A new Matrix Reordering Method for GPU Acceleration of an ILU Preconditioner

Kengo Suzuki[1], Takeshi Fukaya[1], and Takeshi Iwashita[2]
[1]Hokkaido University, [2]Kyoto University

Introduction

Solve linear systems $Ax = b$ efficiently on a GPU

Highlights of this study:

- Improve the FGMRES solver preconditioned by ILU(0) on a GPU.
- Enhance the concurrency of ILU(0) by **matrix reordering**.
- Utilize our previous reordering method, HBMC [1], which can increase the concurrency of ILU(0) while maintaining the convergence rate.

Sparse triangular systems to be solved in ILU(0) have data dependencies, making it difficult to use GPUs effectively.

Matrix reordering

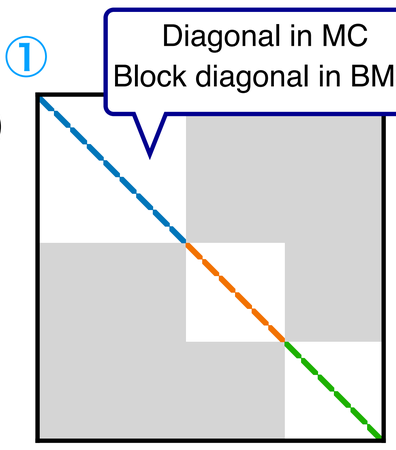
Enhances the concurrency of ILU(0)

However, it includes a trade-off problem between **concurrency and convergence**.

For example,

- **MC ordering** (a strong candidate on GPUs)
  - Increases the concurrency significantly.
  - Reduces the effect of ILU(0) on convergence.
- **BMC ordering** (a typical method on CPUs)
  - Maintains the effect of ILU(0) on convergence.
  - Lacks fine-grained concurrency due to the block diagonal structures.

① Diagonal in MC / Block diagonal in BMC



Example of MC and BMC. Nonzeros only exist in colored area.

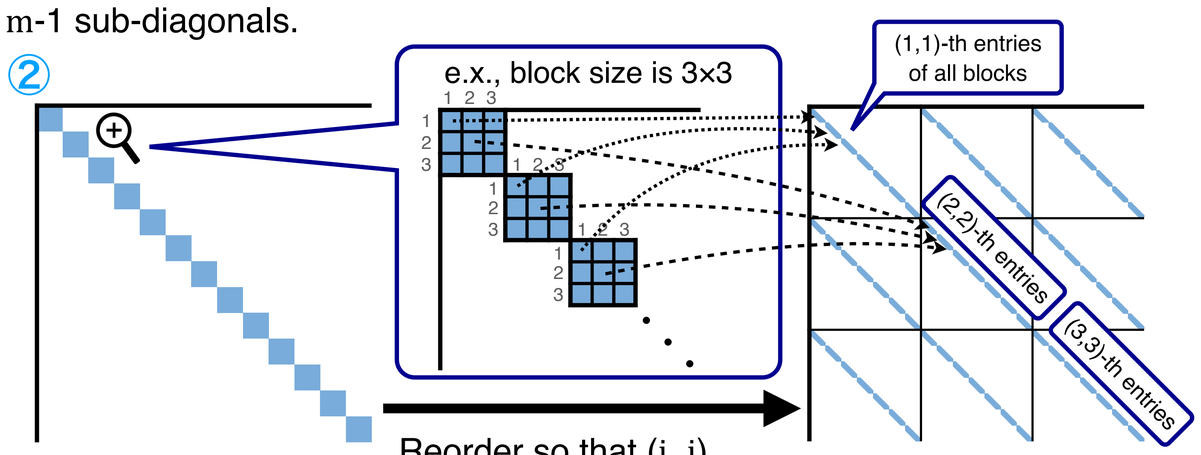
ILU(0) has the same sparse pattern as A; transforming A reorders ILU(0).

HBMC ordering

Enables the vectorization of BMC-ILU(0)

HBMC expands the blocks (of size $m \times m$) of BMC into a main diagonal and m-1 sub-diagonals.

② e.x., block size is 3×3



Upper left part of the example (BMC) in Fig. 1.

Reorder so that (i, i) entries of all blocks are alined consecutively.

Structure generated by HBMC.

