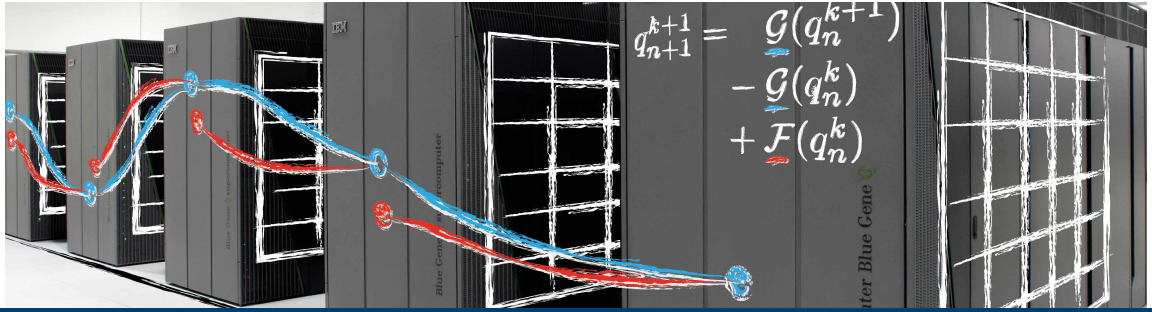
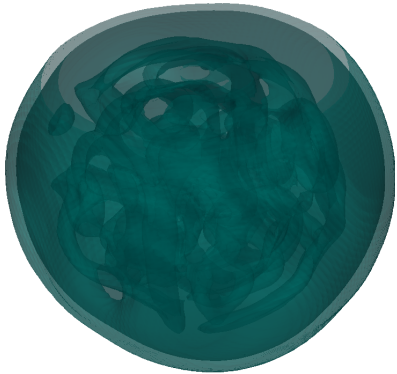
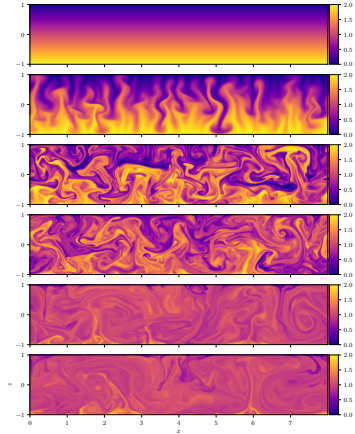



# Massively parallel adaptive spectral deferred correction in Python

March 6, 2025 | Thomas Baumann | Jülich Supercomputing Centre

# Want to solve: Gray-Scott and Rayleigh-Benard convection





Integrate space-time-parallel with spectral space discretization and IMEX SDC

# Spectral deferred correction (SDC)

**[Dutt et al., 2000]**

## Goal

Solve initial value problem (IVP): $u_t = f(u), \quad u(0) = u_0$

## SDC idea

- Integrate IVP over $(0,\Delta t]$: $u(\Delta t) = u_0 + \int_0^{\Delta t} f(u(s))ds$
- Discretize with spectral quadrature rule: $\vec{u} = \vec{u}_0 + \Delta t Q f(\vec{u})$
  $\rightarrow$ $\vec{u}$ are intermediate solutions at quadrature nodes
  $\rightarrow$ defines fully implicit Runge-Kutta method (RKM)
- Solve fully implicit RKM iteratively using preconditioned Picard iteration

## SDC iteration

$(I - \Delta t Q_\Delta f)\left(\vec{u}^{k+1}\right) = \vec{u}_0 + \Delta t\,(Q - Q_\Delta)\, f\left(\vec{u}^{k}\right), \quad Q_\Delta$: preconditioner

# Spectral methods in space

## Use FFT to compute global expansion in Fourier or Chebychev polynomials

$$u(x_n) \approx \sum_{k=0}^{N-1} \hat{u}_k \phi_k(x_n), \quad u(x_n) \xrightarrow{\text{FFT}} \hat{u}_k, \quad \hat{u}_k \xrightarrow{\text{iFFT}} u(x_n)$$

