A well-conditioned **direct** parallel-in-time (PinT) solver for evolutionary PDEs [*]

Jun Liu (juliu@siue.edu)

Department of Mathematics and Statistics
Southern Illinois University Edwardsville (SIUE)

[*] Joint work with Drs Xiang-Sheng Wang (University of Louisiana at Lafayette) and Shu-Lin Wu (Northeast Normal University, China)

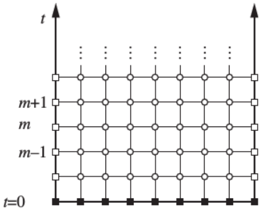
Outline

1D Heat PDE: time-stepping with Backward Euler scheme

$$\begin{aligned} &y_t(x,t) = y_{xx}(x,t), \quad (x,t) \in (0,1) \times (0,T], \\ &y(0,t) = y(1,t) = 0, \quad t \in [0,T], \text{(Boundary Condition)} \\ &y(x,0) = f(x), \quad x \in (0,1). \text{(Initial Condition)} \end{aligned} \tag{1}$$

Uniform space-time mesh $\{x_i = ih, t_m = m\tau\}$ with $h = 1/(M+1), \tau = T/N_t$.

Let $y_i^m \approx y(x_i, t_m)$ and $y_i^0 = f(x_i)$.
The **backward Euler scheme** reads:



$$\frac{y_i^m - y_i^{m-1}}{\tau} = \frac{y_{i-1}^m - 2y_i^m + y_{i+1}^m}{h^2},$$

Let $y_h^m = [y_1^m, y_2^m, \cdots, y_M^m]^\mathsf{T}$, it gives sequential time-stepping (N_t steps $m = 1, 2, \cdots, N_t$)

$$\left(\tfrac{1}{\tau} I_h + A_h\right) y_h^m = \tfrac{1}{\tau} y_h^{m-1}, \tag{2}$$

where $A_h = \frac{1}{h^2}\mathrm{tridiag}(-1,2,-1) \in \mathbb{R}^{M \times M}$ is sparse.

Goal: to **compute** $y(x,t)$ faster with parallel machines?

Choose the IC $f(x) = \mathbb{1}_{[0.4,0.6]}(x)$ and $T = 0.5$.

Figure 1: Simulation of solving 1D heat eq by backward Euler time-stepping.

The parareal algorithm [Lions et al., 2001] for ODEs IVP

Consider the (semi-discretized) ODEs IVP over $[0, T]$:

$$\boldsymbol{u}'(t) = \boldsymbol{f}(\boldsymbol{u}), \qquad \boldsymbol{u}(0) = \boldsymbol{u}_0 \in \mathbb{R}^M. \tag{3}$$

Define two propagation (time-stepping) operators for (3):

- $\mathsf{G}(t_2, t_1, \boldsymbol{u}_1)$ is a **coarse/cheap** approx. to $\boldsymbol{u}(t_2)$ with IC $\boldsymbol{u}(t_1) = \boldsymbol{u}_1$;
- $\mathsf{F}(t_2, t_1, \boldsymbol{u}_1)$ is a **fine/expensive** approx. to $\boldsymbol{u}(t_2)$ with IC $\boldsymbol{u}(t_1) = \boldsymbol{u}_1$;

Partition $(0, T]$ into $(T_{n-1}, T_n]$ with $0 = T_0 < T_1 < T_2 < \cdots < T_N = T$. The parareal algorithm initializes $\boldsymbol{U}_n^0 \approx \boldsymbol{u}(T_n)$ by coarse time-stepping:

$$\boldsymbol{U}_0^0 = \boldsymbol{u}_0; \quad \boldsymbol{U}_{n+1}^0 = \mathsf{G}(T_{n+1}, T_n, \boldsymbol{U}_n^0), \quad n = 0, 1, \cdots, N-1, \tag{4}$$

and then runs the k-th correction iteration (marching $n = 0, 1, 2, \cdots$ and $U_0^{k+1} = \boldsymbol{u}_0$)

$$U_{n+1}^{k+1} = \underbrace{\mathsf{F}(T_{n+1}, T_n, \boldsymbol{U}_n^k)}_{\text{marching in parallel}} + \underbrace{\mathsf{G}(T_{n+1}, T_n, \boldsymbol{U}_n^{k+1})}_{\text{marching in serial}} - \underbrace{\mathsf{G}(T_{n+1}, T_n, \boldsymbol{U}_n^k)}_{\text{already computed}}. \tag{5}$$

