PAUL SCHERRER INSTITUT PSI

ETH zürich

Veronica Montanaro
ETH Zürich

A memory-saving, type-independent improvement to an open boundary FFT solver in IPPL 2.0

Master thesis

October 10, 2023

Overview

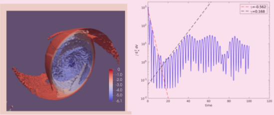
Background

OPAL and the IPPL Framework

ALPINE: A set of portable pLasma physics Particle-in-cell mINi-apps for Exascale

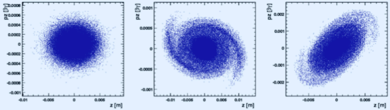
Two-stream Instability | Penning trap | Landau damping

- Light weight codes
- Proxy for real applications
- For implementing new algorithms
- Testing new implementations
- Reference results ensure correctness



OPAL: Object oriented Parallel Accelerator Library

Open source C++ Library for particle accelerators being developed by twelve core developers across seven institutes



Source: OPAL web page

Independent Parallel Particle Layer (IPPL) v 2.0

- FFT based Poisson Solver
- CG
- heFFTe
- D-Operators
- Interpolation
- Fields
- Mesh
- Particles
- Load Balancing
- Domain Decomp.
- Communication
- Buffer factory
- MPI
- Kokkos

Background

Electrostatic Particle in Cell

Initialize
Macroparticles and fields' initial conditions are applied. Grid is superimposed on the domain.

Scatter
Compute ρ by interpolating charge onto the grid points.

Solve
Solve field equation on the grid to get $\vec{E}$'s value.

Gather
Interpolate $\vec{E}$ field values on the particle's position and compute force.

Push
Update particle's velocity and position to the next time-step.

Introduction

Problem to solve as in OPAL (Adelmann et al., 2019)

Poisson equation: $\Delta\phi = \rho.$

$\rightarrow$ Can write the solution as a convolution (Ryne, 2011):

$$\boxed{\phi(\vec{r}) = (G * \rho)(\vec{r}) = \int G(\vec{r} - \vec{r}')\rho(\vec{r}')d\vec{r}'}$$

$\implies$ can use **FFT-based** methods

$$\phi = h_x h_y h_z \mathrm{FFT}^{-1}\{\mathrm{FFT}\{\rho\} \cdot \mathrm{FFT}\{G\}\}$$

Periodic Boundary Conditions
$\rightarrow$ Can use directly

Open Boundary Conditions
$\rightarrow$ Make periodic (Hockney and Eastwood, 1988)

Courtesy of Sonali Mayani (LSM, PSI), 2023

State-of-the-art Open boundaries FFT Solver

The standard Hockney trick (Hockney and Eastwood, 1988)

Procedure

- Double the domain $\implies (2N)^3$.
- On doubled grid, make periodic $\implies \rho_2, G_2$.
- Use FFT to compute convolution.
- Restrict solution to the physical domain.

Problem

Green's function needs to be regularized at $G(0) = -\frac{1}{4\pi}$.

$\implies$ Only second-order convergence.

Adapted from Sonali Mayani (LSM, PSI), 2023

A novel method for Open boundaries: Vico et al. (2016)

Trick

Pre compute Green's function with an approximation in Fourier space:

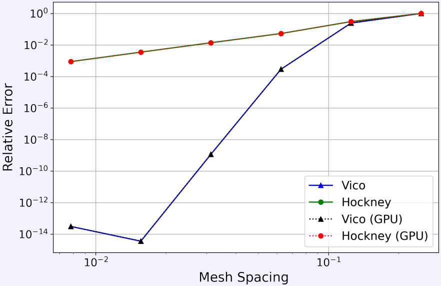
$$G_L(\vec{s}) = 2\left(\frac{\sin(L|\vec{s}|/2)}{|\vec{s}|}\right)^2,$$

where $L > \sqrt{L_x^2 + L_y^2 + L_z^2}$.

$FFT^{-1}\{G_L\}$ is defined at 0.

What do we gain?

Spectral convergence for smooth Gaussian source:



Adapted from Sonali Mayani (LSM, PSI), 2023

Discrete Cosine Transform for Vico method

Problem

G_L Requires **pre-computation** on a $4N^3$ grid to satisfy sampling theorem.