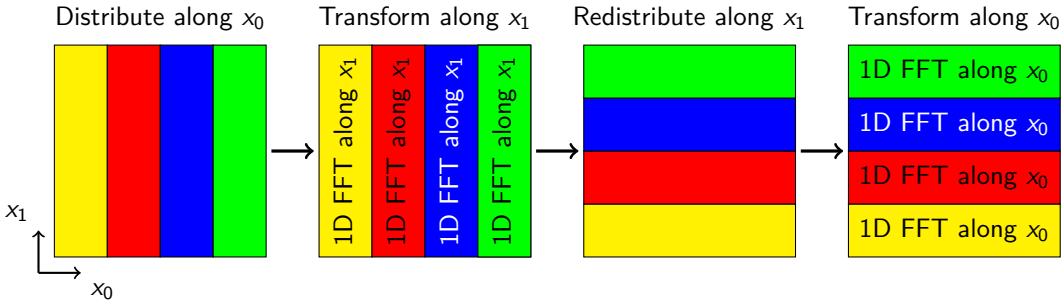
## Evaluate derivatives by multiplication with derivative matrix $D$

$$\partial_x u(x) \approx \sum_{k=0}^{N-1} \hat{u}_k \partial_x \phi_k(x), \quad (\partial_x u)(x_n) \approx \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} D_{kl} \hat{u}_l \phi_l(x_n)$$

- Fourier base: $D$ diagonal $\rightarrow$ evaluate and invert distributedly
- Chebychev base: $D$ dense $\rightarrow$ precondition [Olver and Townsend, 2013], can't parallelize