The illustration and limitation of parareal algorithm

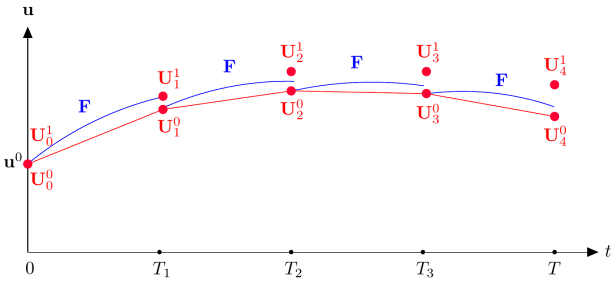
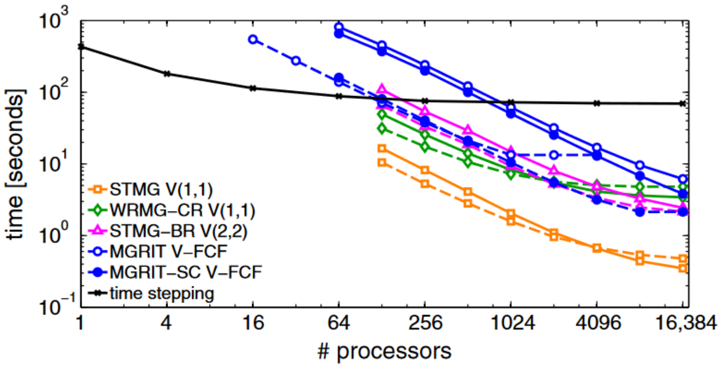


Figure 2: The initialization and 1st iteration of parareal algorithm (red $\boldsymbol{U}_n^k$ are computed in serial and blue $\boldsymbol{F}$ are performed in parallel). See `www.unige.ch/~gander/poly.pdf`

- It always converges to the fine approximation after finite n steps: $\boldsymbol{U}_n^k = \mathsf{F}(T_n, 0, \boldsymbol{u}_0)$ for $k \geq n$. But no speedup if needs $k \geq n$.
- The parallel efficiency $< 1/K$ with K being total iteration number.
- Fast linear convergence for parabolic eq, but poor or no convergence for hyperbolic eq. [Gander and Vandewalle, 2007] (can be improved)

A **fair** comparison for a 2D heat eq using **hypre and XBraid** [Falgout et al., 2017].



- multigrid-reduction-in-time algorithm (MGRIT) [Falgout et al., 2014]
  ☛ a multilevel extension of the **parareal algorithm (2-level)**.
- space-time multigrid method (STMG) ☜ **fastest**
  [Gander and Vandewalle, 2007, Gander and Neumüller, 2016]
- waveform relaxation multigrid (WRMG–CR) [Horton et al., 1995]

A tensor-product direct solver [Maday and Rønquist, 2008]

Couple all the N_t time steps in (2) together:

$$-\frac{1}{\tau}y_h^{m-1} + \left(\frac{1}{\tau}I_h + A_h\right) y_h^m = 0, \qquad m = 1, 2, \cdots, N_t \tag{6}$$

to get an **all-at-once** system (note IC gives $y_h^0 = f_h$)

$$\begin{bmatrix} \frac{1}{\tau}I_h + A_h & & & & \\ -\frac{1}{\tau}I_h & \ddots & & & \\ & -\frac{1}{\tau}I_h & \frac{1}{\tau}I_h + A_h & & \\ & & \ddots & \ddots & \\ & & & -\frac{1}{\tau}I_h & \frac{1}{\tau}I_h + A_h \end{bmatrix} \begin{bmatrix} y_h^1 \\ y_h^2 \\ \vdots \\ \vdots \\ y_h^{N_t} \end{bmatrix} = \begin{bmatrix} \frac{1}{\tau}f_h \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix}. \tag{7}$$

By separating time and space (in Kronecker product $\otimes$), rewrite into

$$L\boldsymbol{y}_h := (B \otimes I_h + I_t \otimes A_h)\, \boldsymbol{y}_h = \boldsymbol{b}_h, \qquad B = \frac{1}{\tau}\begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix} \tag{8}$$

where I_h, I_t are identity matrices. Clearly, B (as a Jordan block) is **not diagonalizable**.