Large memory footprints $\implies$ Inability of running large problems, especially on GPUs.

$\longrightarrow$

A proposed solution

Exploit symmetry of G_L around its $(2N+1)$-th element.

- Pre-compute only the first $2N+1$ coefficients of G_L.
- With an inverse Discrete Cosine Transform compute $G = \mathrm{DCT}^{-1}\{G_L\}$.
- Solve Poisson equation via convolution as in Hockney.

$\rightarrow$ domain size reduced to $2N^3$.

Discrete Cosine Transform for Vico method

DCT definition and choice

DCT $\equiv$ DFT of twice the length operating on periodic and symmetric coefficients.

What we need: **DCT of type 1 (DCT-I)**.

$$\boxed{Y_k = \frac{1}{2(N-1)}\left(x_0 + (-1)^k x_{N-1} + 2\sum_{n=1}^{N-2} x_n \cos\left[\frac{\pi k n}{N-1}\right]\right)}$$

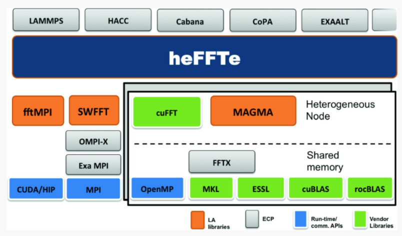
Thesis goals

My goals:

- Implementing the GPU kernels for computing the DCT-I in the Highly Efficient FFT library (heFFTe) $\rightarrow$ used in IPPL for FFT operations.
- Incorporate the algorithmic change in the pre-existing implementation of the Vico solver.
- Make IPPL completely type independent to allow single and mixed precision runs.

Highly Efficient FFT Library for Exascale (Ayala et al., 2020)

Highly linearly scalable FFT algorithms,
designed to target exascale machines.

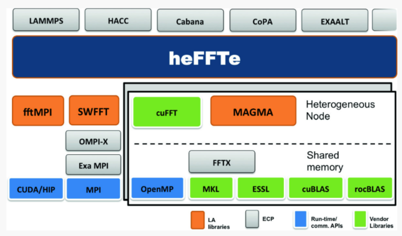


Features

- Vendor libraries as back-ends.

Highly Efficient FFT Library for Exascale (Ayala et al., 2020)

Highly linearly scalable FFT algorithms,
designed to target exascale machines.

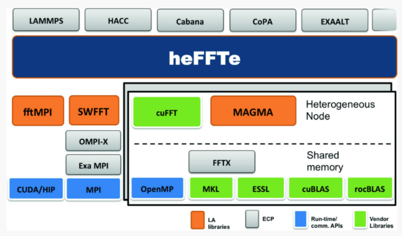


Features

- Vendor libraries as back-ends.
- Reshape+compute algorithm for 3d FFTs.

Highly Efficient FFT Library for Exascale (Ayala et al., 2020)

Highly linearly scalable FFT algorithms,
designed to target exascale machines.



Features

- Vendor libraries as back-ends.
- Reshape+compute algorithm for 3d FFTs.
- Choices for reshaping and communication.

Discrete Cosine Transforms in heFFTe

Native implementation

Only one back-end providing it

Non-native implementation

Rely on R2C and C2R backward and forward transforms, with the help of pre and post processors to match the transform type.

Non-native implementation: algorithm

`inputData` ← data to be transformed

↓

`inputExtended` ← `preProcessingKernel(inputData)`

↓

`outputComplex` ← `R2C(inputExtended)`

↓

`outputReal` ← `postProcessingKernel(outputComplex)`

cuFFT DCT-I implementation

Intuition

Taken a signal x of length N, it is possible to construct its DCT-I through a real to complex DFT.

