

# A simplified AINV method based on nonzero element positions of a coefficient matrix

2022/01/12

HPC Asia 2022

Kengo Suzuki<sup>1)</sup>, Takeshi Fukaya<sup>1)</sup>, Takeshi Iwashita<sup>1)</sup>

1) Hokkaido University





# Table of contents

---

**1. Background**

**2. Outline**

**3. AINV algorithm**

**4. Proposed algorithm : PS-AINV  
algorithm**

**5. Numerical results**

**6. Conclusions**

# 1. Background (1/2)

---

## ⦿ Problem

Linear systems of equations:  $Ax = b$ ,

where,  $A \in \mathbb{R}^{n \times n}$  is sparse, symmetric, and positive definite.

## ⦿ Solving method

Preconditioned conjugate gradient (PCG) method

## ⦿ In recent years...

- Graphic processing units (GPUs) have been used to efficiently execute the PCG method.
- To exploit the GPUs' potential for massive data processing, the preconditioning method is desired to have highly parallelism.
- Many preconditioning methods that utilize the GPUs have been proposed.

For example,

- IC preconditioning with the AMC ordering method,
- IC preconditioning with Jacobi method, and
- Sparse approximate inverse preconditioning.

# 1. Background (2/2)

---

## Preliminary tests

- We compared three preconditioning methods that are suitable for GPUs:
  - IC preconditioning with AMC ordering
  - IC preconditioning whose forward/backward substitutions are approximately performed by Jacobi method
  - Approximate inverse (AINV) preconditioning [1]
- We implement all the PCG solvers so that **each preconditioner is constructed on a CPU** and **each PCG method is executed on a GPU**.

## Brief results

- The AINV preconditioned solver is (almost) the best of the three solvers in terms of execution time for the PCG method (on a GPU).
- However, **AINV preconditioning takes more time to construct its preconditioner (on a CPU)** than the other two preconditioning methods do.
- Similar results are shown in [3].

## 2. Outline

---

### ● Purpose of this study

- To make AINV preconditioning more attractive.
  - Once the AINV preconditioner is constructed, the AINV preconditioned CG solver runs sufficiently fast on GPUs.
  - However, a part of constructing the AINV preconditioner (AINV algorithm) relatively takes a long time and is needed to be improved.

### ● Methods

- We propose a new version of the AINV algorithm.
  - We introduce a simplification that is based on nonzero positions of  $A$  to the AINV algorithm.
  - The simplification will reduce the computational complexity, computational time, and memory usage of the AINV algorithm.
- We evaluate the performance of the proposed algorithm.
  - How much faster will the AINV algorithm be?
  - How much will the performance of the PCG method change?
  - What about the performance of the whole solver?

# 3. AINV algorithm (1/2)

## AINV algorithm

An algorithm to (approximately) calculate  $Z$  and  $D$  that satisfy the following equation :  $A^{-1} \approx ZD^{-1}Z^T$ .

## Dropping method

- Dropping method is...
  - A kind of approximation.
  - Used to create a sparsity of  $z_j$ .
  - Usually based on the magnitudes of elements of  $z_j$  [1, 2].
- In this study
  - We use a dropping method in which the elements of  $z_j$  whose magnitudes are less than a predefined threshold are removed.
  - We set the threshold to 0.1

The AINV algorithm

```
1 : for  $i = 1, \dots, n$  do  
2 :    $z_i = e_i$   
3 :   for  $j = 1, \dots, i - 1$  do  
4 :      $p_j = a_i^T z_j$   
5 :      $z_i = z_i - \frac{p_j}{d_j} z_j$   
6 :     Drop some elements  
       from  $z_i$ .  
7 :   end for  
8 :    $d_j = a_i^T z_j$   
9 : end for
```

