Batched solvers and Preconditioners in Ginkgo

Pratik Nayak, Isha Aggarwal, Aditya Kashi and Hartwig Anzt

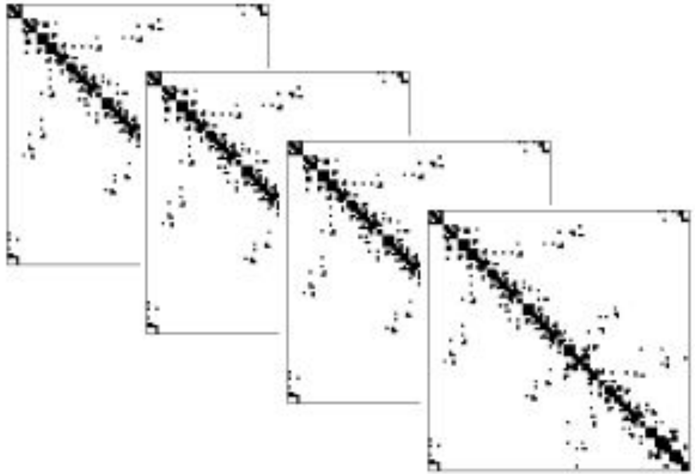
SIAM Conference on Computational Science and Engineering, Amsterdam 2023

Outline

- Motivation
- Design philosophy and choices
- Implementation
- Applications and performance analysis
- Adding in the preconditioners

What are batched methods ?

- Batching: Independent computations that can be scheduled in parallel.
- Are highly suitable for GPUs and processors with many parallel computing units.
- Can maximize utilization of the GPU, due to excellent scalability.



What are batched methods ?

- Related work:
  - Usage in block-Jacobi preconditioners (Anzt. et.al PMAM 17)
  - Dense triangular solves on GPUs, DGETRF (Dong et.al 2014)
  - Tri-/Penta- diagonal solvers on GPUs (Carroll et.al 2021, Gloster et.al 2019, Valero-Lara et.al 2018)
  - Batched BLAS interface (Dongarra et.al 2016)

Why batched iterative methods ?

- Most current research and software focuses on dense and direct solvers.
- For medium sized problems, dense and/or direct methods run into memory issues.
- Very high accuracy sometimes not required. Iterative methods provide tunable accuracy.
- The applications have matrices with relatively low condition numbers.

Opportunities

- Shared sparsity pattern can allow for optimized storage and caching matrices in constant memory.
- Linear system solution inside a non-linear loop can make use of better initial guesses from previous iterations.
- Independent convergence and stopping for each individual linear system.

Ginkgo's batched interface: Objectives

- Store one copy of the sparsity pattern and store the different values.
- Provide different Sparse matrix formats to support different sparsity patterns.
- Provide a wide variety of solvers for both symmetric and non-symmetric problems.
- Maximize cache usage and fuse kernels to reduce kernel launch latency.

https://github.com/ginkgo-project/ginkgo/tree/batch-develop

Ginkgo's batched interface: Design

Design philosophy:

