An investigation of parallel performance of block epsilon-circulant preconditioner for time-dependent PDEs

Ryo Yoda† (ryoda@uni-wuppertal.de), Matthias Bolten†

†University of Wuppertal (Germany)

BERGISCHE UNIVERSITÄT WUPPERTAL

HPC Asia 2024

The International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2024)
Jan 25-27, 2024 | Nagoya, Japan

Introduction and Overview

- **Parallel-in-time approaches for time-dependent PDEs**
  - The classical sequential time-stepping method has parallelism limitations in massively parallel environments.
  - Parallel-in-time approaches extract time parallelism by solving all-at-once systems in parallel.
  - The multigrid-based parallel-in-time approach, which is one of the leading solutions for large-scale time-dependent PDEs, has achieved good performance for parabolic problems, but still has challenges for hyperbolic problems.
  - New approach using block circulant type preconditioning [1, 2]
    - ✓ It achieves good convergence even for hyperbolic problems. [1]
    - ✗ However, its parallel performance has not been fully investigated.
- **Novelty and Originality**
  - Investigate parallel performance of BEC preconditioning with pure MPI impl.
  - Evaluate three types of implementations for FFT parts
  - Compared with the sequential time-stepping method and multigrid reduction in time (MGRIT) [5], a popular multigrid-based parallel-in-time method.

Block epsilon-circulant (BEC) preconditioner

- **Assumption: Use the same time integrator for all time steps**
  - Time-stepping with the backward Euler method $\quad M\left(\frac{u_i - u_{i-1}}{\Delta t}\right) + Ku_i = f_i \in \mathbb{R}^{n_x} \quad (i = 1, \dots, n_t)$
  - All-at-once space-time system ($A_0 = (M + \Delta t K)$ and $A_1 = -M$)

$$A\mathbf{u} := \begin{bmatrix} A_0 & & & \\ A_1 & A_0 & & \\ & \ddots & \ddots & \\ & & A_1 & A_0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n_t} \end{bmatrix} = \begin{bmatrix} \Delta t f_1 - A_1 u_0 \\ \Delta t f_2 \\ \vdots \\ \Delta t f_{n_t} \end{bmatrix} =: \mathbf{f} \in \mathbb{R}^{n_t n_x}.$$

- **Main idea: Use the block circulant type preconditioner**
  - McDonald et al. [2] propose the block circulant (BC) preconditioning.
  - Lin et al. [1] propose the block epsilon-circulant (BEC) preconditioning.
  - BEC precond. achieves independent convergence for spatial mesh size with sufficiently small epsilon $(0 < \epsilon \lesssim \sqrt{\Delta t})$ that BC precond. does not achieve.

$$P_{\mathbf{BC}} = \begin{bmatrix} A_0 & & & A_1 \\ A_1 & A_0 & & \\ & \ddots & \ddots & \\ & & A_1 & A_0 \end{bmatrix} \in \mathbb{R}^{n_t n_x \times n_t n_x}, \quad P_{\mathbf{BEC}} = \begin{bmatrix} A_0 & & & \epsilon A_1 \\ A_1 & A_0 & & \\ & \ddots & \ddots & \\ & & A_1 & A_0 \end{bmatrix} \in \mathbb{R}^{n_t n_x \times n_t n_x}$$

Three-step procedure of BEC preconditioning

- **$R_{\mathbf{BEC}}$ for p-step time discretization matrix**

$$R_{\mathbf{BEC}} = \begin{bmatrix} r_0 & & \epsilon r_p & \cdots & \epsilon r_2 & \epsilon r_1 \\ r_1 & r_0 & & \ddots & \cdots & \epsilon r_2 \\ \vdots & \ddots & \ddots & & \ddots & \vdots \\ r_p & \cdots & \cdots & \ddots & & \epsilon r_p \\ & \ddots & \cdots & r_1 & r_0 & \\ & & r_p & \cdots & r_1 & r_0 \end{bmatrix} \in \mathbb{R}^{n_t \times n_t}$$