# 3. AINV algorithm (2/2)

## Implementation

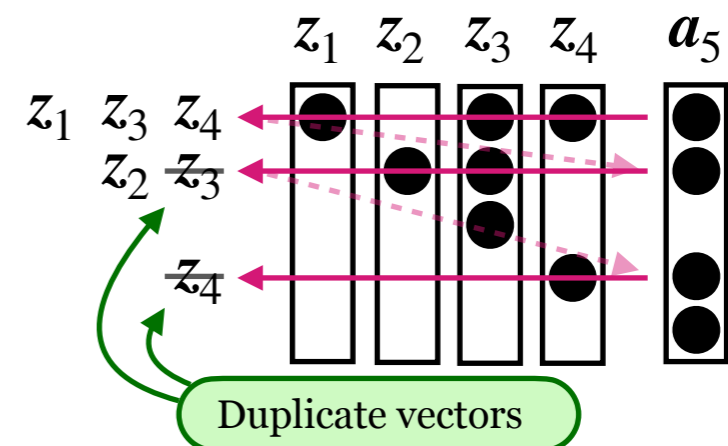
- To efficiently execute...
  - We should store the column vectors  $\mathbf{a}$  and  $\mathbf{z}$  in a compressed format.
  - If  $p_j = 0$ , we should omit the calculation of lines 4-6 for corresponding  $j$ .
- Thus, we should consider line 3 as “for  $j = 1, \dots, i - 1 \wedge p_j \neq 0$  do.”

## How to judge whether $p_j \neq 0$

- Scan the vectors of  $Z$  in row-major order as shown in the figure to the right.
- In order to do that, we have to...
  - Use additional arrays that store the row vectors of  $Z$  to scan in row-major order.
  - Update these row vectors, not only the column vectors.
  - Avoid finding duplicate  $z_j$  during the scanning.
- **These operations take a long time.**

The AINV algorithm

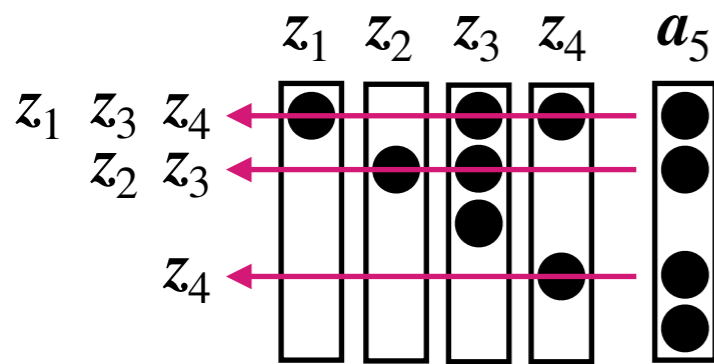
```
1 : for  $i = 1, \dots, n$  do
2 :    $z_i = e_i$ 
3 :   for  $j = 1, \dots, i - 1$  do
4 :      $p_j = \mathbf{a}_i^T z_j$ 
5 :      $z_i = z_i - \frac{p_j}{d_j} z_j$ 
6 :     Drop some elements
       from  $z_i$ .
7 :   end for
8 :    $d_j = \mathbf{a}_i^T z_j$ 
9 : end for
```



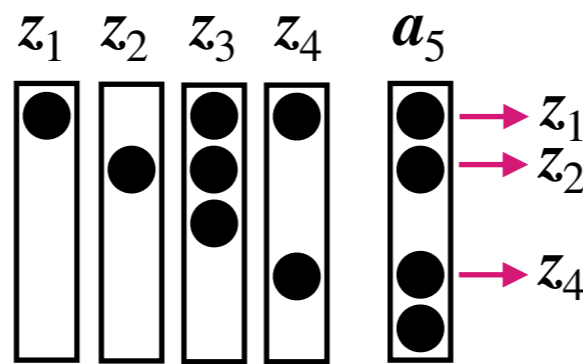
# 4. Proposed algorithm (1/2)

## Position-based Simplified AINV algorithm : PS-AINV algorithm

- In the PS-AINV algorithm...
  - $a_{ji} \neq 0$  is judged instead of  $p_j \neq 0$ .
    - If  $a_{ji} \neq 0$ ,  $p_j$  is unlikely to be 0 because the initial value of  $z_j$  is  $e_j$  (and the diagonal elements of  $A$  are nonzero).
    - Conversely, If  $a_{ji} = 0$ ,  $p_j$  is approximated by 0.
  - The algorithm runs as shown to the right.
- The following two illustrations show the judging procedures in the AINV algorithm and the PS-AINV algorithm.