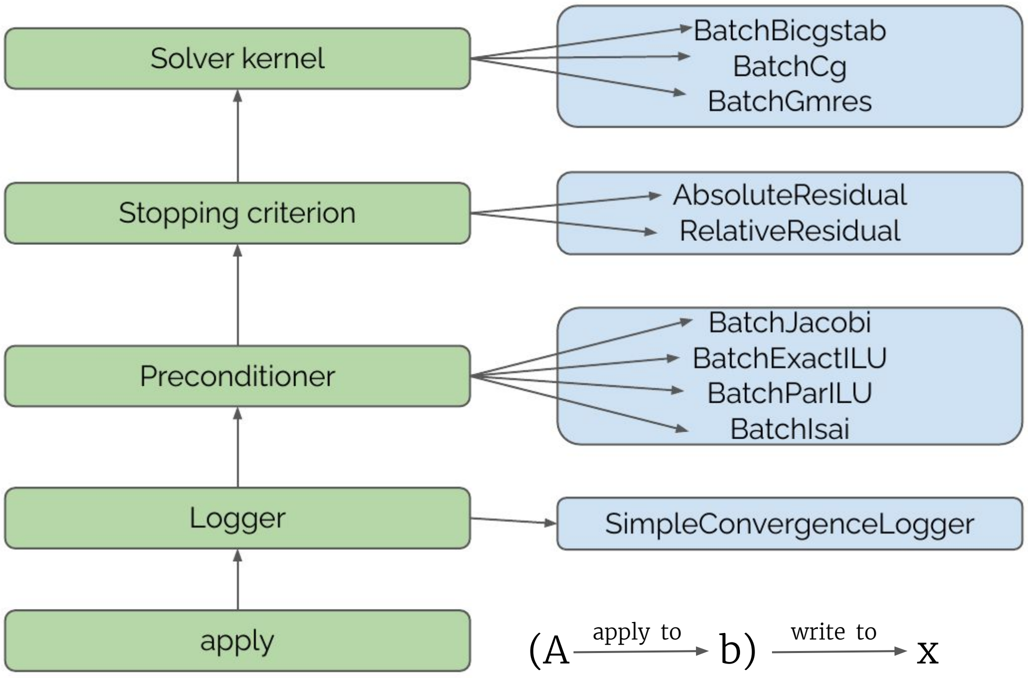
- Template the global solver apply kernel on logger, stopping criterion, matrix format and preconditioner type.
- Auto-configure shared memory based on problem size.
- Solve one linear system on one thread block.

Functionality:

- Sparse matrix formats: BatchCsr and BatchEll
- Iterative solvers: BatchBicgstab, BatchGmres, BatchCg, BatchIdr and BatchRichardson
- Preconditioners: BatchBlockJacobi, BatchILU, BatchISAI, BatchParILU

Multi-level dispatch mechanism

- Single device kernel call, but selection of different parameters through a multi-level dispatch.
- Allows for optimal use of caches and compute resources without launch overheads.



Automatic shared memory config

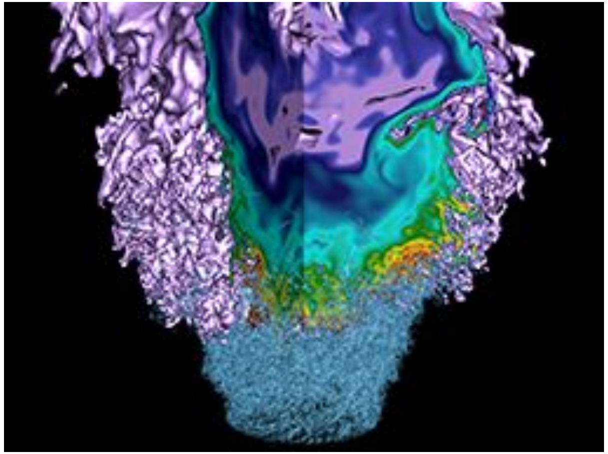
- Red objects: Intermediate vectors in SpMV: High priority
- Blue objects: Other vectors: Low priority
- Green objects: Constant matrices or vectors (In constant cache)

$\boldsymbol{r} \leftarrow \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}, \hat{\boldsymbol{r}} \leftarrow \boldsymbol{r}, \boldsymbol{p} \leftarrow \boldsymbol{0}, \boldsymbol{v} \leftarrow \boldsymbol{0}$
$\rho' \leftarrow 1, \omega \leftarrow 1, \alpha \leftarrow 1$
for $i < N_{iter}$ **do**
  if $\|\boldsymbol{r}\| < \tau$ **then**
    Break
  end if
  $\rho \leftarrow \boldsymbol{r} \cdot \boldsymbol{r}'$
  $\beta \leftarrow \frac{\rho' \alpha}{\rho \omega}$
  $\boldsymbol{p} \leftarrow \boldsymbol{r} + \beta(\boldsymbol{p} - \omega \boldsymbol{v})$
  $\hat{\boldsymbol{p}} \leftarrow \text{PRECOND}(\boldsymbol{p})$
  $\boldsymbol{v} \leftarrow \boldsymbol{A}\hat{\boldsymbol{p}}$
  $\alpha \leftarrow \frac{\rho}{\hat{\boldsymbol{r}} \cdot \boldsymbol{v}}$
  $\boldsymbol{s} \leftarrow \boldsymbol{r} - \alpha \boldsymbol{v}$
  if $\|\boldsymbol{s}\| < \tau$ **then**
    $\boldsymbol{x} \leftarrow \boldsymbol{x} + \alpha \hat{\boldsymbol{p}}$
    Break
  end if
  $\hat{\boldsymbol{s}} \leftarrow \text{PRECOND}(\boldsymbol{s})$
  $\boldsymbol{t} \leftarrow \boldsymbol{A}\hat{\boldsymbol{s}}$
  $\omega \leftarrow \frac{\boldsymbol{t} \cdot \boldsymbol{s}}{\boldsymbol{t} \cdot \boldsymbol{t}}$
  $\boldsymbol{x} \leftarrow \boldsymbol{x} + \alpha \hat{\boldsymbol{p}} + \omega \hat{\boldsymbol{s}}$
  $\boldsymbol{r} \leftarrow \boldsymbol{s} - \omega \boldsymbol{t}$
  $\rho' \leftarrow \rho$
end for

