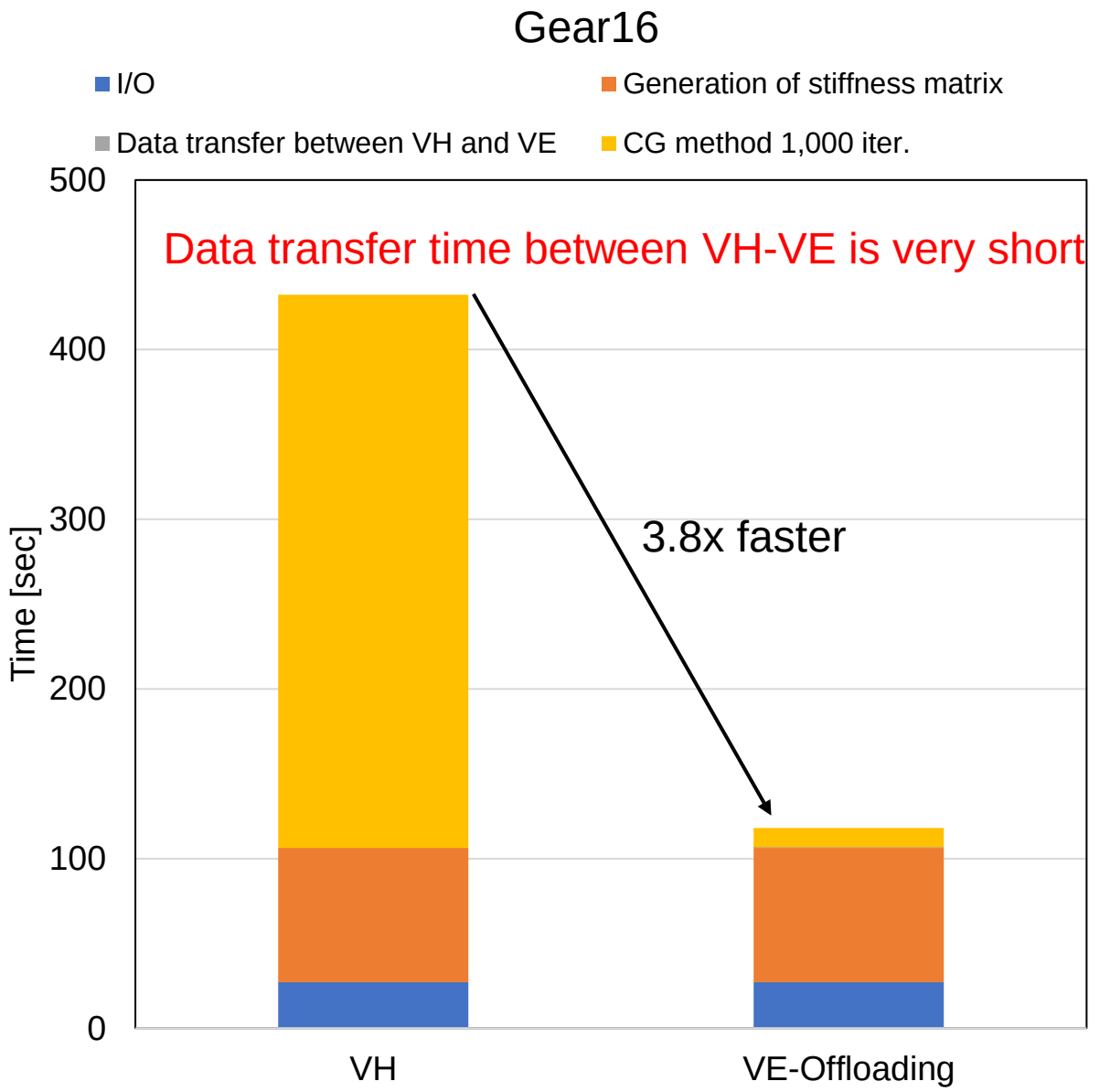
Acceleration of the overall FEM program using AVEO

- Evaluate the VE accelerated FrontISTR in terms of the following four parts:
  (1) I / O on VH
  (2) The generation time of stiffness matrices on VH
  (3) The data transfer time from VH to VE using AVEO
  (4) The Solving time of linear equations by the CG method on VE