The center and lower right parts are reordered in the same way.
BMC- and HBMC-ILU(0) are mathematically the same in convergence.

Two possible GPU Implementations of HBMC-ILU(0)

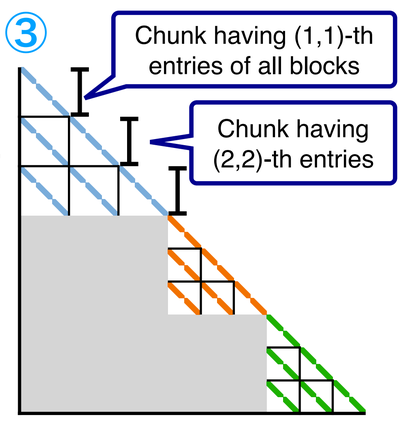
- **Imp-1** (does not use the sub-diagonal structure)

  Considers HBMC as just a (#colors×m)-color MC.
  - Can be programed easily with simple data flow.
  - Requires **(#colors×m)-1** synchronization.
- **Imp-2** (Utilizes the sub-diagonal structure)

  Removes the redundant synchronization.
  - Requires only **#colors-1** synchronization.
  - Needs a more complex program; Per color, the thread responsible for a certain row of the chunk having (1,1)-th entries computes sequentially the same rows in the chunks including (i, i)-th ($i = 2,\ldots,m$) entries.

  Imp-2 is not described in the extended abstract.

③



Example of the lower factor of HBMC-ILU(0)

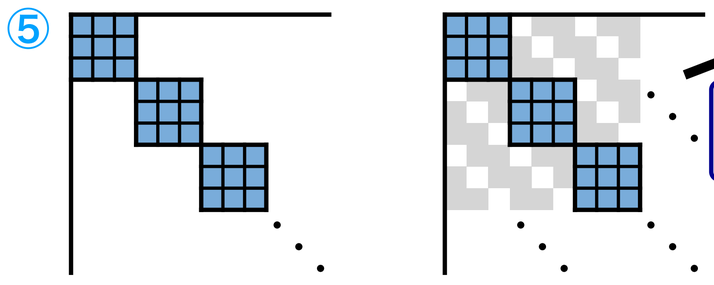
This can be regarded as a 9-color MC-ILU(0) if the sub-diagonals are ignored.

MMC ordering (proposed method)

A variation of Imp-1 of HBMC-ILU(0)

Since Imp-1 accepts a structure shown in Fig. 4, coloring can be performed more aggressively.

⑤

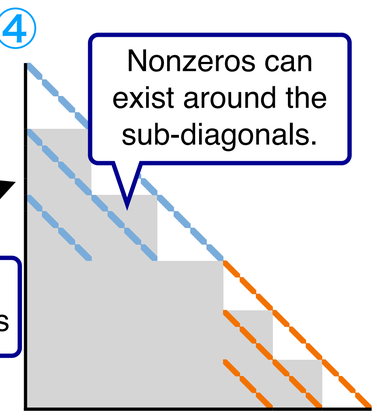


Only unconnected blocks are labeled with the same color in BMC and HBMC.

The same color blocks can have connections that are in off-diagonal.

Expand the blocks

④



Example of the lower factor of MMC-ILU(0). MMC requires **fewer #colors** than HBMC.

The aggressive coloring may degrade the convergence of HBMC-ILU(0).

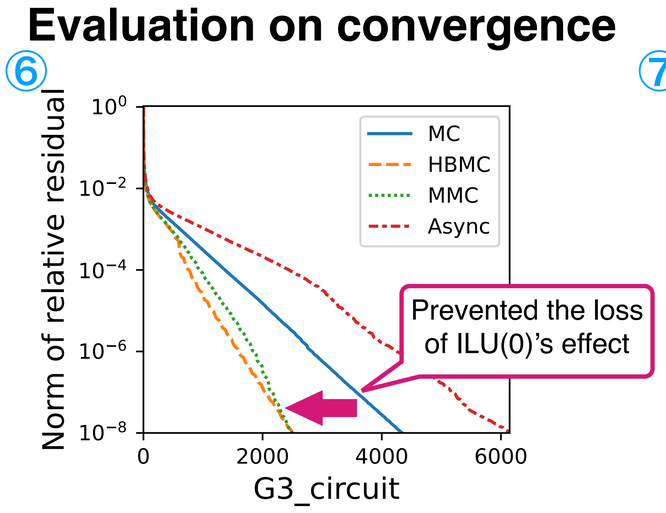
Numerical results

- Selected the 9 test problems from the SuiteSparse Matrix Collection.
- Solved the problems on an NVIDIA V100 controlled by an Intel Xeon.
- Compared the five preconditioners below using the FGMRES(50) solver.

