Efficient and Robust Parallel ILU Preconditioners for Quantum Eigenvalue Problems in ppOpen-HPC/ESSEX-II

Masatoshi Kawai, Akihiro Ida and Kengo Nakajima
Supercomputing Research Division, ITC
SIAM PP 2018, Tokyo, Japan
Mar/8/2018
Index

- Introduction
- Massively parallelized ILU preconditioner with robustness
- Parameter research of the ILU
- Conclusion
Motivation

Unraveling electrical properties of special materials by solving generalized eigenvalue problems

Graphene
Several kinds of allotropes

Topological insulator
A new material which has special properties
Inside: insulation Outside: conduction

Source: Nature
Sample of the topological insulator (SmB6)

Source: Telescope magazine

Source: The electronic properties of graphene
Solvers for generalized eigenvalue problems

**Focusing on Sakurai-Sugiura(SS) or FEAST method**

→ The SS and FEAST methods are obtaining eigenvalues in arbitrary area.

- SLEs are derived from each integral point in the SS or FEAST method.
  → Needed a solver for the SLEs \((A_z x = b)\)

- Properties of the matrix
  \[ A_z := z_i B - A : \quad B = I \]
  ✓ ill conditioned
  ✓ Large scale

- Requires of the solver
  - Robustness
    → Regularized ILU preconditioner
  - Massively parallelism
    → Hierarchical multi-coloring
Index

- Introduction
- Massively parallelized ILU preconditioner with robustness
- Parameter research of the ILU
- Conclusion
For robustness

Applying 2 regularization methods for ILU preconditioner
→ For robustness and improving convergence

- Blocking technique (Regularization①)
  - Applying the incomplete decomposition to a block matrix
    1. More robustness because of including non-small off-diagonals
    2. Better convergence ratio because of allowing more fill-ins

- Diagonal transformation (Regularization②)
  - Adding constant value $\alpha$ to the diagonal elements
  - Directly method to make the diagonally dominant matrix
Conditions and target problems for numerical analysis

- Krylov subspace method: COCG
- \( b = A_Z x \quad x^T = \text{random}(\min = 1, \max = 10) \)
- Iteration is stop if the number of iteration reach to DoF or relative residual fills the requirement \( \left\| \frac{r^k}{r^0} \right\|_2 \leq 10^{-7} \)

- Target problems
  - 128 data sets
    - Graphene model
      - DoF = 1K, 130K, 1.3M, 13M
      - 16 shift values
        - \( 4 \times 16 = 64 \) data sets
    - Topological insulator model
      - DoF = 1K, 30K, 102K, 1M
      - 16 shift values
        - \( 4 \times 16 = 64 \) data sets

Shift : Denotes shift values \( z \) of \( zB - A = A_z \)
Diagonal shifting : Denotes the regularization of preconditioning (proposed method)
Numerical evaluation

We solved all target problems by applying two regularization methods:

- Diagonal shifting $= (0.0, 1.0)$
- The size of block is 64
Proposing parallelization method for multi-coloring algorithms

→ Needed for solving the large scale SLEs on massively parallel systems

- Multi-color ordering is used for parallelize ILU preconditioner.
  - The effect of ILU preconditioner depend on a result of the multi-coloring
  - Multi-coloring algorithms are not parallelized.

```
Multi-coloring
Block IC decomposition
do itr = 1, DoF
   dot_product
   q = \tilde{A}^{-1}r \text{(preconditioning)}
Matrix Vector Multiplication
endo
```

Still sequentially
Parallelizing with multi-coloring
Parallelizing easily
Parallelization of multi-coloring algorithms

Proposing hierarchical parallelization for the multi-coloring algorithms → For supporting any algorithms.