1. Construct the signal y, of length $4(N-1)$, such that:

$$y[n]_{n\in[0,\dots,4(N-1))} = \begin{cases} x[n/2], & \text{if } n \text{ even and } n \leq 2N \\ x[(4(N-1)-n)/2] & \text{if } n \text{ even and } 2N < n < 4(N-1) \\ 0.0 & \text{otherwise} \end{cases}$$

2. Perform a DFT on the extended signal, which will result in:

$$Y[k] = \frac{1}{4(N-1)} \sum_{n=0}^{4(N-1)-1} y_n e^{\frac{-2\pi ikn}{4(N-1)}}$$

cuFFT DCT-I implementation

Intuition: cont'd

3. Re-write in terms of x :

$$= \frac{1}{4(N-1)} \left[e^{0} x_0 + e^{-\pi i k} x_{N-1} + \sum_{n=1}^{N-2} x_n \left(e^{\frac{-2\pi i k 2n}{4(N-1)}} + e^{\frac{-2\pi i k(4(N-1)-2n)}{4(N-1)}} \right) \right]$$

$$= \frac{1}{4(N-1)} \left[x_0 + (-1)^k x_{N-1} + \sum_{n=1}^{N-2} x_n \left(e^{\frac{-\pi i k n}{N-1}} + e^{\frac{\pi i k n)}{N-1}} e^{-2\pi i k} \right) \right]$$

$$= \frac{1}{2(N-1)} \left[x_0 + (-1)^k x_{N-1} + 2 \sum_{n=1}^{N-2} x_n \cos\left[\frac{\pi k n}{N-1} \right] \right]$$

Which is the definition of the (normalized) DCT of type I.

cuFFT DCT-I implementation

Adjustments

Use these steps to write the processing kernels.

Theoretically, kernels should be the same independently from type of transform (forward/backwards), since $DCT\text{-I}^{-1} = DCT\text{-I}$.

In reality, they have minor differences to fit cuFFT data types.

FFT type	Input size	Output size
C2C	N *cufftComplex*	N *cufftComplex*
C2R	$\lfloor \frac{N}{2} \rfloor + 1$ *cufftComplex*	N *cufftReal*
R2C	N *cufftReal*	$\lfloor \frac{N}{2} \rfloor + 1$ *cufftComplex*

cuFFT DCT-I implementation

Forward kernels

Pre forward

```
// DCT-I (REDFT00)
// even symmetry and periodicity;
//(a b c d) -> (a 0 b 0 c 0  d  0 c 0 b 0)
```

$4(N - 1)$ Real numbers

Pre backward

```
// IDCT-I backward kernel for DCT-I.
// set imaginary parts to zero; even symmetry
// (a b c d) -> (a+0i b+0i c+0i  d+0i  c+0i b+0i 0+0i)
```

$2N - 1$ Complex numbers $\rightarrow 4N - 2$ floating-point numbers.

cuFFT DCT-I implementation

Forward kernels

Post backward/forward kernels

```
void cos1_post_backward_kernel(int N, scalar_type const
     *fft_signal, scalar_type *result){
    int ind = blockIdx.x*BLK_X + threadIdx.x;
    if(ind < N){
        result[ind] = fft_signal[2*ind];
    }
}
```

Same code, different meaning.
Forward → extracts real part out of complex vector.
Backward → extracts even coefficients out of real vector.

Setup

Test case

Six iterations of the solver with

$$\rho_{init}(\vec{r}) = \frac{1}{\sigma^3\sqrt{(2\pi)^3}} \mathrm{e}^{-\frac{(\vec{r}-\mu)^2}{\sigma^2}}$$

Grid points $= \{32^3, 64^3, 96^3, 128^3, 256^3\}$.

Architecture

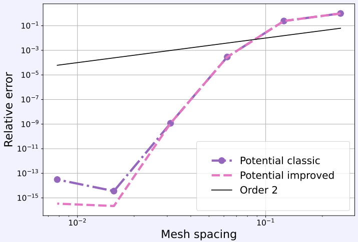
CPU: Two Intel Xeon Gold 6152 (44 CPUs, 380GB).

GPU: One NVIDIA A100 (40GB).