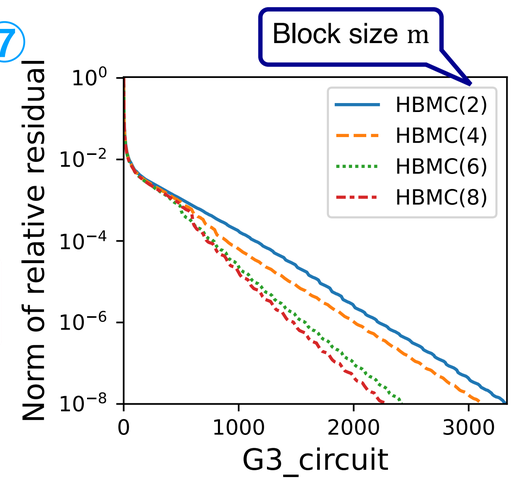
Name	Description
MC (baseline)	ILU(0) with MC ordering
HBMC-1	ILU(0) with HBMC in Imp-1
HBMC-2	ILU(0) with HBMC in Imp-2
MMC	ILU(0) with MMC, a variation of HBMC-1
Async	ILU(0) with an asynchronous block Jacobi solver

Evaluation on convergence

⑥



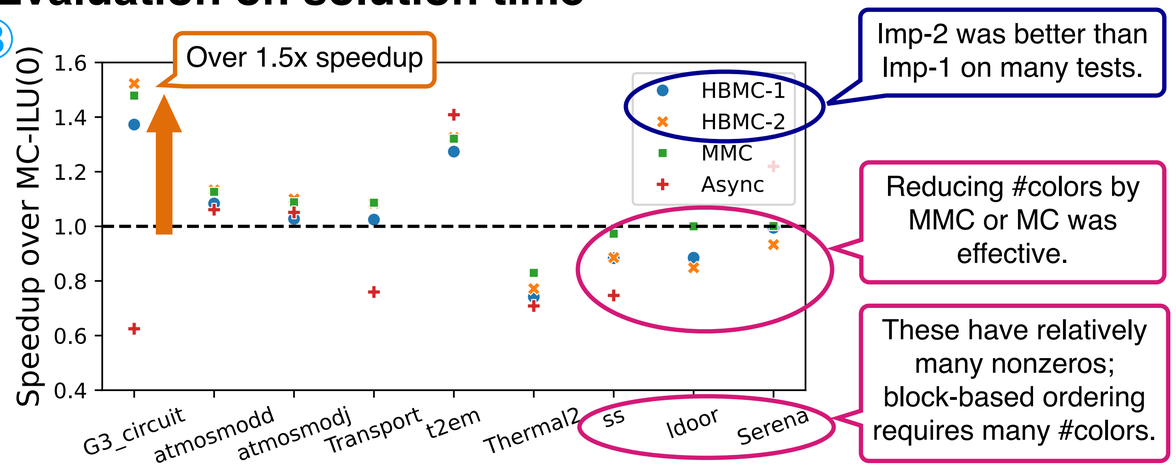
⑦



- As the block size increased, HBMC-ILU(0) achieved better convergence, although execution time per iteration increased.
- The reordering-based ILU(0) performed well even on problems where the Iterative solver-based ILU(0) (Async) required many iterations.

Evaluation on solution time

⑧



Conclusions

- **Block-based matrix reordering** is effective in accelerating ILU(0) **even on a GPU** and can outperform the iterative solver-based approach.
- **HBMC (in imp-2)** is effective for problems with **few nonzeros**, while **MC and MMC** are useful for problems with **many nonzeros**.

[1] T. Iwashita, et. al. *CCF Transaction on High Performance Computing* 2 (2020), 84–97.

HPC Asia 2024, January 25—27 2024, Nagoya, Japan