Existing reordering algorithms

■ Level-set ordering
  • Lexicographic
  • Breadth First Search
  • Cuthill-McKee (CM)
  • Reverse CM

■ Independent set (Multi-color) ordering
  • Greedy
  • Algebraic Multi-coloring (AMC)
  • Block AMC

1. The best algorithm depends on applications.
2. We need the robustness for the target problems.

Need a versatile parallelization method which is not changing properties
Parallelization for multi-coloring algorithms

Proposing hierarchical parallelization for multi-coloring algorithms

→ Versatile method

**Step 1**

1. Each process separates elements to some parts.
2. The master process gathers the separates parts.

**Step 2**

3. The master process creates a new graph.
4. Master process colors the graph with any algorithms

**Step 3**

5. The master process scattering the coloring results.
   → All process gets colored areas.

**Step 4**

6. All process colors elements parallely based on the colored area.
Evaluations of the numerical analysis 2/2

- Oakleaf-FX 128～4800 nodes
  - Node specifications
    - SPARC64™ IXfx 16 cores
    - 32 GiB
  - Network specifications
    - Tofu

- The parameter of IC preconditioner
  - Block size = 4
  - Diagonal shifting = 100.0d0

- Coloring algorithm for test
  - Hierarchical parallelized AMC(10)

- Target problem
  - Graphen32,768x16,384 ≒ 500M DoF
  - ≒ 7G Non-zero elements (Real values)
  - Needed more than 4.7 GiB memory for coloring!
Performance on graphene problem

Numbers of iterations is almost same.

Performance of the IC preconditioned CG is good.
Index

- Introduction
- Massively parallelized ILU preconditioner with robustness
- Parameter research for the ILU
- Conclusion
Best parameter of the Block ILU preconditioner

The size of block has the biggest impact in the performance of the ILU.

Parameter

- Value of diagonal shifting
- Size of block
- Number of parallelism

- Convergence
- Computational time per one iteration
The size of block has the biggest impact in the performance of the ILU.

- Value of diagonal shifting
- Size of block
- Number of parallelism

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Convergence</th>
<th>Computational time per one itr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagonal shifting</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Size of block</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Number of parallelism</td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>

We have to find the best size of block for each problem.
Strategy to find the best size

We try to find the best size of block in the few iterations. ← It is difficult to find it without any trials.

Brief implementation of the FEAST

<table>
<thead>
<tr>
<th>Do 1, till converge eigenvalues....</th>
<th>Do $i = 1$, number of shifts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solve $(z_iI - A)x = r$</td>
<td>Enddo</td>
</tr>
<tr>
<td>Enddo</td>
<td></td>
</tr>
</tbody>
</table>

Number of solving equations

= “Number of shifts” * “Number iterations of outside”

- Measuring the computational time per iteration is easy
  - There are no differences of non-zero patterns of the $(z_iI - A)$ among each shift.

- The convergence depends mainly on the $z_i$.
  1. Looking for the best size of block with each $z_i$ is the easiest.
     → We need many iterations of outside loop.
  2. We try to find it with fewer iterations.
Relationships among convergence, block size and shift

Relationship between the block size and the convergence is similar for any $z_i \rightarrow$ Shifts values are close on imaginary plane.

On each line, the shift value is different.
Relationships among convergence, block size and shift

Relationship between the block size and the convergence is similar for any $z_i \rightarrow$ Shifts values are close on imaginary plane.

On each line, the shift value is different. Y-Axis is calculated as “computational time with each block size / block size 4” on each shift.

We can apply improvement ratios of the convergence to any $(z_i I - A)$.

→ The convergence ratio with block size 32 is twice as fast as block size 4 on shift data $z_i$, it is same on the other shifts
Method to find the best block size.

We can find the best size of block with “number of shift data” + “number of sampling points of block size” times trials.

The target range of the previous result is small.
- $5 \times 10^{-2} < \text{Real part} < 5 \times 10^{-2}$
- $3 \times 10^{-3} < \text{Imaginary part} < 5 \times 10^{-2}$

We verified the same relationship on a larger target range and the other model.