The procedure in the AINV algorithm



The procedure in the PS-AINV algorithm

The PS-AINV algorithm

```

1 : for  $i = 1, \dots, n$  do
2 :    $z_i = e_i$ 
3 :   for  $j = 1, \dots, i - 1$ 
4 :      $p_j \neq 0$   $\wedge a_{ji} \neq 0$  do
5 :        $p_j = a_i^T z_j$ 
6 :        $z_i = z_i - \frac{p_j}{d_j} z_j$ 
7 :       Drop some elements from  $z_i$ .
8 :     end for
9 :   end for
10:   $d_j = a_i^T z_j$ 
11: end for
    
```

Only scanning  $a_i$  is needed. Additional arrays and operations are NOT needed.



## 4. Proposed algorithm (2/2)

### Influence on the performance

- Thanks to the simplification, the PS-AINV algorithm is expected to run faster than the standard AINV algorithm.
  - Let  $\Delta T_{Pre} = (\text{The time taken for AINV}) - (\text{The time taken for PS-AINV})$ .
- However, the performance of the PS-AINV-PCG method might not be better than that of the AINV-PCG method.
  - This is because the PS-AINV preconditioner is likely to be a more rough approximation of  $A^{-1}$  because of the simplification.
  - Let  $\Delta T_{PCG} = (\text{The time taken for PS-AINV-PCG}) - (\text{The time taken for AINV-PCG})$ .

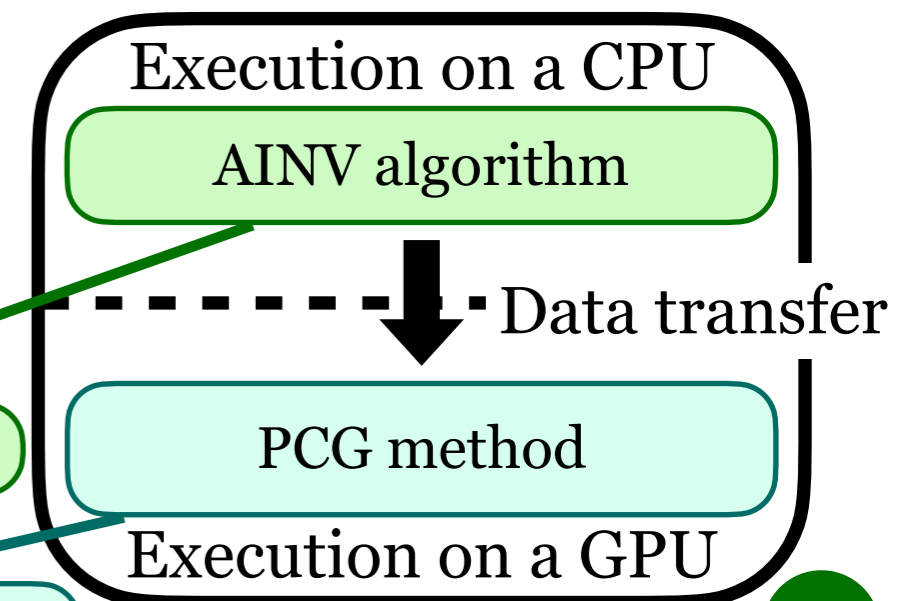
### Therefore,

When  $\Delta T_{Pre} > \Delta T_{CG}$ , the performance of the whole solver, which is the sum of the preconditioner construction and the PCG iterations, is increased.

This part will become faster.

This part might become slower.

Flowchart of the AINV-PCG solver



# 5. Numerical results (1/3)

## Conditions of the tests

We...