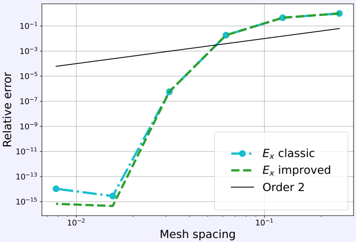
Memory tracker: **Kokkos Memory Highwater.**

Accuracy check

Potential field



Electrostatic field

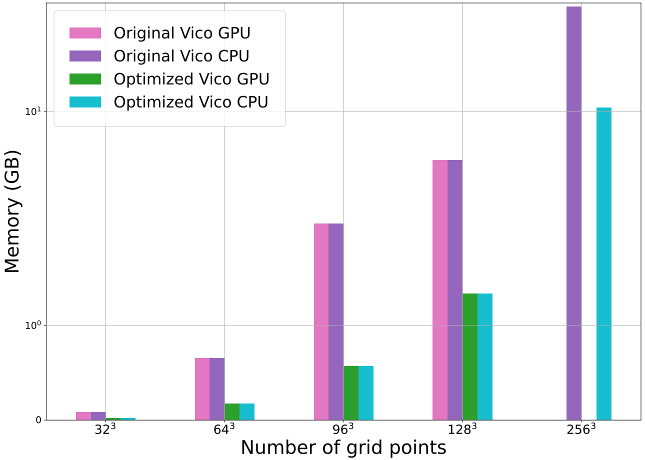


Correctness is verified for the improved method.

We now have better accuracy than the state-of-the-art Hockney (Order 2) without losing to it in memory footprint.

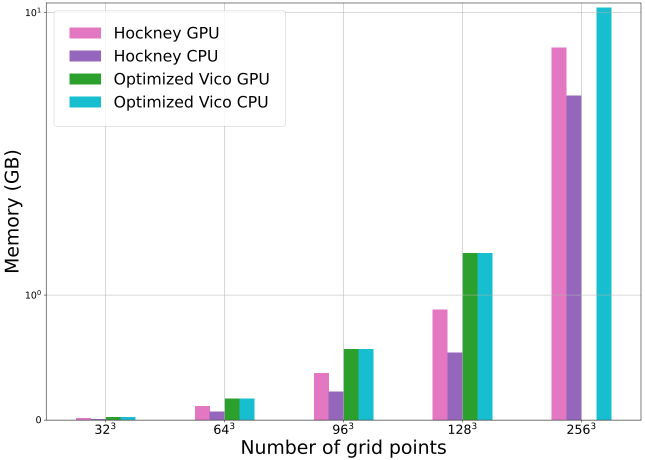
Memory study

Original vs. Optimized Vico



Memory study

Hockney vs. Optimized Vico



Scaling setup

Strong scaling problem sizes

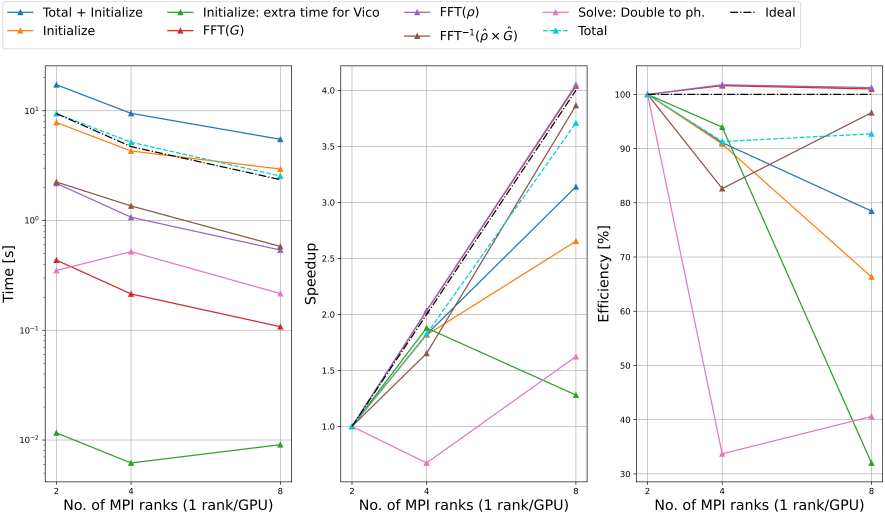
	Improved Vico	$(4N)^3$ Vico
CPU	512^3	256^3
GPU	256^3	128^3