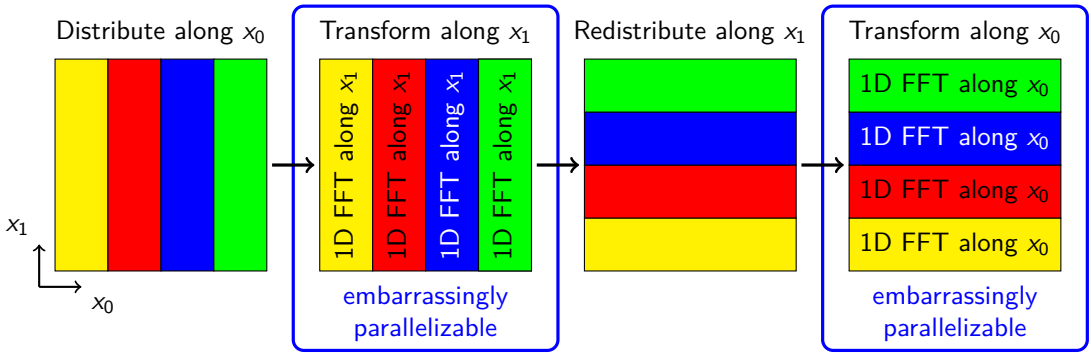
# Distributed FFTs

Distributed ND FFT: Successive concurrent 1D FFTs



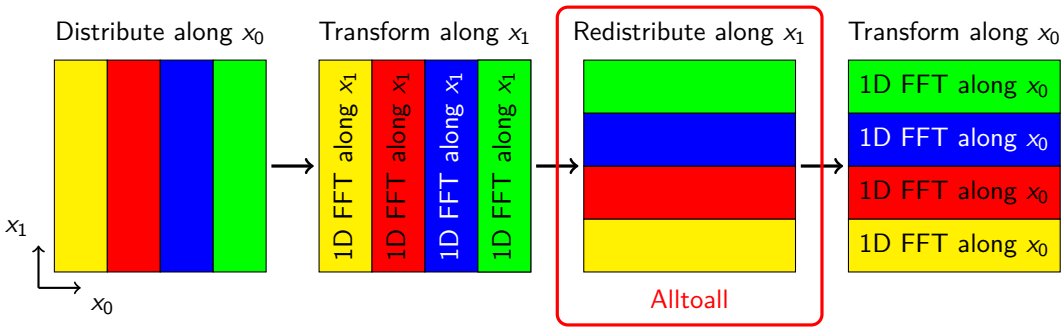
# Distributed FFTs

Distributed ND FFT: Successive concurrent 1D FFTs



# Distributed FFTs

Distributed ND FFT: Successive concurrent 1D FFTs

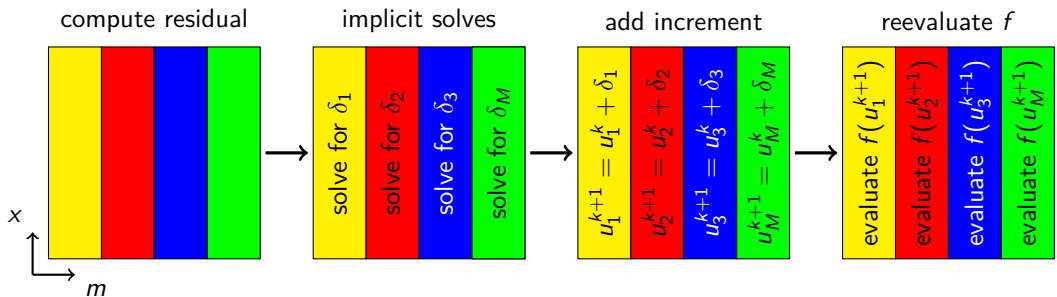


# Diagonal SDC

**[van der Houwen and Sommeijer, 1991, Speck, 2018, Čaklović et al., 2024]**

For $Q_\Delta$ diagonal, SDC iteration becomes

$$(1 - \Delta t (Q_\Delta)_{mm} f)(\delta_m) = u_0 + \Delta t \sum_{j=1}^{M} (Q)_{mj} f\left(u_m^k\right) - u_m^k \qquad u_m^{k+1} = u_m^k + \delta_m$$

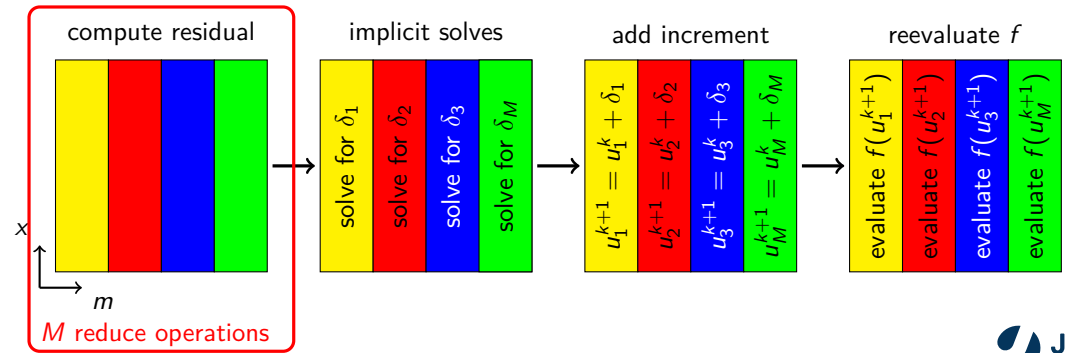


# Diagonal SDC

**[van der Houwen and Sommeijer, 1991, Speck, 2018, Čaklović et al., 2024]**

For $Q_\Delta$ diagonal, SDC iteration becomes

$$(1 - \Delta t (Q_\Delta)_{mm} f)(\delta_m) = u_0 + \Delta t \sum_{j=1}^{M} (Q)_{mj} f\left(u_m^k\right) - u_m^k \qquad u_m^{k+1} = u_m^k + \delta_m$$