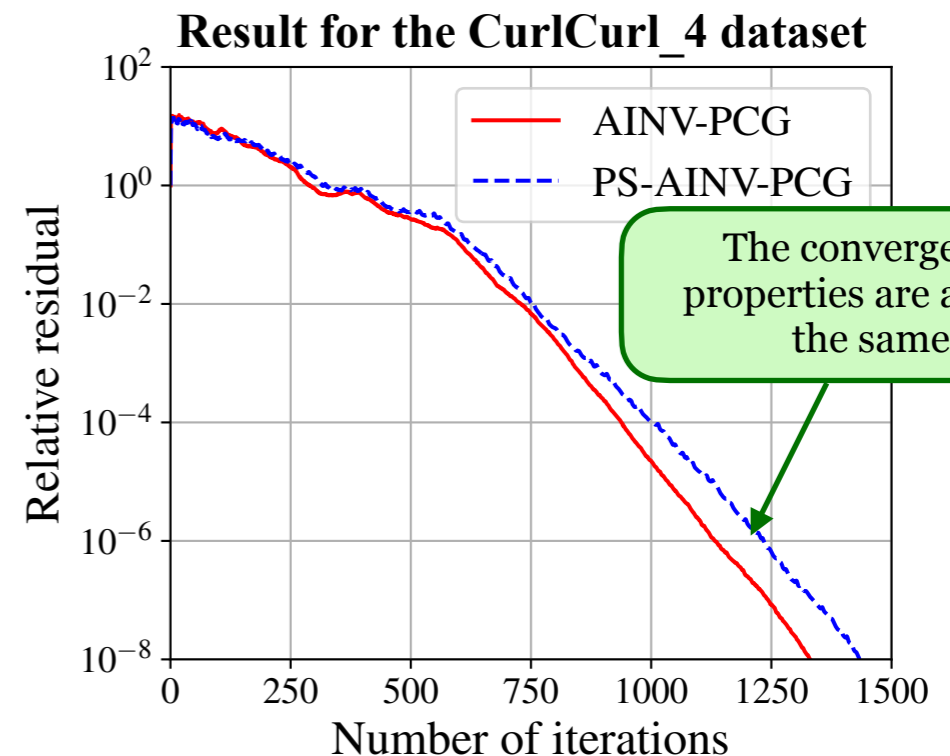
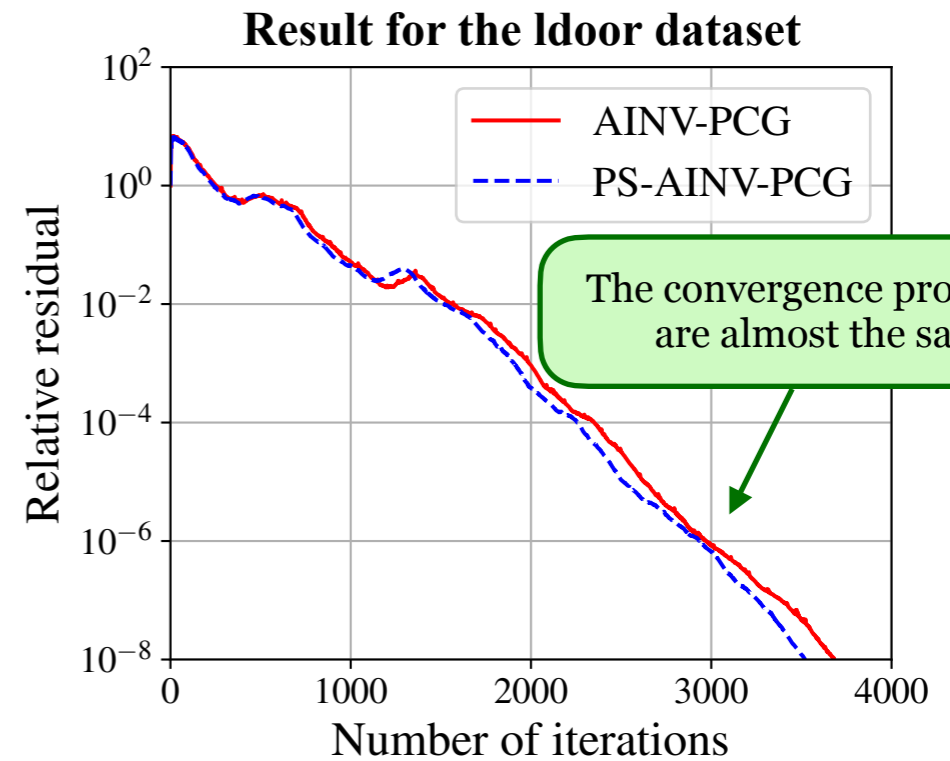
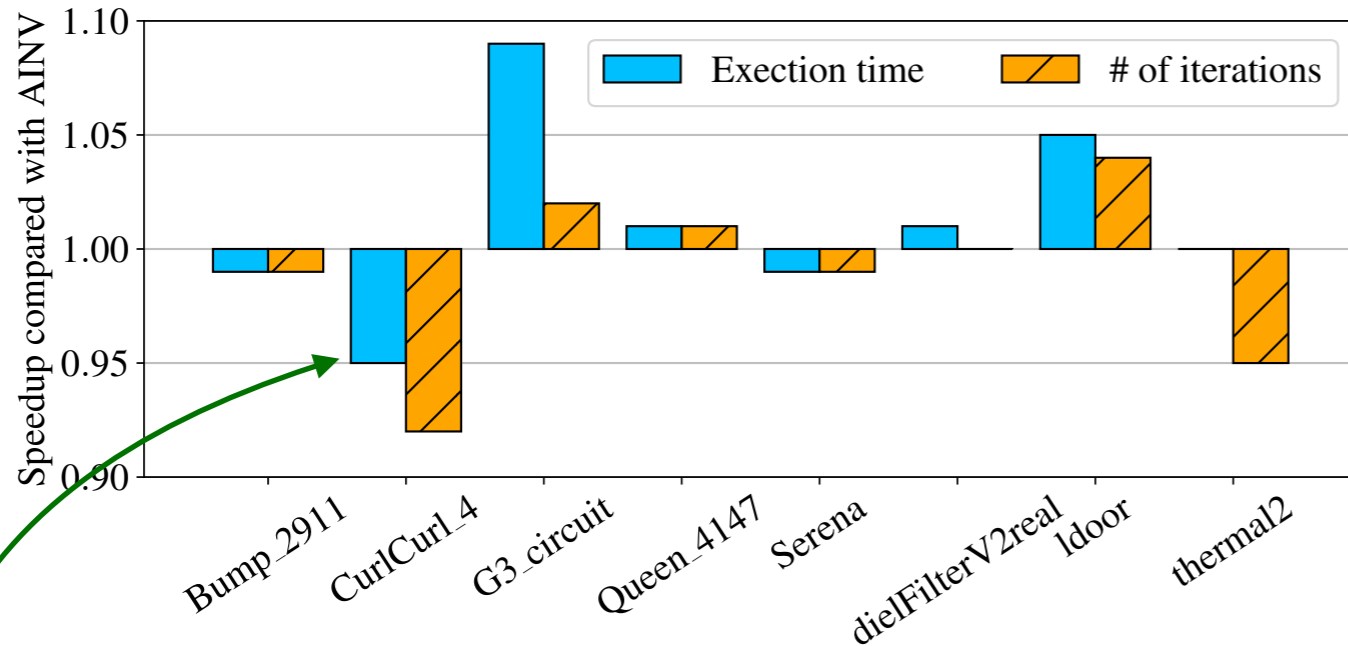
- Used a computer equipped with CPUs and GPUs.
  - ▶ CPUs : Intel Xeon Gold 6230 (Cascade Lake) x2
  - ▶ GPUs : NVIDIA Tesla V100 x4
- Executed the AINV algorithm and PS-AINV algorithm on a CPU.
- Executed the PCG method on a GPU.
- Set the convergence criterion as  $\|\mathbf{b} - A\mathbf{x}\|_2 / \|\mathbf{b}\|_2 < 10^{-8}$
- Used a diagonally shifted matrix  $A' = \{a'_{ij} = a_{ij} (i \neq j), a'_{ij} = \alpha a_{ij} (i = j)\}$  only in each preconditioner construction algorithm to prevent breakdown.