Application: Combustion simulations

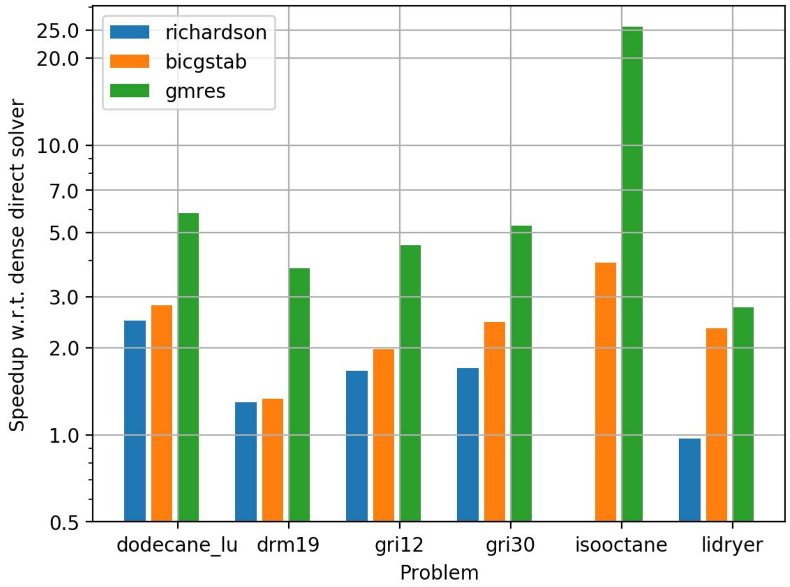
- PeleLM is a parallel, adaptive mesh refinement (AMR) code that solves the Navier-Stokes equations with in the low Mach number regime with the chemical reaction mechanisms.
- Interfaced through the SUNDIALS library.
- https://amrex-combustion.github.io/PeleLM/overview.html

Problem	Size	Non-zeros (A)	Non-zeros (L+U)
dodecane_lu	54	2,332 (80%)	2,754 (94%)
drm19	22	438 (90%)	442 (91%)
gri12	33	978 (90%)	1,018 (93%)
gri30	54	2,560 (88%)	2,860 (98%)
isooctane	144	6,135 (30%)	20,307 (98%)
lidryer	10	91 (91%)	91 (91%)



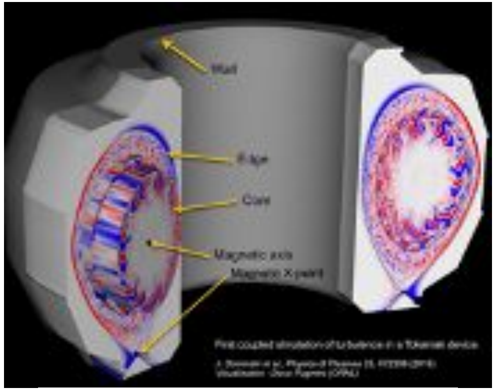
Application: Combustion simulations

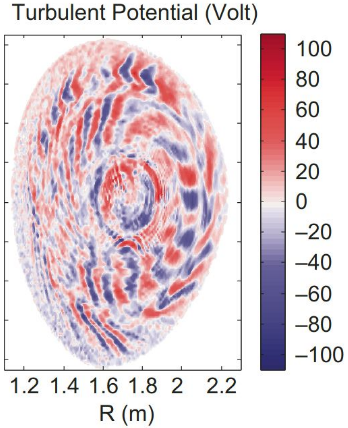
- cuBLAS Batched dense direct solver can be out-performed even for small problems.
- GMRES can perform really well with very low iteration counts.
- We also support scaling ($S_1 A\, S_2$), to allow for better conditioned matrices.