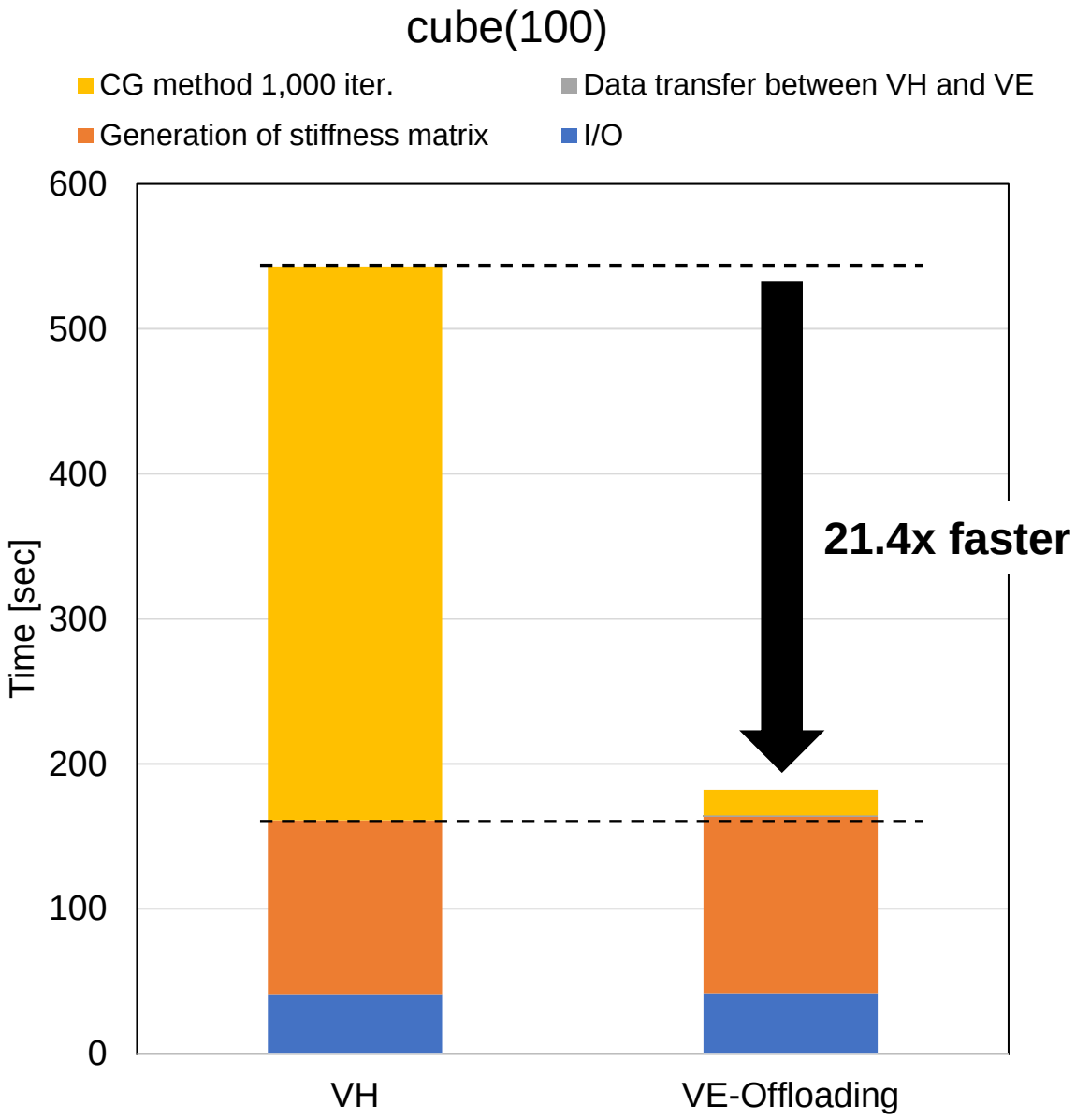
- Target data are cube(100) and Gear16
- # of thread
  - VE: 8
  - VH: 32