A tensor-product direct PinT solver [Maday and Rønquist, 2008]

But, if all time steps sizes τ_j are **all different**, then it **is indeed diagonalizable**:

$$\widehat{B} = \begin{bmatrix} 1/\tau_1 & & & & \\ -1/\tau_2 & 1/\tau_2 & & & \\ & & \ddots & \ddots & \\ & & & -1/\tau_{N_t} & 1/\tau_{N_t} \end{bmatrix} \xlongequal[\text{with an explicit } V]{\text{diagonalizable}} VDV^{-1}, \tag{9}$$

which leads to a diagonalization-based PinT direct solver by inverting

$$\widehat{L} := \left(\widehat{B} \otimes I_h + I_t \otimes A_h\right) = (V \otimes I_h)(D \otimes I_h + I_t \otimes A_h)(V^{-1} \otimes I_h). \tag{10}$$

Since $(D \otimes I_h + I_t \otimes A_h)$ is **block-diagonal**, inverting $\boldsymbol{y}_h = \widehat{L}^{-1}\boldsymbol{b}_h$ in three steps:

$$\begin{cases} \boldsymbol{g} = (V^{-1} \otimes I_h)\boldsymbol{b}_h, & \text{step-(a)}, \\ (\lambda_j I_h + A_h)w_j = g_j, \ j = 1, 2, \ldots, n, & \text{step-(b), parallel in time}, \\ \boldsymbol{y}_h = (V \otimes I_h)\boldsymbol{w}, & \text{step-(c)}, \end{cases} \tag{11}$$

where $D = \text{diag}(\lambda_1, \ldots, \lambda_n), \boldsymbol{w} = (w_1^{\mathsf{T}}, \ldots, w_n^{\mathsf{T}})^{\mathsf{T}}$ and $\boldsymbol{g} = (g_1^{\mathsf{T}}, \ldots, g_n^{\mathsf{T}})^{\mathsf{T}}$.

The limitation of $\widehat{B}$-based direct PinT solver

Key limitation:

1. The roundoff errors of y_h (due to diagonalization) are proportional to

$$\mathrm{Cond}_2(V) := \|V\|_2 \|V^{-1}\|_2.$$

2. With optimized $\tau_j = (1+\varepsilon)^j$, $\mathrm{Cond}_2(V)$ **shows exponential growth**, which limits $N_t \lesssim 25$ for stable solutions. [Gander et al., 2016, Gander et al., 2019]
3. The restriction $N_t \lesssim 25$ implies only $\sim$25 processors are useful, although it can still be applied to many short time windows in serial. **Less efficient for large N_t.**

Two ways to control $\mathrm{Cond}_2(V)$:

- Iterative: preconditioning B by an α-circulant matrix C_α (diagonalizable by FFT)
- 👉Direct: modify the time scheme to get a 'normal' B with well-conditioned V

A normal matrix is diagonalizable by a unitary matrix (with condition number=1)!

A block α-circulant PinT preconditioner [Lin and Ng, 2020]

Recall the all-at-once nonsymmetric system matrix

$$L := B \otimes I_h + I_t \otimes (-A_h), \qquad B = \frac{1}{\tau}\begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix}. \tag{12}$$

A block α-circulant PinT preconditioner has the form (C_α is diagonalizable by FFT)

$$P_\alpha := C_\alpha \otimes I_h + I_t \otimes (-A_h), \quad C_\alpha = \frac{1}{\tau}\begin{bmatrix} 1 & & & & -\alpha \\ -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \end{bmatrix} \tag{13}$$

where the Toeplitz matrix B is replaced by α-circulant matrix (with $\alpha \in (0,1]$)

- For a sufficiently small α, preconditioned GMRES or fixed-point iteration has a mesh-independent convergence rate (nontrivial to prove due to non-normality).
- To control the roundoff errors in diagonalization, α should not be too small.
- The similar PinT preconditioner also works for wave PDE [Liu and Wu, 2020].

Revisit a boundary value method [Axelsson and Verwer, 1985]

Consider linear ODEs IVP (e.g. from semi-discretized PDE)

$$u'(t) + Au(t) = g(t), \quad u(0) = u_0 \in \mathbb{R}^M. \tag{14}$$

Use a uniform mesh $t_j = j\tau$ with $\tau = T/n$. Applying centered difference for the first $(n-1)$ time steps and backward Euler scheme for the last step, we get

$$\begin{cases} \frac{u_{j+1}-u_{j-1}}{2\tau} + Au_j = g_j, \; j = 1, 2, \ldots, n-1, \\ \frac{u_n - u_{n-1}}{\tau} + Au_n = g_n. \end{cases} \tag{15}$$

Couple all the n time steps in (15) to get an all-at-once system with a **better** B:

$$(B \otimes I_h + I_t \otimes A)\,\boldsymbol{u}_h = \boldsymbol{g}_h, \qquad B = \frac{1}{\tau}\begin{bmatrix} 0 & 1/2 & & & \\ -1/2 & 0 & 1/2 & & \\ & \ddots & \ddots & \ddots & \\ & & -1/2 & 0 & 1/2 \\ & & & -1 & 1 \end{bmatrix} \tag{16}$$

Such an all-at-once time scheme should not be solved in a time-stepping fashion.