To find the best size of block,
1. Solving the equations on all shifts with block size 4
   → Measuring the computational time per one iteration and convergence
2. Solving the equations with all block size (4, 8, 16, 32, 64).

After above steps, we can use the suggested best size of the block on each iterations.
Method to find the best block size.

We can find the best size of block with “number of shift data” + “number of sampling points of block size” times trials.

The target range of the previous result is small.
- $-5 \times 10^{-2} < \text{Real part} < 5 \times 10^{-2}$
- $3 \times 10^{-3} < \text{Imaginary part} < 5 \times 10^{-2}$

We verified the same relationship on a larger target range and the other model.

To find the best size of block,
1. Solving the equations on all shifts with block size 4
   → Measuring the computational time per one iteration and convergence
2. Solving the equations with all block size (4, 8, 16, 32, 64).

After above steps, we can use the suggested best size of the block on each iterations.

I’m sorry, there is no evaluations.... This is a idea.
Index

- Introduction
- Massively parallelized ILU preconditioner with robustness
- Parameter research of the ILU
- Conclusion
Introduction the parallel ILU preconditioner for eigenvalue problems of quantum systems
- The regularizations for ILU preconditioner
- The hierarchical parallelization for the multi-coloring

We discussed to find the best size of block

Future works
- Numerical evaluations
- Checking the trends on more shifts and models
  - I checked on 2 models, 4 different DoF, 2 types of shifts
- The effect of matrix $B$ and right hand vector $r$
Publication of the parallelized ILU

We have implemented our ILU preconditioner in the PHIST.

PHIST is developed by ESSEX-II member of SPPEXA project. By using the PHIST, user can choose mathematical kernel libraries. Implemented ILU preconditioner only supports symmetric real values. Complex values will support in the near future.

http://ppopenhpc.cc.u-tokyo.ac.jp/ppopenhpc/

For the Fortran user, we publish pk-Open-SOL on the ppOpen-HPC web page.

git@bitbucket.org:essex/phist.git
Thank you for your kind attention
Evaluating differences between the sequential and the parallelized multi-coloring on the convergence and the performance

- **Reedbush-u 32 nodes**
  - Node specifications
    - Intel Xeon E5-2695v4 x 2socket (1.210 TF)
      - Broadwell-EP, 2.1GHz 18core
    - 36 Cores per node
    - 256 GiB (153.6GB/sec)
  - Network specifications
    - InfiniBand EDR

- **Hybrid parallelization**
  - 1process-18 threads (1process per socket)

- **Coloring algorithm for test**
  - Greedy, AMC, CM-Greedy, CM-AMC

- **Iteration is stop if relative residual fills the requirement**
  \[ \left\| \frac{r^k}{r^0} \right\|_2 \leq 10^{-7} \]

- **Target problems**
  - Parabolic_FEM, Thermal2, FLAN1565, Original Poisson model
  - (Florida matrix collections)
Comparing the convergence and calculation time

Convergence and calculations are similar to sequential coloring.

Normalized result based on sequential coloring

Y-axis = sequential / parallel of computational time or convergence

1.32(largest)
Parallelization of multi-coloring algorithms

Proposing hierarchical parallelization for the multi-coloring algorithms
→ For supporting any algorithms.

1. The best algorithm depends on applications.
2. We need the robustness for the target problems.

Existing reordering algorithms

- Level-set ordering
  - Lexicographic
  - Breadth First Search
  - Cuthill-McKee (CM)
  - Reverse CM

- Independent set
  (Multi-color) ordering
  - Greedy
  - Algebraic Multi-coloring (AMC)
  - Block AMC

Need a versatile parallelization method which is not changing properties
Objective of multi-coloring

**Coloring all nodes without neighboring same color**

→ Parallelizing IC preconditioner based on coloring results

Parallelizing this process with hierarchical approach
Hierarchical parallelization (two-level) 1/9