$$R_{\mathbf{BEC}} = D_\epsilon^{-1} \mathcal{F}_{n_t}^* \Lambda_\epsilon \mathcal{F}_{n_t} D_\epsilon$$
$$\mathcal{F}_{n_t} := n_t\textbf{-sized DFT matrix}$$
$$D_\epsilon := \mathbf{diag}\left(\epsilon^{\frac{0}{n_t}}, \epsilon^{\frac{1}{n_t}}, \dots, \epsilon^{\frac{n_t-1}{n_t}}\right)$$
$$\Lambda_\epsilon := \mathbf{diag}\left(\lambda_0^{(\epsilon)}, \lambda_1^{(\epsilon)}, \dots, \lambda_{n_t-1}^{(\epsilon)}\right)$$

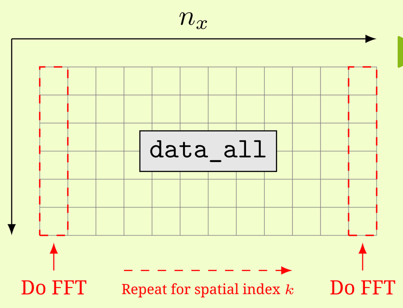
- **Kronecker form $P_{\mathbf{BEC}}$**

$$P_{\mathbf{BEC}} = R_{\mathbf{BEC}} \otimes M + \Delta t I_{n_t} \otimes K$$
$$= \left[(D_\epsilon^{-1}\mathcal{F}_{n_t}^*) \otimes I_{n_x}\right] \mathbf{blockdiag}(B_0, B_1, \dots, B_{n_t-1}) \left[(\mathcal{F}_{n_t} D_\epsilon) \otimes I_{n_x}\right],$$
where $\quad B_k = \lambda_k^{(\epsilon)} M + \Delta K \in \mathbb{C}^{n_x \times n_x}, \quad (k = 0, 1, \dots, n_t - 1)$

Algorithm 1 Three-step procedure of BEC preconditioning: $\mathbf{z} = P_{\mathbf{BEC}}^{-1}\mathbf{y}$

1: Compute $\tilde{\mathbf{y}} = \left[(\mathcal{F}_{n_t} D_\epsilon) \otimes I_{n_x}\right] \mathbf{y}$ ▷ FFT part
2: Solve $B_k \tilde{z}_k = \tilde{y}_k$ for $\tilde{z}_k \quad (k = 0, 1, \dots, n_t - 1)$ ▷ Complex solver part
3: Compute $\mathbf{z} = \left[(D_\epsilon^{-1}\mathcal{F}_{n_t}^*) \otimes I_{n_x}\right] \tilde{\mathbf{z}}$ ▷ FFT part

- **FFT parts**
  - For one-dimensional time-step-sized vectors



  - Three type FFT impls.
    1. FFTW [3] for 1D array
    2. FFTE for 1D array
    3. The redistributed FFTW for 2D array
    - ⇒ Assigns temporal parallelism to the spatial domain
    - ⇒ More stable and faster