The diagonalization of the matrix B: formulas for n eigenpairs

Consider the tridiagonal matrix

$$\mathbb{B} = \tau B = \begin{bmatrix} 0 & 1/2 & & & \\ -1/2 & 0 & 1/2 & & \\ & \ddots & \ddots & \ddots & \\ & & -1/2 & 0 & 1/2 \\ & & & -1 & 1 \end{bmatrix}.$$

Let $T_n(x) = \cos(n \arccos x)$ and $U_n(x) = \sin[(n+1)\arccos x]/\sin(\arccos x)$ denotes the n-th degree Chebyshev polynomials of the 1st- and 2nd-kind.

Theorem 1

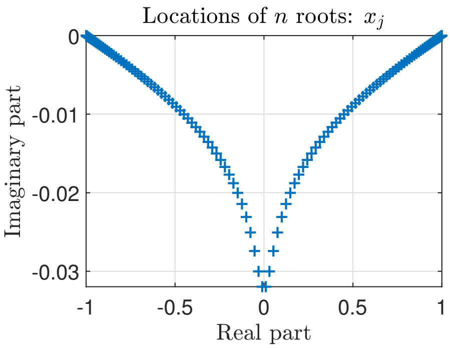
Let $\mathrm{i} = \sqrt{-1}$. The eigenvalues of $\mathbb{B}$ are $\lambda_j = \mathrm{i}x_j$, with $\{x_j\}_{j=1}^n$ being the n roots of $U_{n-1}(x) - \mathrm{i}T_n(x) = 0$, and the corresponding eigenvector $\boldsymbol{p}_j = [p_{j,0}, \cdots, p_{j,n-1}]^\mathsf{T}$ with $p_{j,0} = 1$ is $p_{j,k} = \mathrm{i}^k U_k(x_j)$, $k = 0, \cdots, n-1$.

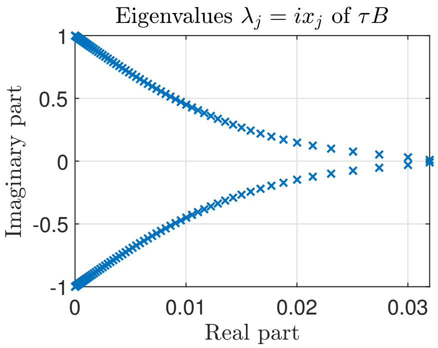
The diagonalization of the matrix B: eigenvalues are simple

Theorem 2

*All n roots of $U_{n-1}(x) - \mathrm{i}T_n(x) = 0$ are **simple**, complex with negative imaginary parts, and have modulus less than $1 + 1/\sqrt{2n}$. Moreover, if x is a root, then so is $-\bar{x}$.*

In particular, n distinct eigenvalues implies $\mathbb{B}$ or B is indeed **diagonalizable**.





The diagonalization of the matrix B: condition number of V

Theorem 3

Let $V = [\boldsymbol{p}_1, \boldsymbol{p}_2, \cdots, \boldsymbol{p}_n]$ *be the eigenvector matrix. There holds* $\mathrm{Cond}_2(V) = O(n^2)$.

Explicitly, the eigenvector matrix V has the Chebyshev-Vandermonde structure

$$V = \underbrace{\mathrm{diag}\left(\mathrm{i}^0, \mathrm{i}^1, \cdots, \mathrm{i}^{n-1}\right)}_{:=\mathbf{I}\text{ unitary}} \underbrace{\begin{bmatrix} U_0(x_1) & \cdots & U_0(x_n) \\ \vdots & \cdots & \vdots \\ U_{n-1}(x_1) & \cdots & U_{n-1}(x_n) \end{bmatrix}}_{:=\Phi}. \tag{17}$$

(i) Some fast inversion algorithms (e.g. [Gohberg and Olshevsky, 1994]) for V^{-1} in the literature may be unstable, mainly due to complex nodes $\{x_i\}_{i=1}^n$.
(ii) By the **Vandermonde-like structure** of $V = [\mathrm{i}^k U_k(x_j)]$, we designed a **stable** fast algorithm with $O(n^2)$ complexity to compute V^{-1}. Based on 3 terms recursion.