Application: Fusion simulation

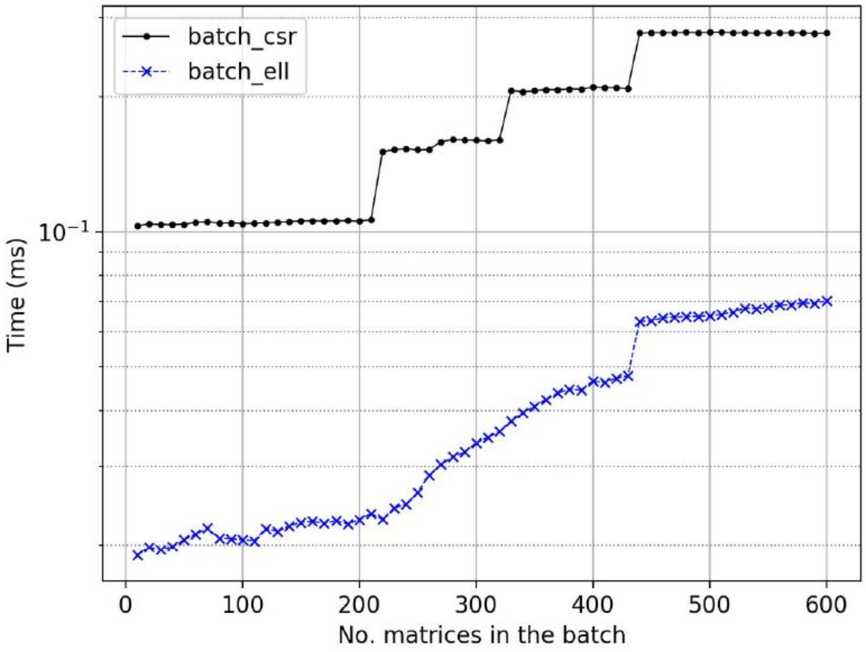
- XGC is a X-point Gyrokinetic particle in cell code.
- Aims to simulate the edge region of the magnetically confined thermonuclear fusion plasma.
- A module of the WDMApp (Whole Device Modelling Application) of ECP.
- More details and results in Part II: [*MS341*]





Performance results: XGC - SpMV

- On A100 with increasing number of matrices in the batch.
- Ell matrix format allows for coalesced access with one thread per row.
- Jumps seen at #compute cores limit boundaries