Process of a two-level hierarchical parallelization

Initial conditions
Process of the two-level hierarchical parallelization

Initial conditions
1. Each process separates calculation area.
Hierarchical parallelization (two-level) 3/9

Process of the two-level hierarchical parallelization

Initial conditions
1. Each process separates calculation area.
2. Gather the graph structures.
Hierarchical parallelization (two-level) 4/9

Process of the two-level hierarchical parallelization

Initial conditions
1. Each process separates calculation area.
2. Gather the graph structures.
3. Master process colors the nodes with any method.
Hierarchical parallelization (two-level) 5/9

Process of the two-level hierarchical parallelization

Initial conditions
1. Each process separates calculation area.
2. Gather the graph structures.
3. Master process colors the nodes with any method.
4. Broadcast coloring result.
Hierarchical parallelization (two-level) 6/9

Process of the two-level hierarchical parallelization

Initial conditions
1. Each process separates calculation area.
2. Gather the graph structures.
3. Master process colors the nodes with any method.
4. Broadcast coloring result.
5. Each process colors all nodes with any algorithm in parallel.
Hierarchical parallelization (two-level) 7/9

Process of the two-level hierarchical parallelization

Initial conditions
1. Each process separates calculation area.
2. Gather the graph structures.
3. Master process colors the nodes with any method.
4. Broadcast coloring result.
5. Each process colors all nodes with any algorithm in parallel.
Initial conditions
1. Each process separates calculation area.
2. Gather the graph structures.
3. Master process colors the nodes with any method.
4. Broadcast coloring result.
5. Each process colors all nodes with any algorithm in parallel.
Process of the two-level hierarchical parallelization

Initial conditions
1. Each process separates calculation area.
2. Gather the graph structures.
3. Master process colors the nodes with any method.
4. Broadcast coloring result.
5. Each process colors all nodes with any algorithm in parallel.

Finish
ILU preconditioner for the target problems

Needed a modification of the ILU preconditioner for more robustness

- IC preconditioning matrix is constructed through the incomplete version of a LU factorization.
- On ill-conditioned problem, ILU factorization increases numerical errors.

If the diagonal entries are much smaller than off-diagonal entries......

- Factorized part
- Factorizing part
- Unfactorized part

① In updating elements of a same row, off-diagonal(large) values divided by a diagonal(small) value have large numerical errors.

② Numerical errors are scattered in the process of updating unfactorized part.

→ On the worst case, a factorization fault occurs
Applying 2 regularization methods for ILU preconditioner → For robustness and improving convergence

- **Blocking technique** (Regularization ①)
  - Applying the incomplete decomposition to a block matrix
    1. More robustness because of including non-small off-diagonals
    2. Better convergence ratio because of allowing more fill-ins

1. o: Small entry  ●: Non-small entry

2. Coefficient matrix $A$
   - ICB factorization
   - ICB preconditioning matrix (2 x 2 block)
Applying 2 regularization methods for ILU preconditioner
→ For robustness and improving convergence

■ Blocking technique (Regularization①)
  • Applying the incomplete decomposition to a block matrix
    1. More robustness because of including non-small off-diagonals
    2. Better convergence ratio because of allowing more fill-ins

■ Diagonal transformation (Regularization②)
  • Adding constant value $\alpha$ to the diagonal elements
  • Directly method to make the diagonally dominant matrix

\[ \widetilde{A}_Z = A_Z + \alpha I \quad I = \text{identity matrix} \]
Target problems

Total: 128 data sets

- **Graph**
  - 4 data sets: 1k, 10k, 100k, 1M
    - | DoF | 1k    | 10k   | 100k  | 1M    |
    - |     | 1,000 | 10,000| 100,000| 1,000,000 |
    - | # non-zero | 13,000 | 130,000 | 1,300,000 | 13,000,000 |
  - 2 shift data sets: shift-1, shift-2
  - 8 shift values in each data