Extension to 2nd-order problems (e.g. wave eq): $B \dashrightarrow B^2$

Consider the 2nd-order nonlinear ODEs IVP:

$$u''(t) + f(u(t)) = 0, u(0) = u_0, u'(0) = \tilde{u}_0, \tag{18}$$

Rewritten into first-order ODEs (with $v = u'$) and then apply the same scheme:

$$\begin{bmatrix} u(t) \\ v(t) \end{bmatrix}' = \begin{bmatrix} v(t) \\ -f(u(t)) \end{bmatrix}, \quad \begin{bmatrix} u(0) \\ v(0) \end{bmatrix} = \begin{bmatrix} u_0 \\ \tilde{u}_0 \end{bmatrix}, \tag{19}$$

but **the system size is doubled**. Luckily, we can eliminate v's after discretization:

Lemma 4 (reduced all-at-once system for (18))

The boundary-value method for the 2nd-order problem (18) can be formulated into

$$(B^2 \otimes I_x)\boldsymbol{u} + F(\boldsymbol{u}) = \boldsymbol{b}, \tag{20}$$

where $\boldsymbol{b} = \left(\frac{\tilde{u}_0^\top}{2\tau}, -\frac{u_0^\top}{4\tau^2}, 0, \ldots, 0 \right)^\top$. *Hence no extra cost due to* $B^2 = VD^2V^{-1}$.

Fast diagonalization algorithms for $B = VDV^{-1}$

Based on Theorem 1 it holds $\lambda_j = \mathrm{i}\cos(\theta_j)$, where θ_j is the j-th root of $\rho(\theta) := \sin(n\theta) - \mathrm{i}\cos(n\theta)\sin\theta = 0$. Applying 1D Newton's iteration leads to

$$\theta_j^{(l+1)} = \theta_j^{(l)} - \rho(\theta_j^{(l)})/\rho'(\theta_j^{(l)}),\ l = 0, 1, 2, \cdots . \tag{21}$$

with n **accurately** estimated complex initial guesses: $\theta_j^{(0)} = \frac{1}{2}\left(\frac{j\pi}{n} + \frac{j\pi}{n+1}\right) + \frac{\mathrm{i}}{n}$.

Table 1: Comparison of `eig+backslash` function and our fast diagonalization algorithm

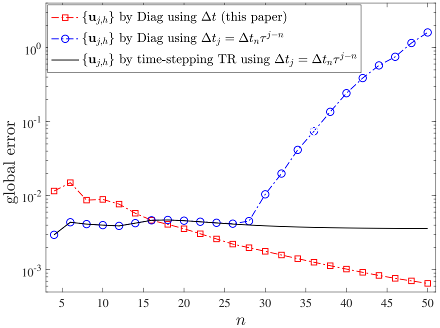
	MATLAB's `eig+backslash`		Our fast algorithm with MATLAB codes			
n	CPU	$\text{Res}_{\texttt{eig}}$	Iter	CPU	$\text{Res}_{\texttt{fast}}$	$\text{EigError}_{\texttt{fast}}$
512	0.277	9.69e-13	8	0.073	5.30e-11	8.89e-15
1024	1.107	3.85e-12	9	0.301	2.04e-10	2.63e-14
2048	6.741	1.01e-11	9	1.206	5.12e-10	1.25e-13
4096	60.257	4.02e-11	10	5.054	6.75e-09	5.16e-13
8192	**606.045**	2.25e-10	10	23.402	2.85e-08	4.07e-13

Approximation accuracy and $\mathrm{Cond}_2(V)$ comparison

$$u_{tt} - u_{xx} = 0,\ u(x,0) = \sin(2\pi x),\ u'(x,0) = 0,\ (x,t) \in (-1,1) \times (0,T), \quad (22)$$

with periodic BC $u(-1,t) = u(1,t)$. With centered difference in space it gives

$$\boldsymbol{u}_h'' + A_h \boldsymbol{u}_h = 0,\ \boldsymbol{u}_h(0) = \boldsymbol{u}_0,\ \boldsymbol{u}_h'(0) = 0,\ t \in (0,T). \quad (23)$$



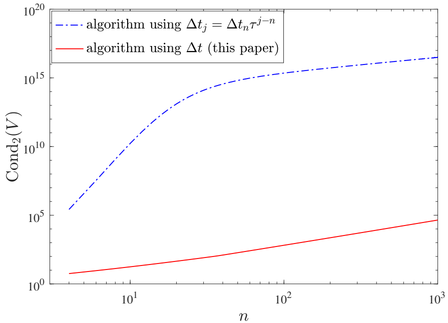


Figure 3: Left: the errors for our algorithm, the algorithm in [Gander et al., 2019] and the time-stepping TR. Right: comparison of the growing $\mathrm{Cond}_2(V)$.