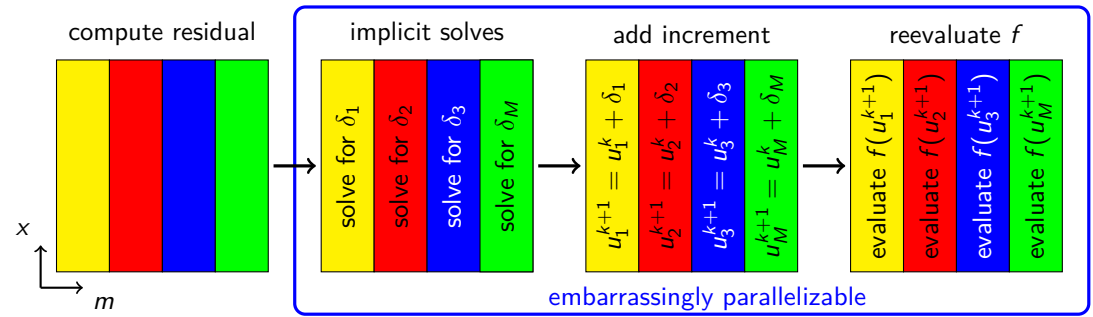


# Diagonal SDC

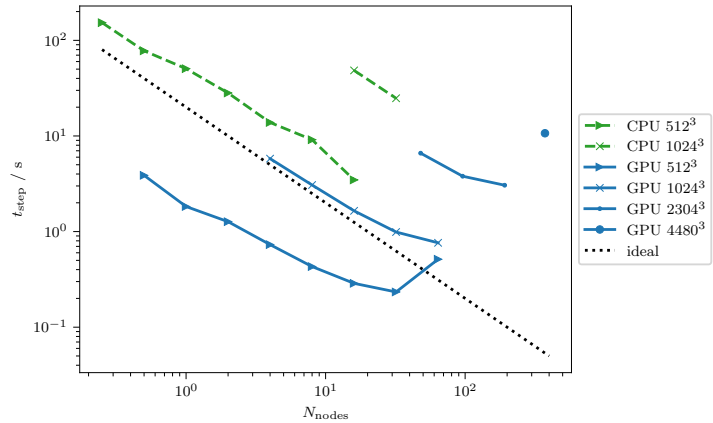
**[van der Houwen and Sommeijer, 1991, Speck, 2018, Čaklović et al., 2024]**

For $Q_\Delta$ diagonal, SDC iteration becomes

$$(1 - \Delta t (Q_\Delta)_{mm} f)(\delta_m) = u_0 + \Delta t \sum_{j=1}^{M} (Q)_{mj} f\left(u_m^k\right) - u_m^k \qquad u_m^{k+1} = u_m^k + \delta_m$$



# Parallel scaling of Gray-Scott implementation



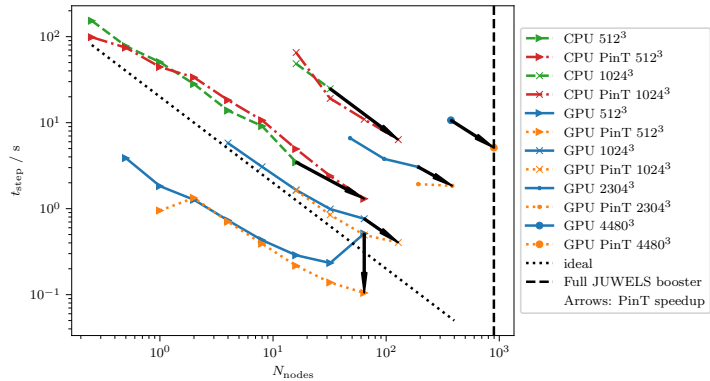

## GPUs > CPUs

- 16 CPUs per node
- 4 GPUs per node
- ≈ 10x speedup with GPUs

## Diagonal SDC extends scaling

- Uses 4 tasks in time
- Shifts from all-to-all to reduce
- Improves strong scaling
- Enables scaling up to 3584 GPUs → entire JUWELS booster