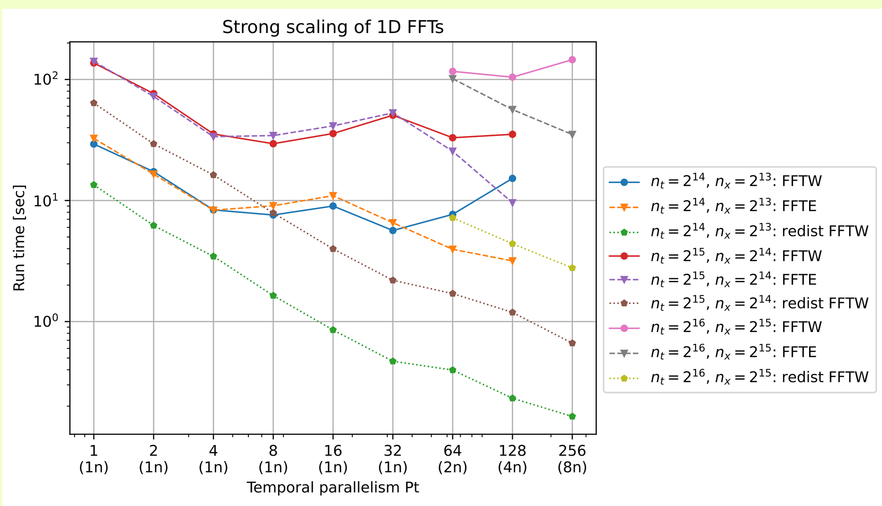


Figure 1: Strong scaling for three FFT implementations

- **Complex solver parts**
  - Linear solvers for space-sized complex-valued system
  - Equivalent real-valued K1-formulation for $B_k \tilde{z}_k = \tilde{y}_k$

$$B_k = B_k^{(R)} + iB_k^{(I)}$$
$$\tilde{z}_k = \tilde{z}_k^{(R)} + i\tilde{z}_k^{(I)}$$
$$\tilde{y}_k = \tilde{y}_k^{(R)} + i\tilde{y}_k^{(I)}$$

$$\begin{bmatrix} +B_k^{(R)} & -B_k^{(R)} \\ +B_k^{(I)} & +B_k^{(R)} \end{bmatrix} \begin{bmatrix} \tilde{z}_k^{(R)} \\ \tilde{z}_k^{(I)} \end{bmatrix} = \begin{bmatrix} \tilde{y}_k^{(R)} \\ \tilde{y}_k^{(I)} \end{bmatrix}$$

  - Linear solver setting using Trilinos packages [4]
    - AMG preconditioning for the diagonal block matrices in ML
    - GMRES solver in AztecOO

Numerical experiments

Measurement environments: Wisteria/BDEC-01 Odyssey system (Fujistu compiler and MPI v4.9.0, FFTW v.3.3.9, Trilinos v14.5)

- **Two-dimensional diffusion problems**

$$u_t(x,y,t) = 10^{-5}\Delta u(x,y,t)$$
$$u(x,y,t) = 0 \quad \textbf{on} \quad \partial\Omega$$
$$u(x,y,0) = x(x-1)y(y-1)$$

