PAUL SCHERRER INSTITUT PSI

ETH zürich

V. Montanaro, S. Mayani, A. Adelmann :: AMAS Group, LSM
S. Tomov, M. Stoyanov :: ICL, UTK

Improvements to a free-space FFT Poisson solver in the Independent Parallel Particle Layer

Master thesis

September 22, 2023

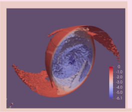
Overview

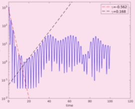
OPAL and IPPL

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

Two-stream Instability | Penning trap | Landau damping

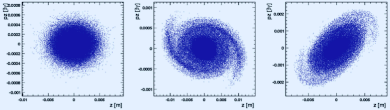
- Light weight codes
- Proxy for real applications
- For implementing new algorithms
- Testing new implementations
- Reference results ensure correctness





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

Introduction: Problem to solve as in OPAL

Adelmann et al. (2019)

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

$\rightarrow$ Can write the solution as a convolution:

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

$\implies$ 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)

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

Based on the standard Hockney trick (Hockney and Eastwood, 1988)

Hockney and Eastwood (1988)

- 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.

Vico et al. (2016)

Green's function in Fourier space:

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

where $L > \sqrt{L_x^2 + L_y^2 + L_z^2}$ is the length of the truncation window.

Motivations

Problem

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

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

Solution

Exploit symmetry of G around its $2N+1$-th element. Instead of inverse-FFT on a domain of size $4N^3$, pre-compute G with an inverse DCT.

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

DCT definition and choice

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

To fit the location of the symmetry centre, we want type I.

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

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:

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

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 :

$$= \frac{1}{4(N-1)}\left[e^{0}x_0 + e^{-\pi ik}x_{N-1} + \sum_{n=1}^{N-2} x_n\left(e^{\frac{-2\pi ik2n}{4(N-1)}} + e^{\frac{-2\pi ik(4(N-1)-2n)}{4(N-1)}}\right)\right]$$

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

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

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];
    }
}
```

In the case of post processing kernels, code is the same but mathematical meaning is different.
Forward → extracts real part out of complex vector.
Backward → extracts even coefficients out of real vector.

Improved Vico: algorithm

Extend ρ to ρ_{2N} on $(2N)^3$ grid.

↓

Pre-compute $G_{2N+1} \leftarrow$ Vico Green's function up to the $2N+1$-th point (up to centre of symmetry).

↓

Compute $DCT^{-}1\{G_{2N+1}\}$ and restrict $G_{2N+1} \rightarrow G_{2N}$ to $(2N)^3$ grid.

↓

Solve Poisson via convolution
$\phi = h_x h_y h_z \mathrm{FFT}^{-1}\{\mathrm{FFT}\{\rho_{2N}\} \cdot \mathrm{FFT}\{G_{2N}\}\}$

Improved Vico

Testing Setup

Test case

Gaussian test (Mayani, 2021)

Runs a single iteration of the solver with a Gaussian source.

Grid points = from 64^3 until memory overflow for the non-optimized case is met.

Architecture

Single CPU node from the **Merlin** cluster at PSI.

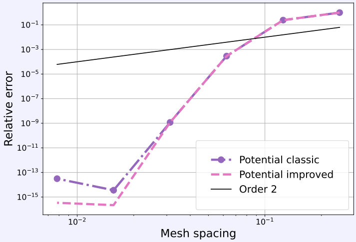
$\rightarrow$ Two Intel Xeon Gold 6152 (44 CPUs, 380GB).

Memory tracker: **Kokkos Memory Highwater.**

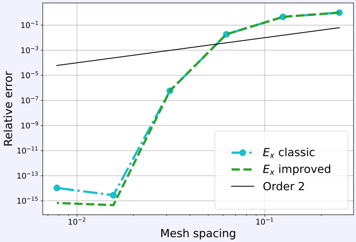
Improved Vico

Accuracy

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.

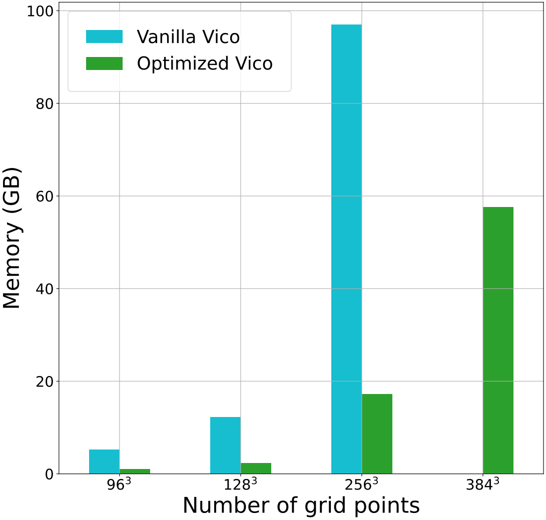
Improved Vico

Memory results

Possible to fit **more problem sizes** $< 512^3$ in a single node **with the optimized version**.

Highwater tracked a memory saving of $[\mathbf{70\%} \div \mathbf{80\%}]$.

Theoretical memory footprint reduction ≈ 8



Next steps

heFFTe

- GPU benchmarking for DCT-I
- Finalize DST-I kernels for sake of completeness
- Add kernels for CPU
- Test mixed-precision communication

IPPL

- Test mixed-precision communication
- Find and fix bug in GPU execution space
- GPU and CPU benchmarking
- Memory study on GPU

Next steps: heFFTe

Testing mixed-precision communication

Goal

Test the accuracy of the mixed precision communication currently being implemented in heFFTe.

$\rightarrow$ Choose physics-based candidate application from IPPL for testing.

Candidate application

Gaussian convergence test with Hockney solver.

$\Longrightarrow$ Well-studied physical solution to check the results.

$\Longrightarrow$ Converges at second order, more accuracy is not needed.

Next steps:IPPL

GPU bug hunting

Problem

`Solver` class fails with any R2R transform when executed on GPU (not only DCT-I).

Checks:

R2R transforms specialised class	FFT transform	Solver class
✓	✓	X

Aknowledgements

Thanks to Prof. Stanimire Tomov and Dr. Miroslav Stoyanov for guiding me, and for the great interest shown throughout the project.

Thanks to the whole of the Linear algebra group and ICL for making my staying possible for these two months.

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.

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

Mayani, S. (2021). A performance portable poisson solver for the hose instability.

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