A 2D linear heat eq: scaling results with C/MPI/PETSc

$$u_t(x,y,t) - \Delta u(x,y,t) = r(x,y,t), \text{ in } \Omega \times (0,T), \tag{24}$$

where $\Omega = (0,\pi)^2$ and the exact solution reads $u(x,y,t) = \sin(x)\sin(y)e^{-t}$.

Table 2: Error and Scaling results of example 1: a heat PDE ($T = 2$ with $M = 512^2$)

Core#	strong scaling					weak scaling			
s	n	Error	CPU	Speedup	Efficiency	n	Error	CPU	Efficiency
1	512	2.23e-06	1318.8	1.0	100.0%	2	7.93e-02	5.4	100.0%
2	512	2.23e-06	667.8	2.0	98.7%	4	1.19e-02	5.4	100.0%
4	512	2.23e-06	346.4	3.8	95.2%	8	3.22e-03	5.4	100.0%
8	512	2.23e-06	173.0	7.6	95.3%	16	8.26e-04	5.5	98.2%
16	512	2.23e-06	90.7	14.5	90.9%	32	2.09e-04	5.8	93.1%
32	512	2.23e-06	51.1	25.8	80.7%	64	5.28e-05	6.6	81.8%
64	512	2.23e-06	32.0	41.2	64.4%	128	1.37e-05	8.3	65.1%
128	512	2.23e-06	23.0	57.3	44.8%	256	4.25e-06	12.0	45.0%
256	512	2.23e-06	19.4	68.0	26.6%	512	2.23e-06	19.6	27.6%

- Run on SIUE Campus Cluster: 10 CPU nodes via 25 Gbps Ethernet network, each node has two 3.5GHz AMD EPYC 7F52 16-Core CPU and 256GB RAM. Slow network may affect parallel efficiency. Here $512^3 \approx 134$ millions

A 2D linear wave eq: difficult for parareal-type algorithms

$$u_{tt}(x,y,t) - \Delta u(x,y,t) = r(x,y,t), \text{ in } \Omega \times (0,T), \quad (25)$$

where $\Omega = (0,1)^2$ and the exact solution $u(x,y,t) = x(x-1)y(y-1)\sin(2\pi t)$.

Table 3: Error and Scaling Results of Example 2: a wave PDE ($T = 2$ with $M = 512^2$)

Core#	Strong scaling					Weak scaling			
s	n	Error	CPU	Speedup	Efficiency	n	Error	CPU	Efficiency
1	512	7.88e-05	1328.6	1.0	100.0%	2	9.19e-03	5.4	100.0%
2	512	7.88e-05	676.3	2.0	98.2%	4	2.21e-02	5.4	100.0%
4	512	7.88e-05	332.6	4.0	99.9%	8	3.16e-01	5.5	100.0%
8	512	7.88e-05	172.6	7.7	96.2%	16	1.33e-01	5.7	100.0%
16	512	7.88e-05	91.2	14.6	91.0%	32	2.30e-02	6.0	94.8%
32	512	7.88e-05	51.7	25.7	80.3%	64	5.21e-03	7.1	82.1%
64	512	7.88e-05	31.2	42.6	66.5%	128	1.27e-03	9.5	67.9%
128	512	7.88e-05	23.2	57.3	44.7%	256	3.16e-04	14.8	46.6%
256	512	7.88e-05	20.3	65.4	25.6%	512	7.88e-05	27.4	28.2%

- The parallel efficiency is **the same** as heat eq, without any extra treatments;

Extension to nonlinear cases: simplified Newton iterations

Consider the nonlinear ODEs IVP: $u'(t)+f(u)=0,\ u(0)=u_0$. The scheme reads

$$\begin{cases} \frac{u_{j+1}-u_{j-1}}{2\tau}+f(u_j)=0,\ j=1,2,\dots,n-1, \\ \frac{u_n-u_{n-1}}{\tau}+f(u_n)=0, \end{cases} \tag{26}$$

Rewrite (26) into all-at-once form (with $F(\boldsymbol{u})=[f^\intercal(u_1),f^\intercal(u_2),\dots,f^\intercal(u_n)]^\intercal$)

$$(B\otimes I_x)\boldsymbol{u}+F(\boldsymbol{u})=\boldsymbol{b}.$$

The Newton's iteration (with $\nabla F(\boldsymbol{u}^k)=\texttt{blkdiag}(\nabla f(u_1^k),\dots,\nabla f(u_n^k))$) reads

$$(B\otimes I_x+\nabla F(\boldsymbol{u}^k))\boldsymbol{u}^{k+1}=\boldsymbol{b}+\left(\nabla F(\boldsymbol{u}^k)\boldsymbol{u}^k-F(\boldsymbol{u}^k)\right). \tag{27}$$