Weak scaling problem sizes

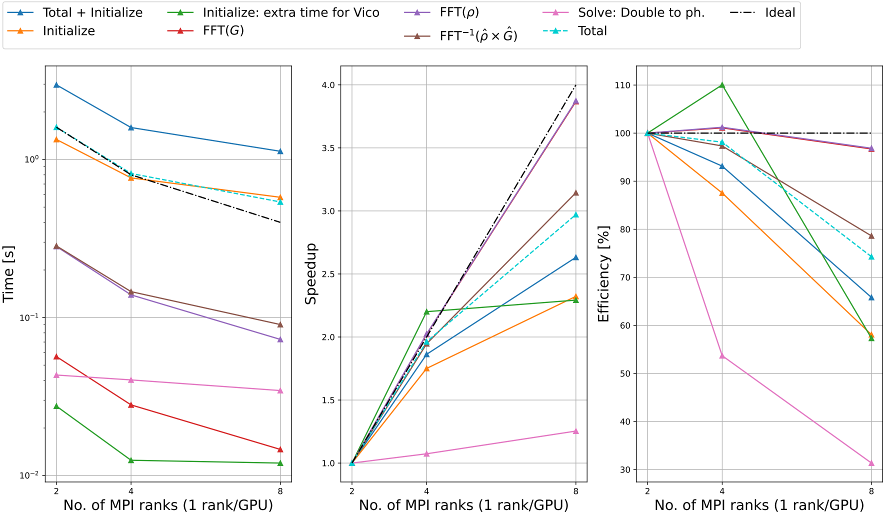
	Improved Vico	$(4N)^3$ Vico
CPU	256^3 to 512^3	128^3 to 256^3
GPU	256^3 to $512^2 \times 256$	128^3 to $256^2 \times 128$

Strong scaling on GPU

Improved Vico



Strong scaling on GPU
$(4N)^3$ Vico



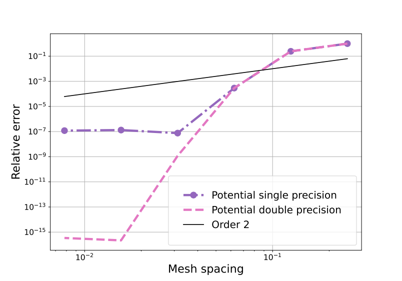
Conclusions

- Implementing the DCT-I in heFFTe $\implies$ between 50% and 70% less memory with respect to the vanilla solver.
- Two GPUs can now fit an additional power of two before exceeding memory (256^3 now, vs 128^3 before).
- $\implies$ We can now **increase the workload per GPUs** to fully exploit them.

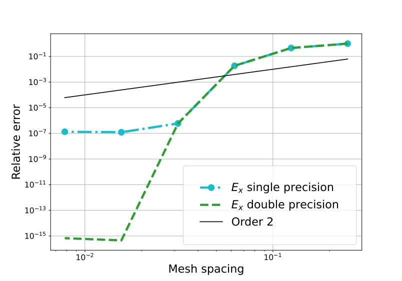
Conclusions
cont'd

- Type independency is also useful for GPU runs $\Longrightarrow$ Less memory, most GPU's machine number is single-precision

Potential field



Electrostatic field



Conclusions
cont'd

- Efficiency of approx. 90% observed in **all scalings** on **all architectures**.
- $\Longrightarrow$ Vico solver is competitive with the state-of-the art Hockney (much better accuracy using approximately the same memory).

	Hockney	Vico (Opt.)	Hockney	Vico (Opt.)
Relative error	Grid size		Memory (MB)	
$\sim 10^{-1}$	8^3	8^3	$\sim 10^{-1}$	$\sim 10^{-1}$
$\sim 10^{-4}$	128^3	16^3	$\sim 10^{1}$	$\sim 10^{1}$
$\sim 10^{-9}$	$\sim (2^{15})^3$	32^3	$\sim 10^{3}$	$\sim 10^{3}$

Aknowledgements

Thanks to everyone who has joined.

Feel free to ask me questions.

References

Adelmann, A., Calvo, P., Frey, M., Gsell, A., Locans, U., Metzger-Kraus, C., Neveu, N., Rogers, C., Russell, S., Sheehy, S., Snuverink, J., and Winklehner, D. (2019). OPAL a Versatile Tool for Charged Particle Accelerator Simulations. *arXiv:1905.06654 [physics]*. arXiv: 1905.06654.