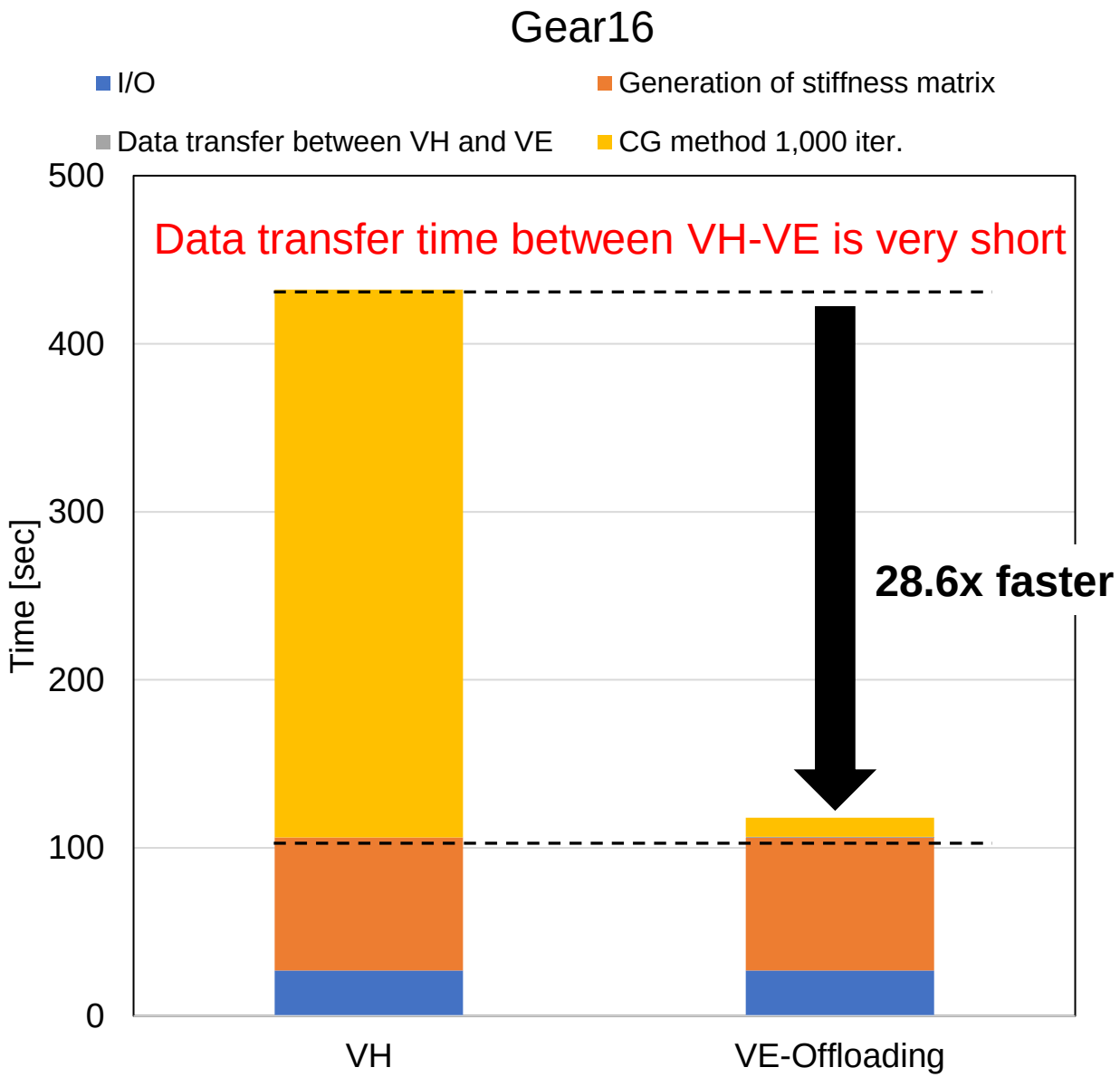
Elapsed time of the overall FEM program using AVEO (1,000 CG iter.)