Following the idea [Gander and Halpern, 2017] of averaging Jacobian matrix

$$\nabla F(\boldsymbol{u}^k)\approx {\color{red}I_t}\otimes A_k, \qquad A_k=\frac{1}{n}\left(\sum_{j=1}^n\nabla f(u_j^k)\right), \tag{28}$$

which gives a simplified Newton iteration (SNI) can be solved PinT again:

$$(B\otimes I_x+I_t\otimes A_k)\boldsymbol{u}^{k+1}=\boldsymbol{b}+\left((I_t\otimes A_k)\boldsymbol{u}^k-F(\boldsymbol{u}^k)\right). \tag{29}$$

A semi-linear parabolic equation: with $f(u) = u^3 - u$

$$u_t(x,y,t) - \Delta u(x,y,t) + f(u) = r(x,y,t), \text{ in } \Omega \times (0,T), \quad (30)$$

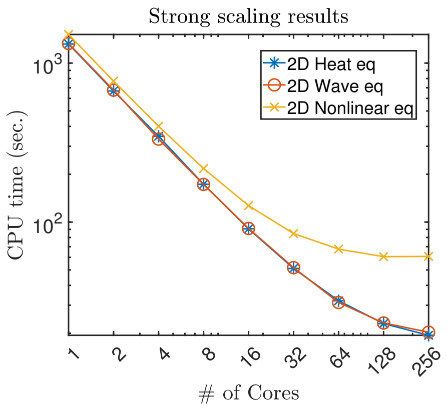
where $\Omega = (-1,1)^2$ and the exact solution reads $u(x,y,t) = (x^2-1)(y^2-1)e^{-t}$.

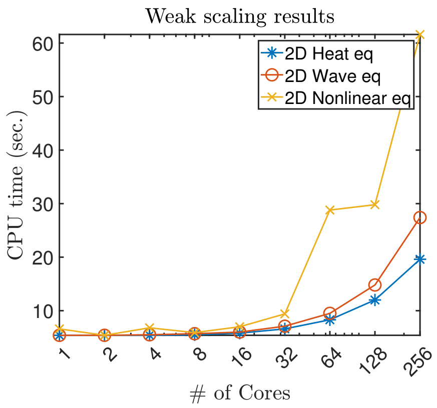
Table 4: Error and Scaling Results of Example 3: a semi-linear PDE ($T = 2$ with $M = 256^2$)

Core#	Strong scaling						Weak scaling				
s	n	Error	SNI	CPU	Speedup	Efficiency	n	Error	SNI	CPU	Efficiency
1	512	6.36e-07	9	1514.0	1.0	100.0%	2	4.40e-01	11	6.6	100.0%
2	512	6.36e-07	9	770.4	2.0	98.3%	4	7.64e-03	9	5.4	122.2%
4	512	6.36e-07	9	400.6	3.8	94.5%	8	2.33e-03	11	6.8	97.1%
8	512	6.36e-07	9	217.2	7.0	87.1%	16	6.38e-04	9	5.9	111.9%
16	512	6.36e-07	9	126.9	11.9	74.6%	32	1.63e-04	9	7.0	94.3%
32	512	6.36e-07	9	84.7	17.9	55.9%	64	4.07e-05	9	9.4	70.2%
64	512	6.36e-07	9	67.6	22.4	35.0%	128	1.02e-05	9	28.8	22.9%
128	512	6.36e-07	9	60.6	25.0	19.5%	256	2.55e-06	9	29.8	22.1%
256	512	6.36e-07	9	60.8	24.9	9.7%	512	6.36e-07	9	61.6	10.7%

- The SNI shows a mesh-independent linear convergence rate
- A clear drop in efficiency when across computer nodes (32 cores per node), but over 20 times speedup is still higher than the parareal (< 10 times).

The summarized strong/weak scaling results





- The nonlinear case's parallel efficiency is much lower, which is expected due to the extra communication cost in (i) distributing the averaged Jacobian matrices and (ii) dispatching the residuals during the sequential SNI iterations.
- The current weak scaling results is less satisfactory; no MPI coding experience.

Summary

Major contribution:

- New direct PinT solver works equally well for parabolic and hyperbolic eqs
- The condition number of eigenvector matrix V can be controlled ($O(n^2)$)
- Fast $O(n^2)$ diagonalization (i.e. D, V^{-1}) algorithms of B are designed

Some open problems:

- The convergence (rate) analysis for nonlinear problems is still missing?
- How to efficiently solve n independent complex-shifted systems in step-(b)? multigrid, model order reduction [Liu and Wang, 2020]
- Which time scheme leads to a diagonalizable B with minimal $\mathrm{Cond}(V)$? optimize parametric time scheme for better conditioned V
- Use adaptive time step sizes for improved accuracy e.g., adaptive parareal algorithm [Maday and Mula, 2020]

Thank you for your attention! Questions?