# Parallel scaling of Gray-Scott implementation



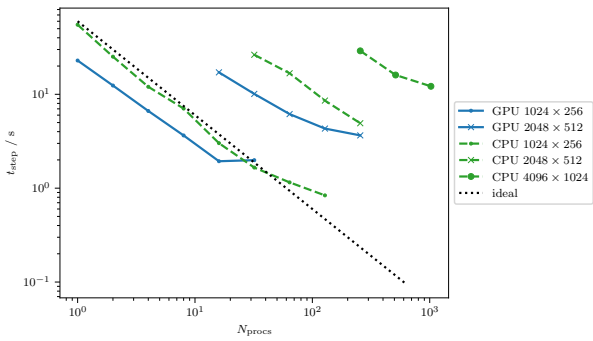

## GPUs > CPUs

- 16 CPUs per node
- 4 GPUs per node
- ≈ 10x speedup with GPUs

## Diagonal SDC extends scaling

- Uses 4 tasks in time
- Shifts from all-to-all to reduce
- Improves strong scaling
- Enables scaling up to 3584 GPUs → entire JUWELS booster

# Parallel scaling of RBC implementation



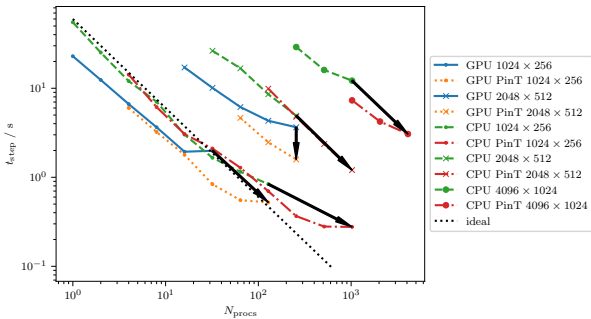

## CPUs vs GPUs

- Sparse linear solves not accelerated on GPU
- FFTs faster on GPU

## Use diagonal SDC to extend scaling

- Uses 4 tasks in time
- Circumvents space-decomposition limit
- Improves strong scaling
- Enables scaling up to 4096 CPUs