## Data sets

Name	Dimension	# nonzero	# nnz / row	Field of problems	$\alpha$
Bump_2911	2,911,419	127,729,899	43.87	2D/3D Problem	1.2
CurlCurl_4	2,380,515	26,515,867	11.14	Model Reduction	1.2
G3_circuit	1,585,478	7,660,826	4.83	Circuit Simulation	1.0
Queen_4147	4,147,110	316,548,962	76.33	2D/3D Problem	1.3
Serena	1,391,349	64,131,971	46.09	Structural Problem	1.2
dielFilterV2real	943,695	77,651,847	82.28	Structural Problem	1.2
ldoor	952,203	42,493,817	44.62	Structural Problem	1.3
thermal2	1,228,045	8,580,313	6.99	Thermal Problem	1.0

# 5. Numerical results (2/3)

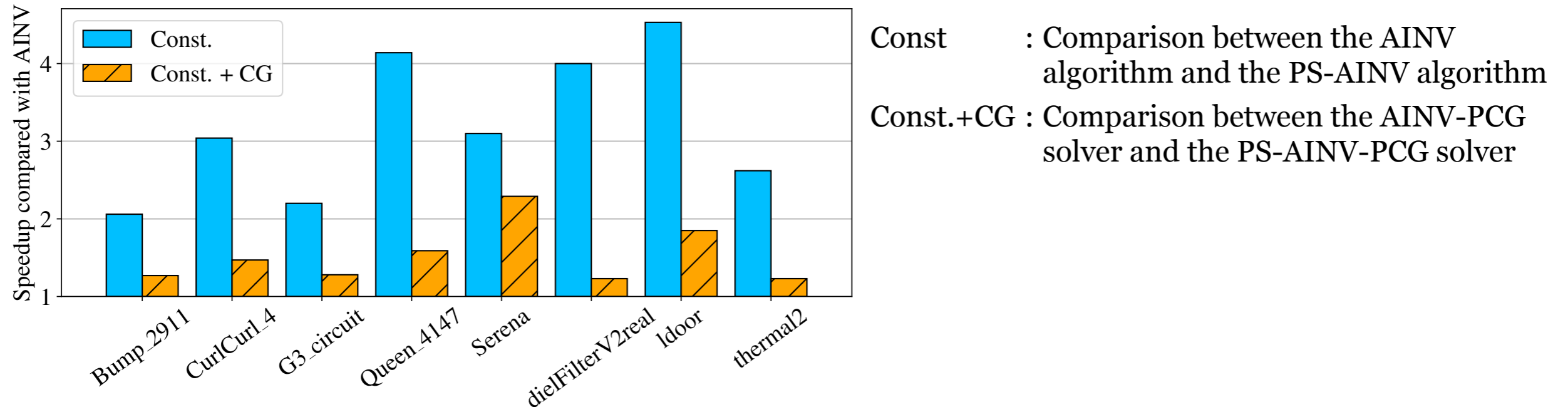
## Comparisons with respect to the performance of the PCG method





- These figures show how much the simplification influences the PCG method.
- The execution time and number of iterations are almost the same between AINV-PCG and PS-AINV-PCG.
- Even in the worst case, PS-AINV-PCG is only about 5% worse than AINV-PCG in terms of the execution time.

# 5. Numerical results (3/3)

## Comparisons with respect to the execution time of the construction algorithms and the whole solvers



- This figure shows how much faster the PS-AINV algorithm and the PS-AINV preconditioned CG solver are, compared with the AINV algorithm and the AINV preconditioned CG solver, respectively.
- The PS-AINV algorithm runs faster than the standard AINV algorithm for all the test datasets. 
- The PS-AINV-PCG solver also achieved superior performance compared with the AINV-PCG solver because, as we mentioned on the previous page, the effect of the PS-AINV preconditioner on the CG method is almost the same as that of the AINV preconditioner. 

# 6. Conclusions

---

## ● We proposed a PS-AINV algorithm.

- The PS-AINV algorithm...
  - Is derived by simplifying the AINV algorithm based on nonzero positions of  $A$ .
  - Is expected to run faster than the AINV algorithm because of the simplification.
- However, the PS-AINV preconditioner might be poor performance compared with the AINV preconditioner.
- When the reduced time in the preconditioner construction part is greater than the increased time in the PCG execution part, the performance of the overall solver will increase.

## ● Numerical results show that...

- The PS-AINV algorithm runs faster than the AINV algorithm for all the test datasets.
- The performance of the PCG execution part is almost the same between AINV preconditioning and PS-AINV preconditioning.
- The overall solver performance also increases for all the test datasets.

## ● Future works

- We will extend this method to asymmetric version of the AINV algorithm and evaluate its performance.
- We will apply other dropping methods to the PS-AINV algorithm.



## 6. References

---

- [1] M. Benzi and C.D. Meyer and M. Tũma. “A sparse approximate inverse preconditioner for the conjugate gradient method.”, *SIAM J. Sci. Comput* 17 (1996): 1135–1149.
- [2] S. Fujino and Y. Ikeda, "An improvement of SAINV and RIF preconditionings of CG method by double dropping strategy," *Seventh International Conference on High Performance Computing and Grid in Asia Pacific Region* (2004): 142–149.
- [3] M. Benzi and M. Tũma. “A comparative study of sparse approximate inverse preconditioners”, *Applied Numerical Mathematics* 30 (1999): 305–340.

**Thank you!**