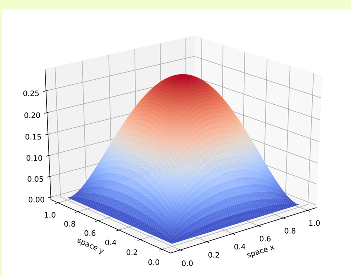


Figure 2: 3D Plot

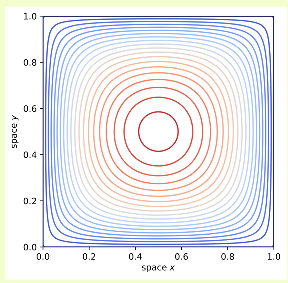


Figure 3: 2D Plot

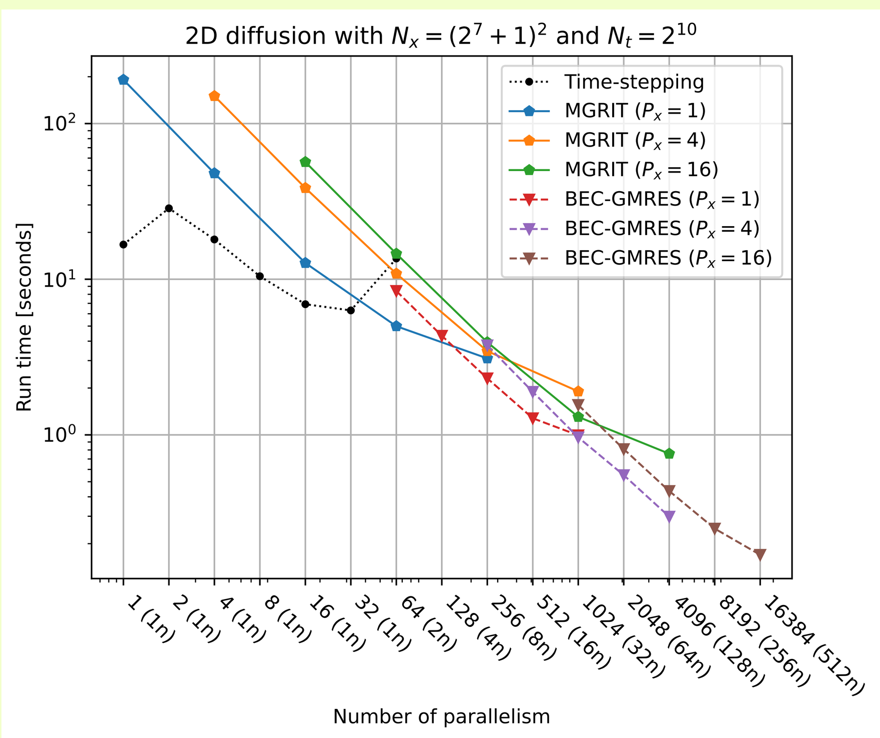


Figure 4: Strong scaling with $n_x = (2^7+1)^2$ and $n_t = 2^{10}$

- Time-stepping stagnates.
- MGRIT and BEC-GMRES achieve good scaling.
- BEC-GMRES outperforms MGRIT in high parallelism.

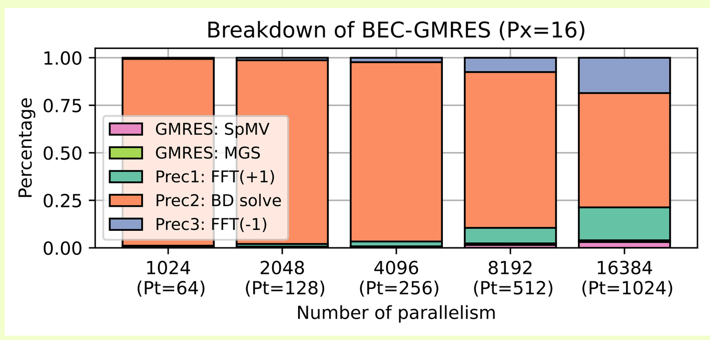


Figure 5: Breakdown of BEC-GMRES with $P_x = 16$

- **Two-dimensional convection-diffusion problems**

$$u_t(x,y,y) = \frac{1}{200}\Delta u - \overrightarrow{w} \cdot u$$
$$u(x,y,t) = (1 - \exp(-10t))\phi(x,y)$$
$$u(x,y,0) = 0$$