# Parallel scaling of RBC implementation



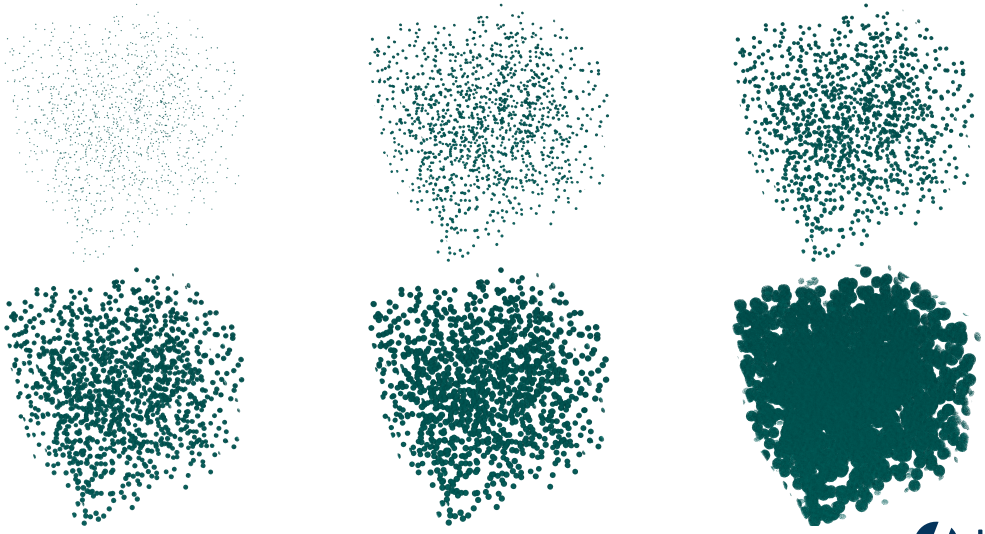

## CPUs vs GPUs

- Sparse linear solves not accelerated on GPU
- FFTs faster on GPU

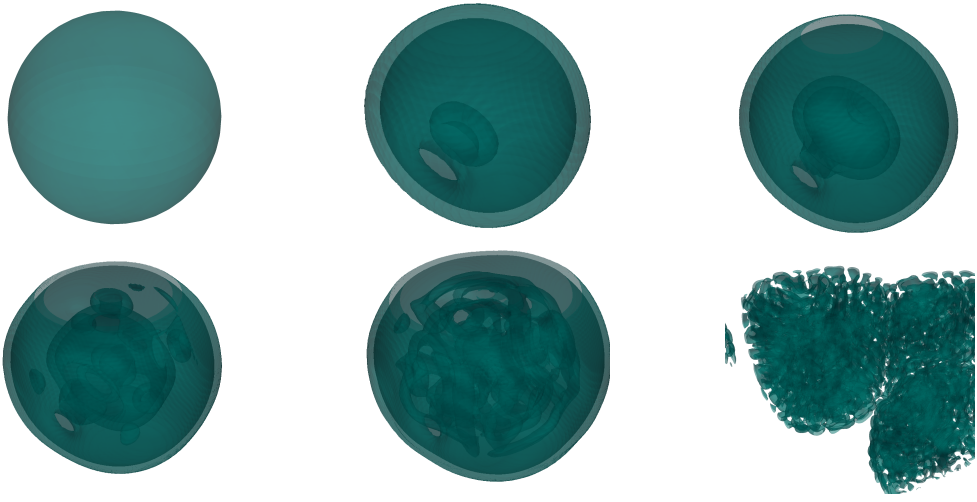
## Use diagonal SDC to extend scaling

- Uses 4 tasks in time
- Circumvents space-decomposition limit
- Improves strong scaling
- Enables scaling up to 4096 CPUs

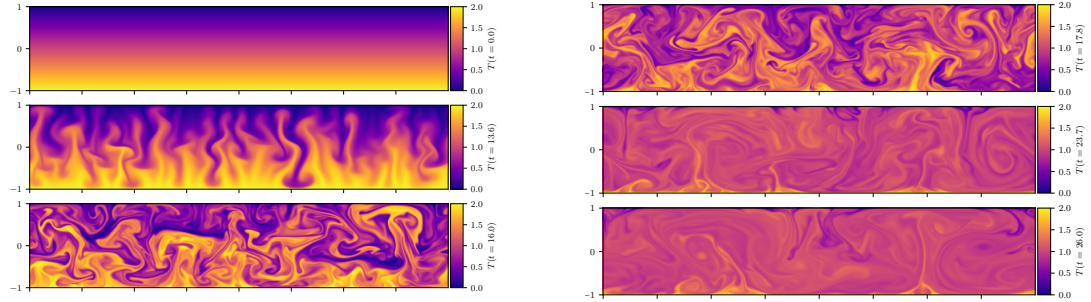
# Large space-time parallel Gray-Scott simulation



# Large space-time parallel Gray-Scott simulation: Zoom



# Large space-time parallel Rayleigh-Benard simulation




- Use Rayleigh number $3.2 \times 10^8$ on $N = 4096 \times 1024$ grid
- Use $\Delta t$-$k$-adaptivity with four Gauß-Radau nodes for step size selection
- Use $4 \times 1024 = 4096$ CPUs on JURECA DC (32 nodes)
- Use approx. 140k CPU hours to reach $t = 26$ because IMEX stability limit requires $\Delta t \approx 10^{-3}$

# Codes used and acknowledgements

GPU port of mpi4py-fft


## pySDC [Speck, 2019]

- prototyping library for SDC up to PFASST
- highly modular: Problems, sweepers, parallel, serial, ...
- on GitHub with extensive continuous integration

pySDC


## Collaborators

- Sebastian Götschel (TUHH)
- Thibaut Lunet (TUHH)
- Daniel Ruprecht (TUHH)
- Robert Speck (FZJ)

# Sources I



Čaklović, G., Lunet, T., Götschel, S., and Ruprecht, D. (2024).
Improving efficiency of parallel across the method spectral deferred corrections.