- **Topi**
  - 4 data sets: 1k, 10k, 100k, 1M
    - | DoF | 1k    | 10k   | 100k  | 1M    |
    - |     | 1,000 | 10,240| 102,400| 1,024,000 |
    - | # non-zero | 12,200 | 122,880| 1,228,800| 12,492,800 |
  - 2 shift data sets: shift-1, shift-2
  - 8 shift values in each data

Shift-2 of each problems make difficult conditions, comparatively
Multi-color ordering for parallelizing BIC preconditioner

Parallelization method of BIC preconditioner with multi-coloring

Example of 3-colored matrix

Forward and backward substitution has sequentiality.

By applying the Multi-color ordering,
Multi-color ordering for parallelizing BIC preconditioner

Parallelization method of BIC preconditioner with multi-coloring

Example of 3-colored matrix

Forward and backward substitution has sequentiality.

By applying the Multi-color ordering,
- No relationship between same color
- Calculating the elements with same color, parallely

The convergence of the multi-colored ICCG is changed.
In the program, we calculate the upper triangular matrix directly from the colored base matrix.
Applying algebraic multi-coloring (AMC) method

Sample code of AMC

```fortran
ncolor=some value ! Set number of used colors
color(1:n)=0 ! Initialize the array color
icolor=1
doi=1,n
  j=1
  do while(j <= lnz(i))
    if (color(lnzc(i, j))==icolor) then
      icolor=mod(icolor, ncolor)+1 !To next color
      j=0
    endif
    j=j+1
  enddo
  color(i)=icolor ! Assignment of color
  icolor=mod(icolor, ncolor)+1 !To next color
enddo
```

- We can control the number of colors.
  → The convergence ratio and computation time have dependency on the number of colors.

- The number of unknowns are nearly even in each colors, relatively.
  → Load is evenly distributed.

Compared the convergence between the sequential method and the proposed (32 processes) method. → Showed similar properties.
Unshowed large difference in the calculation times.

Graphs shows the computational time of iterations (32 processes).
Hierarchical parallelization (Multi-level)

We applied the multi-level algorithm for massively parallelism.

→ To reduce the time of the gathering, the scattering and the coloring on the master process

Gathering new graph

Coloring graph parallelly on each level

Scattering coloring results
Implementation of hierarchical approach
For robustness (Previous study)

IC preconditioned CG with regularization methods solve the targets

- Applied regularization methods
  - Blocking technique
  - Diagonal shifting

A simplified code of BIC-CG

Block IC decomposition

\[
\text{do } \text{itr} = 1, \text{DoF}
\]

\[
\text{dot_product}
\]

\[
q = \tilde{A}^{-1}r \text{(preconditioning)}
\]

Matrix Vector Multiplication

\[
\text{enddo}
\]
Proposing hierarchical parallelization for multi-color algorithms

→ Needed for solving the large scale SLEs on massively parallel systems

Block IC decomposition

do itr = 1, DoF
    dot_product
    \( q = \tilde{A}^{-1}r \) (preconditioning)
    Matrix Vector Multiplication
endo
For massively parallelization (Objective)

Proposing hierarchical parallelization for multi-coloring algorithms
→ Needed for solving the large scale SLEs on massively parallel systems

- Multi-color ordering is used for parallelize BIC-CG method.
  - Multi-coloring algorithms are not parallelized.

<table>
<thead>
<tr>
<th>Multi-coloring</th>
<th>Still sequentially</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block IC decomposition</td>
<td>Parallelizing with multi-coloring</td>
</tr>
<tr>
<td>do itr = 1, DoF</td>
<td>Parallelizing easily</td>
</tr>
<tr>
<td>dot_product</td>
<td>Matrix Vector Multiplication</td>
</tr>
<tr>
<td>$q = \tilde{A}^{-1}r$ (preconditioning)</td>
<td></td>
</tr>
<tr>
<td>enddo</td>
<td></td>
</tr>
</tbody>
</table>
Index

- Introduction
- Parallelization of BIC-CG
- Hierarchical parallelization of multi-coloring algorithms
- Numerical experiments
- Conclusion
Parallelizing the BIC preconditioner with multi-coloring

Example of IC decomposed matrix (Only upper)

Sparse nonzero entries

Forward and backward substitution has sequentiality.
Effect of multi-coloring on convergence

The result of multi-coloring influences the convergence and performance.