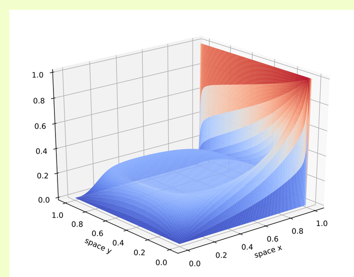


Figure 6: 3D Plot

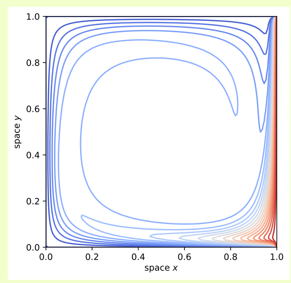


Figure 7: 2D Plot

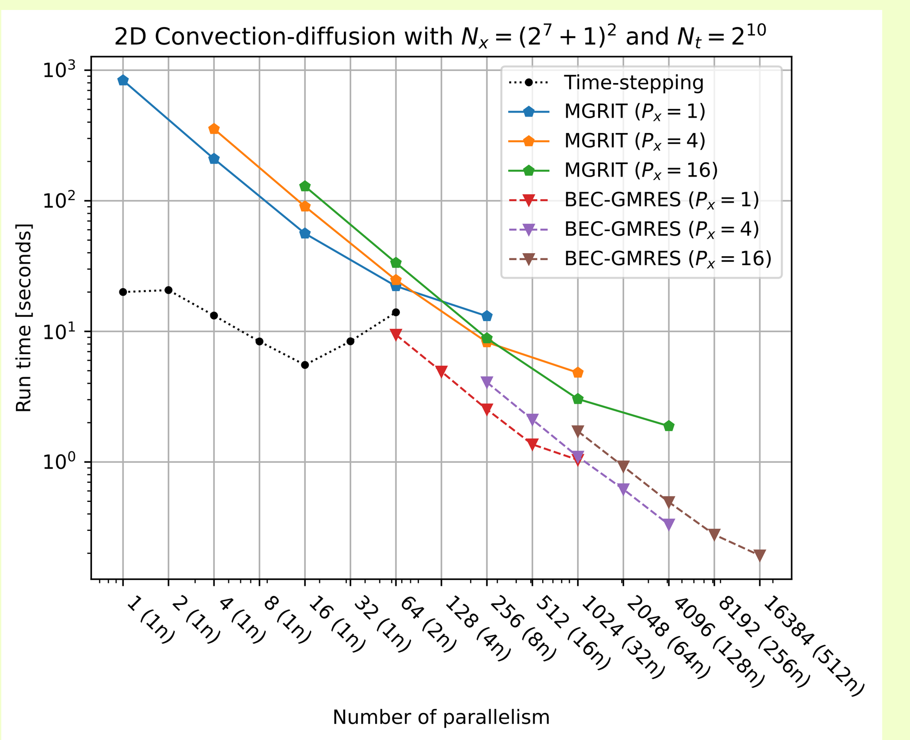


Figure 8: Strong scaling with $n_x = (2^7+1)^2$ and $n_t = 2^{10}$

- MGRIT iterations increase for conv. dominated problems.
- BEC-GMRES still has good convergence.
- Much of the time is spent on complex solver parts.

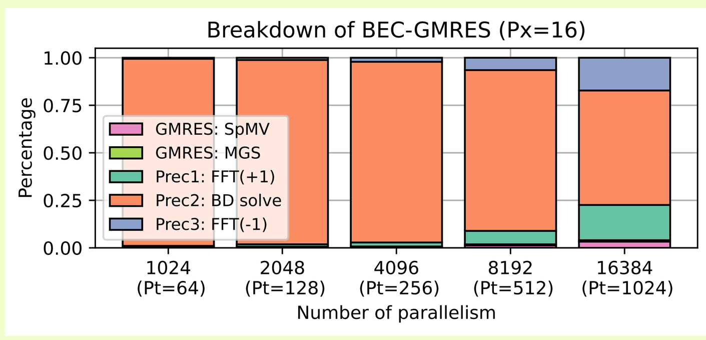


Figure 9: Breakdown of BEC-GMRES with $P_x = 16$

Conclusion

- The redistributed FFT, which assigns temporal parallelism to the spatial domain, achieves stable and good scaling performance.
- BEC-GMRES achieves good temporal-parallelism scaling performance up to maximum temporal parallelism.
- Future work will reduce the time spent on complex solver parts.

References

[1] Xue-lei Lin and Michael Ng. 2021. An All-at-Once Preconditioner for Evolutionary Partial Differential Equations. SIAM Journal on Scientific Computing 43, 4 (2021), A2766–A2784.

[2] Eleanor McDonald, Jennifer Pestana, and Andy Wathen. 2018. Preconditioning and Iterative Solution of All-atOnce Systems for Evolutionary Partial Differential Equations. SIAM Journal on Scientific Computing 40, 2 (2018), A1012–A1033.

[3] Matteo Frigo and Steven G. Johnson. 2005. The Design and Implementation of FFTW3. Proc. IEEE 93, 2 (2005), 216–231. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[4] Heroux, M. A., Bartlett, R. A., Howle, V. E., Hoekstra, R. J., Hu, J. J., Kolda, T. G., ... and Stanley, K. S. An overview of the Trilinos project. ACM Transactions on Mathematical Software (TOMS), 31(3), (2005) 397–423.

[5] R. D. Falgout, S. Friedhoff, Tz. V. Kolev, S. P. MacLachlan, and J. B. Schroder. 2014. Parallel Time Integration with Multigrid. SIAM Journal on Scientific Computing 36, 6 (2014), C635–C661.