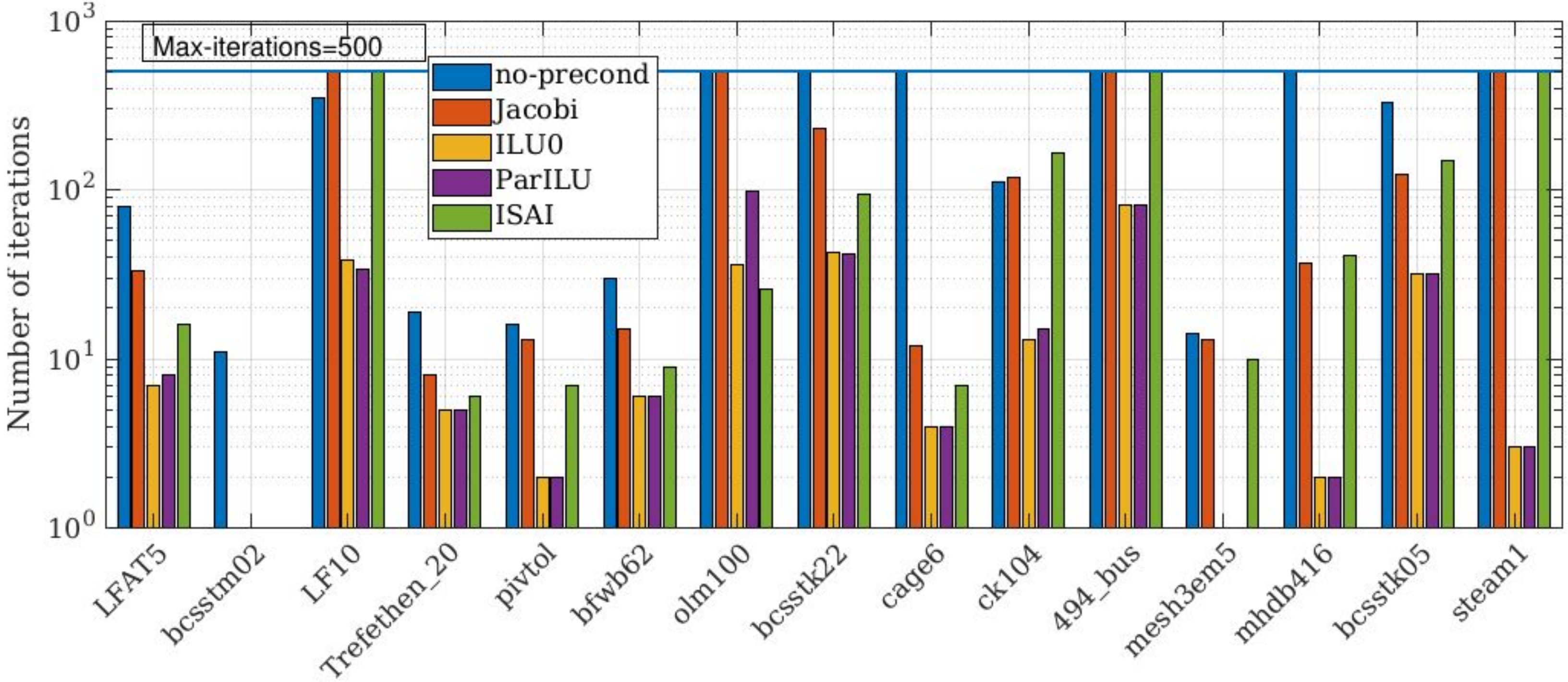


Batched Preconditioners

- 3 test cases.
  - Scaling with 3 point stencil
  - General matrices from Suitesparse.
  - Practical problems from PeleLM
  - Ordered in terms of increasing nnz count.
- Iteration counts shown for different preconditioners

	size	nonzeros	*No Precond*	*Jacobi*	*ILU(0)*	*ParILU*	*ISAI*
1D Laplace							
3pt-stencil-64	64	190	16	11	1	1	6
Suitesparse							
LFAT5	14	46	80	33	7	8	16
bcsstm02	66	66	11	1	1	1	1
LF10	18	82	351	–	38	34	–
Trefethen_20	20	158	19	8	5	5	6
pivtol	102	306	16	13	2	2	7
bfwb62	62	342	30	15	6	6	9
olm100	100	396	–	–	36	98	26
bcsstk22	138	696	493	229	43	42	95
cage6	93	785	–	12	4	4	7
ck104	104	992	112	118	13	15	164
494_bus	494	1666	–	–	81	81	–
mesh3em5	289	1889	14	13	1	1	10
mhdb416	416	2312	–	37	2	2	41
bcsstk05	153	2423	325	124	32	32	149
steam1	240	3762	–	–	3	3	–
PeleLM							
isooctane	144	6135	–	38	3	4	–