Ayala, A., Tomov, S., Haidar, A., and Dongarra, J. (2020). heFFTe: Highly Efficient FFT for Exascale. In Krzhizhanovskaya, V. V., Závodszky, G., Lees, M. H., Dongarra, J. J., Sloot, P. M. A., Brissos, S., and Teixeira, J., editors, *Computational Science – ICCS 2020*, Lecture Notes in Computer Science, pages 262–275, Cham. Springer International Publishing.

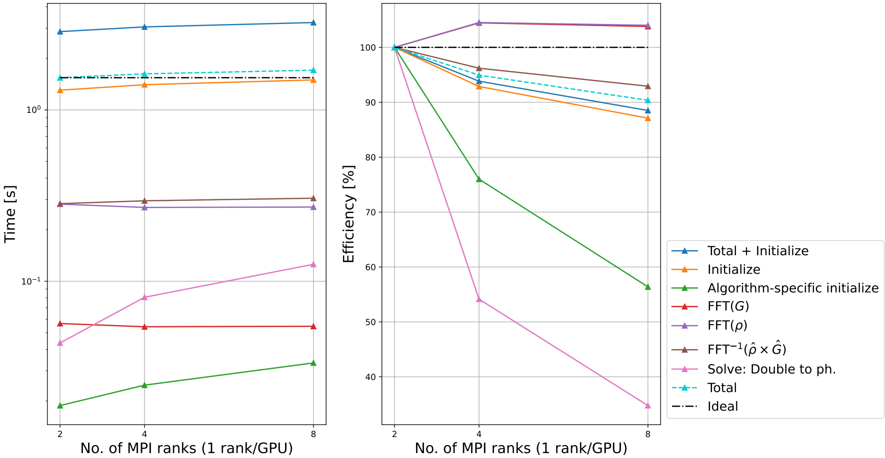
Hockney, R. and Eastwood, J. W. (1988). *Computer Simulation Using Particles*. CRC Press.

Ryne, R. D. (2011). On FFT-based convolutions and correlations, with application to solving Poisson's equation in an open rectangular pipe. *arXiv:1111.4971 [physics]*. arXiv: 1111.4971.

Vico, F., Greengard, L., and Ferrando, M. (2016). Fast convolution with free-space Green's functions. *Journal of Computational Physics*, 323:191–203.