→ Multi-coloring changes the order of the matrices.
There are many multi-coloring algorithms.

\[ A_z \rightarrow C(A_z) \]

Zero and Nonzero pattern is same completely.

\[ IC(A_z) = C(A_z) \]

Same matrix

\[ A_z \rightarrow C(A_z) \]

Decomposed matrix is dense

\[ IC(A_z) \approx C(A_z) \]
Index

- Introduction
- Parallelization of BIC-CG
- Hierarchical parallelization of multi-coloring algorithms
- Numerical experiments
- Conclusion
Result of computational time on larger problems.

Showed the similar convergence and performance on larger problems.

Poisson 512*512*1024
Conclusion

- Proposing the hierarchical parallelization for multi-coloring algorithms
  - Versatile parallelization method

- Numerical experiments showed the results as we had expected.
  - There are no large differences between the sequentially and the parallelized coloring.
    → Properties of the existing coloring algorithms are not changed.
  - We have solved the large scale problem.

Future works
- Needed more performance for massively parallel system
  - Communication hide by coloring
- Attacking to more large problems derived from quantum systems
The performance of coloring routine is good.

The time of coloring is less than 2.5% in 128 nodes.
Evaluations of the numerical analysis 2/2

- Oakleaf-FX 4 ~ 1024 nodes
  
  Node specifications
  - SPARC64™ IXfx 16cores
  - 32GiB

  Network specifications
  - Tofu

- Hybrid parallelization
  1process-16 threads (1process per node)

- Coloring algorithm for test
  Hierarchical parallelized AMC(10)

- Target problem
  Graphen8194 × 4096 ≈ 33M DoF

Source: The electronic properties of graphene
Performance on graphene problem

Showed good performance both iteration and coloring part

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th># of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>27,812</td>
</tr>
<tr>
<td>8</td>
<td>27,823</td>
</tr>
<tr>
<td>16</td>
<td>27,812</td>
</tr>
<tr>
<td>32</td>
<td>27,823</td>
</tr>
<tr>
<td>64</td>
<td>27,801</td>
</tr>
<tr>
<td>128</td>
<td>27,823</td>
</tr>
<tr>
<td>256</td>
<td>27,823</td>
</tr>
<tr>
<td>512</td>
<td>27,801</td>
</tr>
<tr>
<td>1024</td>
<td>27,823</td>
</tr>
</tbody>
</table>
Needed robust IC preconditioner to solve target SLEs
→Proposing to apply regularizations

**Properties of the matrix** $A_Z$

- Complex symmetric
- Sparse
- Large DoF
- Small diagonal entries compared with off-diagonal
- Positive and negative diagonal entries
- ill-conditioned: high condition number
Needed robust IC preconditioner to solve target SLEs
→ Proposing to apply regularizations

Properties of the matrix $A_z$
- Complex symmetric
- Sparse
- Large DoF
- Small diagonal entries compared with off-diagonals
- Positive and negative diagonal entries
- Ill-conditioned: high condition number

These properties increase computational errors in the IC factorization.
→ ① By applying regularizations, we try to approximating $A_z$ by $\widetilde{A}_z$ which is a diagonal dominant matrix.
② Factorizing $\widetilde{A}_z$
Applying 2 regularization methods for IC preconditioner → For robustness and improving convergence

- Blocking technique (Regularization①)
  - Applying the incomplete decomposition to a block matrix
    1. More robustness because of including non-small off-diagonals
    2. Better convergence ratio because of allowing more fill-ins
Applying 2 regularization methods for IC preconditioner  
→ For robustness and improving convergence