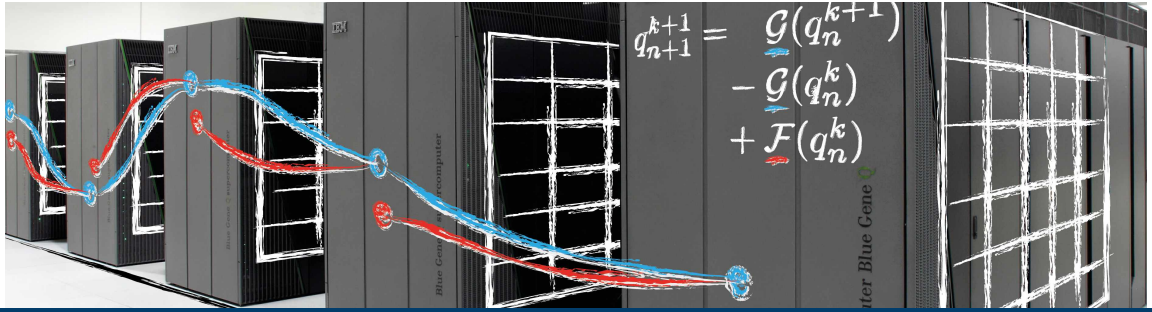

Dutt, A., Greengard, L., and Rokhlin, V. (2000).
Spectral deferred correction methods for ordinary differential equations.
*BIT Numerical Mathematics*, 40(2):241–266.


Olver, S. and Townsend, A. (2013).
A fast and well-conditioned spectral method.
*SIAM Review*, 55(3):462–489.


Speck, R. (2018).
Parallelizing spectral deferred corrections across the method.
*Computing and Visualization in Science*, 19(3):75–83.


Speck, R. (2019).
Algorithm 997: pySDC—Prototyping spectral deferred corrections.
*ACM Trans. Math. Softw.*, 45(3).


van der Houwen, P. J. and Sommeijer, B. P. (1991).
Iterated runge–kutta methods on parallel computers.
*SIAM Journal on Scientific and Statistical Computing*, 12(5):1000–1028.




# Massively parallel adaptive spectral deferred correction in Python

March 6, 2025 | Thomas Baumann | Jülich Supercomputing Centre

# Porting mpi4py-fft to GPU: Alltoall in NCCL

$N$,$r$ ← Number of GPUs, MPI rank ▷ Initialize variables
sendbufs, sendbufs ← {}, {}

NCCL group start ▷ Launch all communications in a single kernel
**for** i in 0, 1, ..., N: **do**
    send_to ← (r + i) % N
    recv_from ← (r - i + N) % N
    sendbufs[i] ← contiguous copy of data to send
    recvbufs[i] ← contiguous empty array
    NCCL receive from recv_from
    NCCL send to send_to
**end for**
NCCL group end ▷ Wait for all communication to finish before unpacking

copy recvbufs to destination array

Record all of this to CUDA graph!

# Porting mpi4py-fft to GPU: Alltoall in NCCL

$N$,$r$ ← Number of GPUs, MPI rank ▷ Initialize variables
sendbufs, sendbufs ← {}, {}

NCCL group start ▷ Launch all communications in a single kernel
**for** i in 0, 1, ..., N: **do**
    send_to ← (r + i) % N
    recv_from ← (r - i + N) % N
    sendbufs[i] ← contiguous copy of data to send
    recvbufs[i] ← contiguous empty array
    NCCL receive from recv_from
    NCCL send to send_to
**end for**
NCCL group end ▷ Wait for all communication to finish before unpacking

copy recvbufs to destination array

Record all of this to CUDA graph!