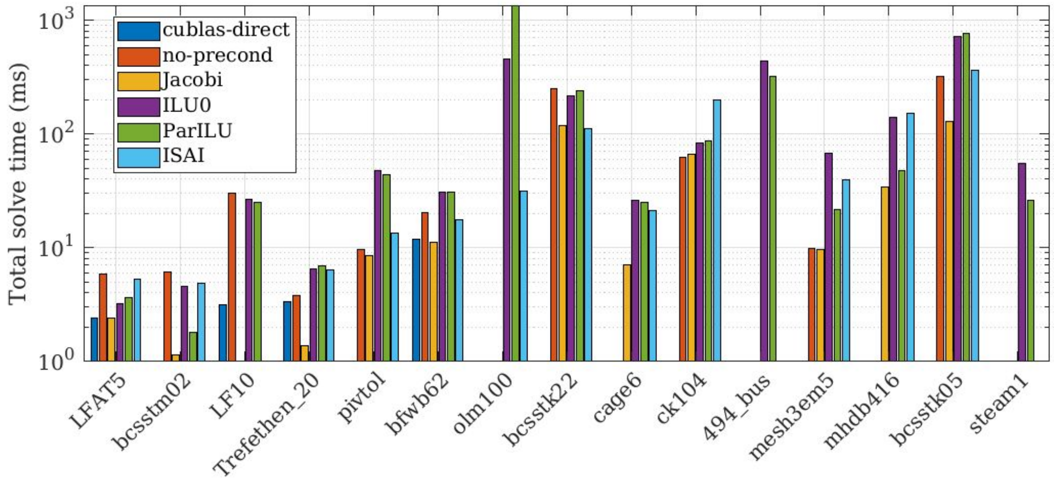
Iteration counts



Total solve time

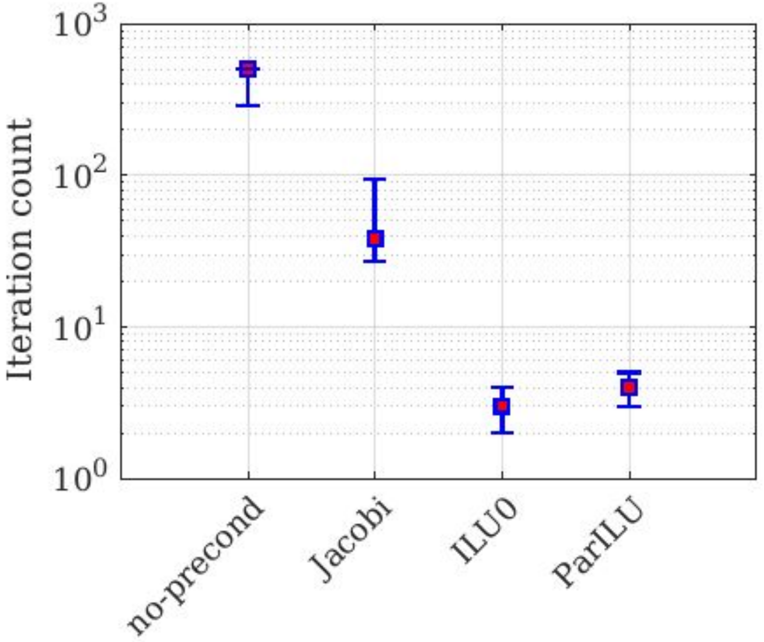
Karlsruhe Institute of Technology

- ILU(0) is the most robust and enables solution for all problems.
- ParILU can win in some cases due to cheaper generation.
- Scalar Jacobi can still be very effective despite large number of iterations.



Are preconditioners useful ? Isooctane problem

- Some variation in iteration counts in problems in a batch.
- Preconditioners can significantly reduce the iteration count.



Summary

- Batched iterative methods and preconditioners can be very beneficial for parallel solution of many small independent problems.
- Reduction of launch overhead and efficient utilization of cache hierarchy crucial.
- Lesser modularity in terms of composability, but significant performance gains possible.

Future work and perspectives

- Banded solvers for banded and multi-diagonal matrices.
  - Optimized storage and using adapted LAPACK dgbsv routines.
- Extensions to monolithic problems by maximizing the cache usage and aiming to cache the matrix in the L2/L3 cache.
  - Has shown promise for medium problems, but larger problems needs more care.

Karlsruhe Institute of Technology

Acknowledgements

KIT | NHR
National High-Performance Computing Center

ECP
EXASCALE COMPUTING PROJECT

HELMHOLTZ
RESEARCH FOR GRAND CHALLENGES

Thank you!

Ginkgo

https://github.com/ginkgo-project/ginkgo

nayak@kit.edu

Summary

- Batched iterative methods can be very beneficial for parallel solution of many small independent problems.
- Reduction of launch overhead and efficient utilization of cache hierarchy crucial.
- Lesser modularity in terms of composability, but significant performance gains possible.

https://github.com/ginkgo-project/ginkgo

nayak@kit.edu

Why not Block Diagonal assembly ?

- Need to wait for slowest problem.
- Eigenvalues of the monolithic problems union of the eigenvalues of the individual problems.
- Independent stopping is difficult and may result in divergence.

$$\mathbf{A} = \begin{bmatrix} A_1 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_K \end{bmatrix}.$$