- **Blocking technique (Regularization ①)**
  - Applying the incomplete decomposition to a block matrix
    1. More robustness because of including non-small off-diagonals
    2. Better convergence ratio because of allowing more fill-ins

<table>
<thead>
<tr>
<th>Coefficient matrix $A$</th>
<th>Block IC factorized matrix (block size is 2)</th>
</tr>
</thead>
</table>

**BCRS format is used.**
Applying 2 regularization methods for the IC preconditioner → For robustness and improving convergence

- **Blocking technique (Regularization①)**
  - Applying the incomplete decomposition to a block matrix
    1. More robustness because of including non-small off-diagonals
    2. Better convergence ratio because of allowing more fill-ins

- **Diagonal transformation (Regularization②)**
  - Adding constant value $\alpha$ to the diagonal elements
  - Directly method to make the diagonally dominant matrix

\[
\begin{align*}
\widetilde{A}_z &= A_z + \alpha I \\
I &= \text{identity matrix}
\end{align*}
\]

* $\widetilde{A}_z$ is a matrix for IC factorization.
  Regularization② is only applied for the matrix.
Analysis conditions (previous report)

- Iteration method: COCG
  - The algorithm is similar to CG method
  - For the complex symmetric coefficient matrix

- Preconditioner: IC decomposition with
  - Blocking technique
  - Diagonal shifting

\[ b = randam(min = 1, max = 10) \]
*both real and imaginary parts

- Iteration is stop if the number of iteration reach to DoF
  or relative residual fills the requirement \[ \frac{r^k}{r^0} \leq 10^{-7} \]
Target problems

Total: 128 data sets

- **Graphene (Real values)**
  - 4 data sets: 1k, 10k, 100k, 1M
    
    |          | 1k      | 10k     | 100k    | 1M      |
    |----------|---------|---------|---------|---------|
    | DoF      | 1,000   | 10,000  | 100,000 | 1,000,000 |
    | # non-zero| 13,000  | 130,000 | 1,300,000 | 13,000,000 |

  - 16 zero data sets

- **Topological insulator (Complex values)**
  - 4 data sets: 1k, 10k, 100k, 1M
    
    |          | 1k      | 10k     | 100k    | 1M      |
    |----------|---------|---------|---------|---------|
    | DoF      | 1,000   | 10,240  | 102,400 | 1,024,000 |
    | # non-zero| 12,200  | 122,880 | 1,228,800 | 12,492,800 |

  - 16 zero data sets
Result

Solved all data sets

(0.0, 1.0)-BICCG(64) : Diagonal shifting is (0.0, 1.0) Block size is 64
Example of eigenvalue

Freq: 147.0 Hz

Ref: Nastran HP
Block ICCG preconditioner for CG method

The ICCG preconditioner with a blocking technique

\[ U_{i,j}^b = (D_i^b)^{-1} \left( B_{i,j}^b - \sum_{k=1}^{i-1} (U_{k,i}^b)^t D_k^b U_{k,j}^b \right) \]

- More convergence than the general ICCG
  \( \rightarrow \) Permitting more fill-ins

- More robustness
  \( \rightarrow \) Diagonal matrix including off-diagonal elements
Hierarchical parallelization (Multi-level)

We applied the multi-level algorithm
Parallelization of multi-coloring

Proposing hierarchical parallelization for the multi-coloring algorithms

General coloring algorithms are sequential.

Two approaches for parallelizing the multi-coloring algorithms

■ Proposing special parallelized multi-coloring algorithms.
  ● Better for performance of coloring
  ● Sometimes, not good for convergence and performance of BIC-CG

■ Proposing versatile parallelization method
  → Hierarchical approach
  ● Parallelize any coloring algorithms
    ✓ Can we parallelize coloring algorithms without changing properties?
      ← Investigating with numerical experiments