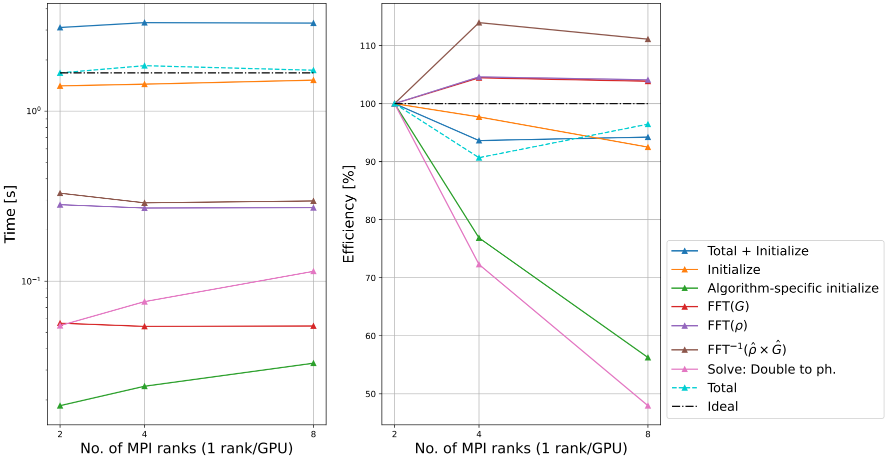
Weak scaling on GPU

Improved Vico



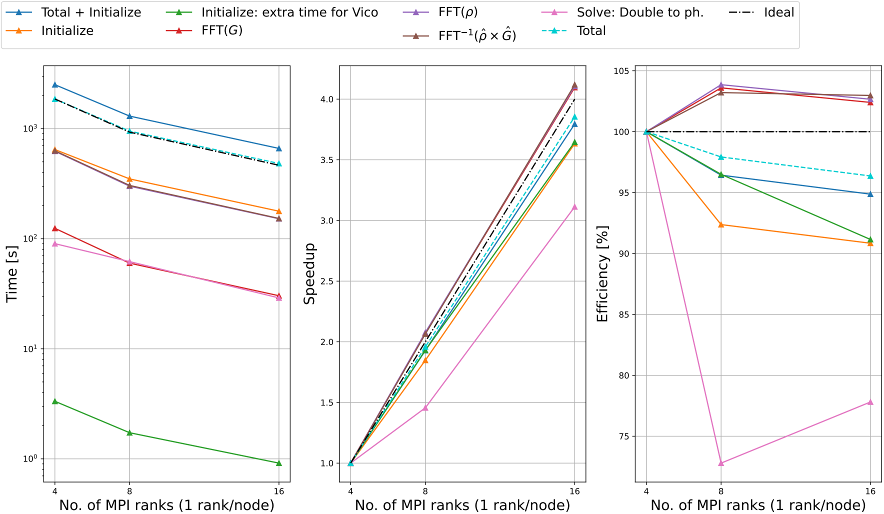
Weak scaling on GPU

$(4N)^3$ Vico



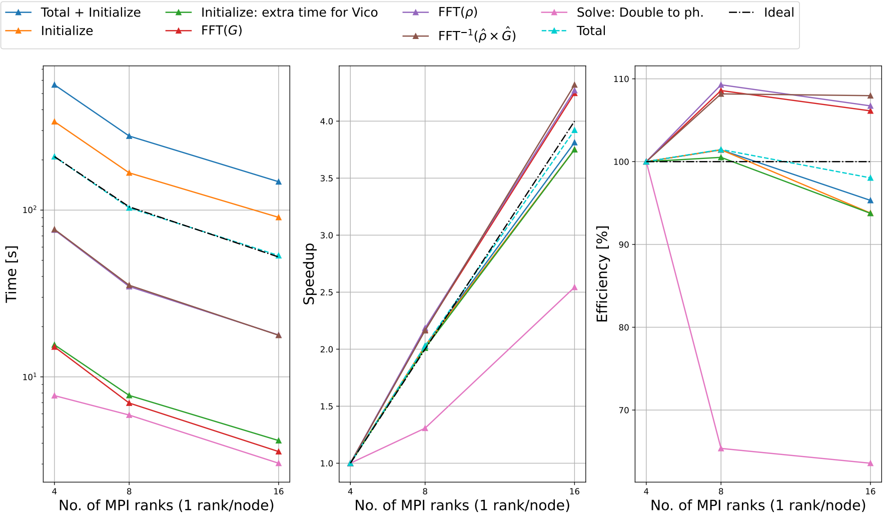
Strong scaling on CPU

Improved Vico



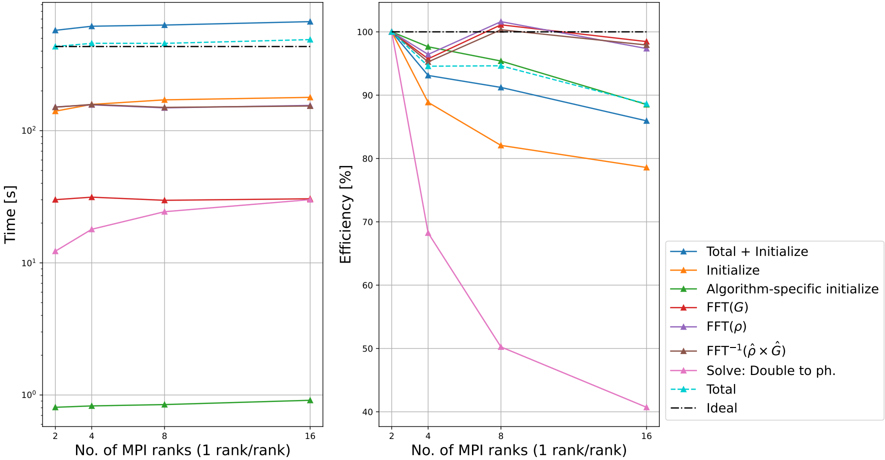
Strong scaling on CPU

$(4N)^3$ Vico



Weak scaling on CPU

Improved Vico



Weak scaling on CPU

$(4N)^3$ Vico