Iterative and Direct ParaDIAG related algorithms with C/MPI codes:

`github.com/wushulin/ParaDIAG`

Main reference:

- **Jun Liu**, Xiang-Sheng Wang, Shu-Lin Wu, and Tao Zhou, *A well-conditioned direct PinT algorithm for first-and second-order evolutionary equations*, arXiv preprint, 2021. `arxiv.org/abs/2108.01716`. Under minor revision.

References I

[Axelsson and Verwer, 1985] Axelsson, A. O. H. and Verwer, J. G. (1985).
Boundary value techniques for initial value problems in ordinary differential equations.
Math. Comp., 45:153–171.

[Falgout et al., 2014] Falgout, R. D., Friedhoff, S., Kolev, T. V., MacLachlan, S. P., and Schroder, J. B. (2014).
Parallel time integration with multigrid.
SIAM Journal on Scientific Computing, 36:C635–C661.

[Falgout et al., 2017] Falgout, R. D., Friedhoff, S., Kolev, T. V., MacLachlan, S. P., Schroder, J. B., and Vandewalle, S. (2017).
Multigrid methods with space–time concurrency.
Computing and Visualization in Science, 18(4):123–143.

[Gander et al., 2019] Gander, M., Halpern, L., Rannou, J., and Ryan, J. (2019).
A direct time parallel solver by diagonalization for the wave equation.
SIAM Journal on Scientific Computing, 41(1):A220–A245.

[Gander and Halpern, 2017] Gander, M. J. and Halpern, L. (2017).
Time parallelization for nonlinear problems based on diagonalization.
In Lee, C.-O., Cai, X.-C., Keyes, D. E., Kim, H. H., Klawonn, A., Park, E.-J., and Widlund, O. B., editors, *Domain Decomposition Methods in Science and Engineering XXIII*, pages 163–170. Springer International Publishing.

References II

[Gander et al., 2016] Gander, M. J., Halpern, L., Ryan, J., and Tran, T. T. B. (2016).
A Direct Solver for Time Parallelization.
In Dickopf, T., Gander, M. J., Halpern, L., Krause, R., and Pavarino, L. F., editors, *Domain Decomposition Methods in Science and Engineering XXII*, pages 491–499. Springer International Publishing.

[Gander and Neumüller, 2016] Gander, M. J. and Neumüller, M. (2016).
Analysis of a new space-time parallel multigrid algorithm for parabolic problems.
SIAM Journal on Scientific Computing, 38(4):A2173–A2208.

[Gander and Vandewalle, 2007] Gander, M. J. and Vandewalle, S. (2007).
Analysis of the Parareal Time-Parallel Time-Integration Method.
SIAM Journal on Scientific Computing, 29(2):556–578.

[Gohberg and Olshevsky, 1994] Gohberg, I. and Olshevsky, V. (1994).
Fast inversion of Chebyshev–Vandermonde matrices.
Numer. Math., 67:71–92.

[Horton et al., 1995] Horton, G., Vandewalle, S., and Worley, P. (1995).
An Algorithm with Polylog Parallel Complexity for Solving Parabolic Partial Differential Equations.
SIAM Journal on Scientific Computing, 16(3):531–541.

[Lin and Ng, 2020] Lin, X.-L. and Ng, M. (2020).
An all-at-once preconditioner for evolutionary partial differential equations.
arXiv preprint arXiv:2002.01108.

References III

[Lions et al., 2001] Lions, J.-L., Maday, Y., and Turinici, G. (2001).
A "parareal" in time discretization of PDE's.
Comptes Rendus de l'Académie des Sciences - Series I - Mathematics, 332:661–668.

[Liu and Wang, 2020] Liu, J. and Wang, Z. (2020).
A rom-accelerated parallel-in-time preconditioner for solving all-at-once systems from evolutionary pdes.

[Liu and Wu, 2020] Liu, J. and Wu, S.-L. (2020).
A fast block α-circulant preconditoner for all-at-once systems from wave equations.
SIAM Journal on Matrix Analysis and Applications, 41(4):1912–1943.

[Maday and Mula, 2020] Maday, Y. and Mula, O. (2020).
An adaptive parareal algorithm.
Journal of computational and applied mathematics, 377:112915.

[Maday and Rønquist, 2008] Maday, Y. and Rønquist, E. M. (2008).
Parallelization in time through tensor-product space-time solvers.
Comptes Rendus Mathematique, 346(1–2):113–118.