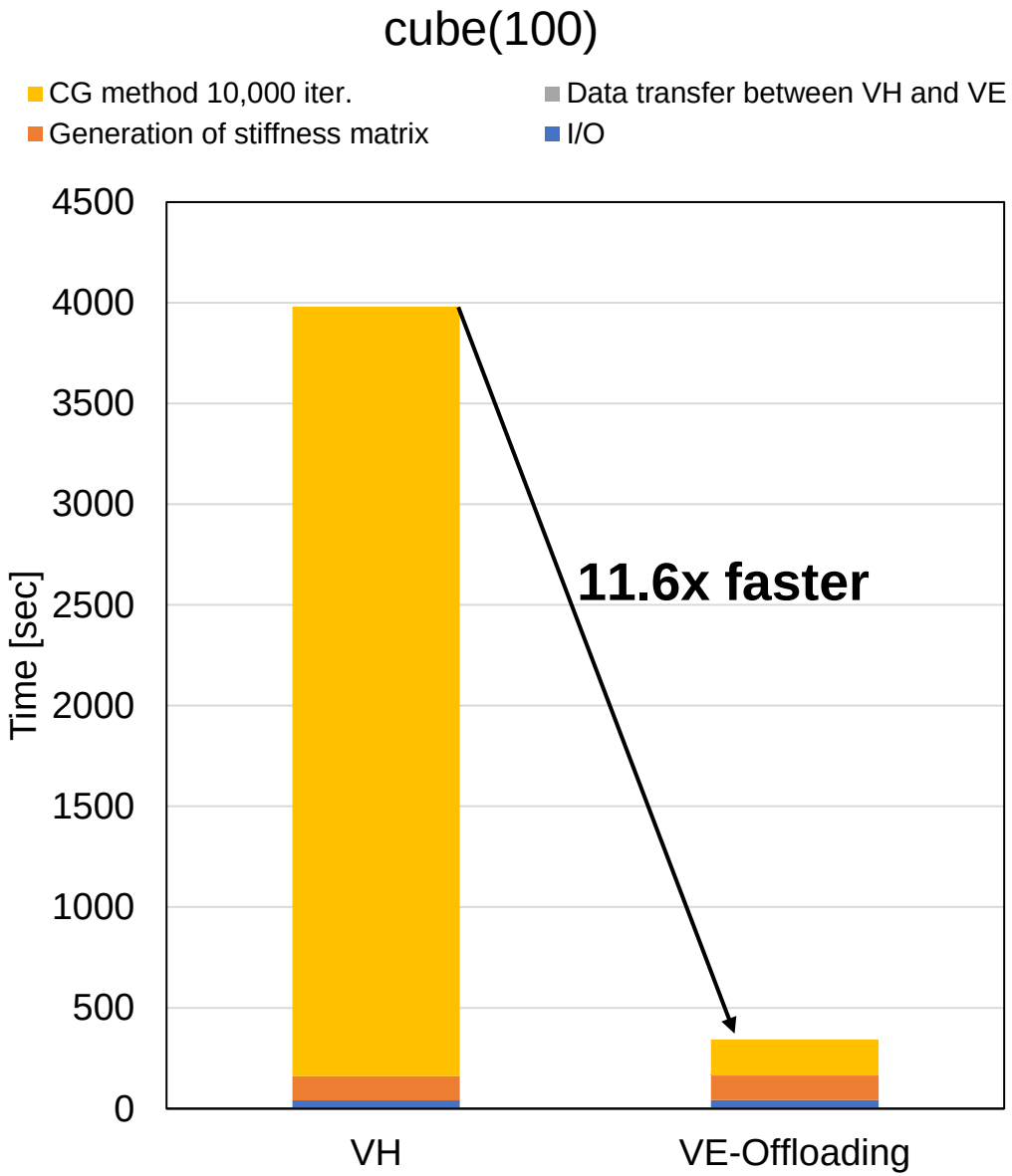


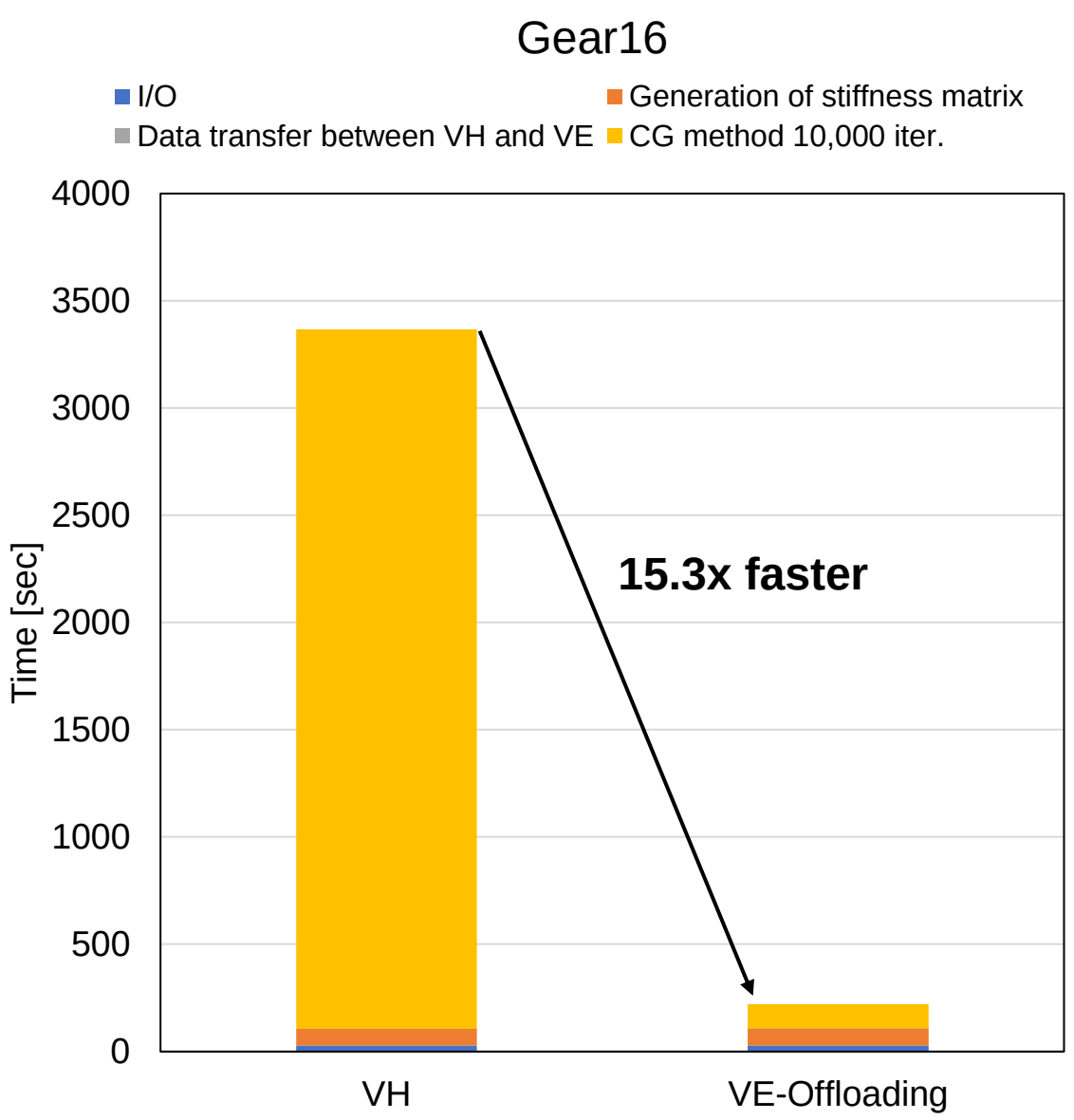
Elapsed time of the overall FEM program using AVEO (1,000 CG iter.)





Elapsed time of the overall FEM program using AVEO (10,000 CG iter.)





Conclusion

- Acceleration of Open-source FEM Software FrontISTR on VE

- Evaluate the performance of SpMV stored in BCRS3x3 and JAD format
- OpenMP parallelization important to take advantage of wide HBM memory B/W
- In liner equation solver, the effect of VE acceleration is remarkable
  - SpMV stored in JAD on VE is 17-22x faster than that stored in BCRS3x3 on VH
  - SpMV stored in JAD is 2.2-7.8x faster than that in BCRS3x3 on VE
    - Authentic optimization for vector architecture is effective

Conclusion

- Hybrid implementation of VH and VE using AVEO
  - VE Offloading is 3.0-3.8x faster than VE at 1,000 CG iter.
  - VE Offloading is 11.6-15.3x faster than VE at 10,000 CG iter.
- The speed-up effect depends on # of CG iter.
  however VE-Offloading can always expect 3-10x faster than VH
  - Performance of SpMV (CG method) depends on matrix size
  - # of iter. depends on matrix condition
- VE is never slower than VH for practically-sized matrices
  - Data transfer time is very short relative to total time

Future work

- From these results, good performance on single VE
- We will implement multi-VE or MPI for large-scale simulations