– PFTC2d – A time-discrete Vlasov approach to LSC driven microbunching in FEL-like beam lines

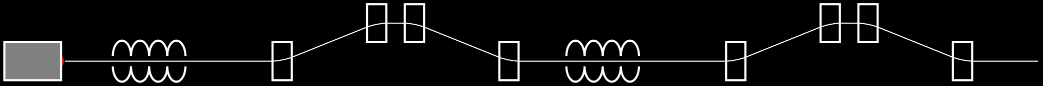
Philipp Amstutz[1] Mathias Vogt[2]

{[1]Universität Hamburg,[2]DESY}, Hamburg, Germany, EU, $\bigoplus$

NOCE 2017

Motivation

- FEL process requires high brightness beams
- High peak current is achieved by bunch compression
- Successive compression at increasing energies $\Longrightarrow$ multi-stage BC

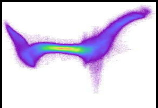


*Chicane model simplified for visualization purposes

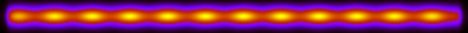
Motivation

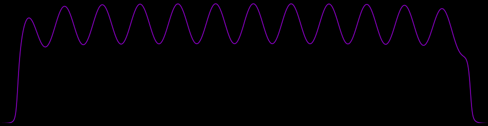
- Bunch compression adds to longitudinal dynamics

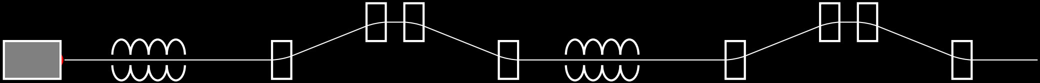
$$\rho\text{-inhomog.} \underset{\text{dispersion}}{\overset{\text{space-charge}}{\rightleftarrows}} \gamma\text{-modulation}$$

- For instance: microbunching amplification
  $\Longrightarrow$ detrimental to FEL process
- 

PFTC2d







*Chicane model simplified for visualization purposes

Bunch-Compressor Model

- Consider longitudinal phase space with coordinates $\vec{z} = \begin{pmatrix} q \\ p \end{pmatrix} = \begin{pmatrix} \xi - \xi_0 \\ E - E_0 \end{pmatrix}$.
- Let $\Psi(\cdot\,; s) \in \mathcal{L}_1(\mathbb{R}^2, \mathbb{R})$ be the phase-space distribution of the bunch at position s.

- Magnetic Chicane
  - Neglect CSR, neglect SR
  - Short, compared to Linac section
    $\implies$ Neglect LSC

$$\boxed{D^{\text{chic}} : \begin{pmatrix} q \\ p \end{pmatrix} \mapsto \begin{pmatrix} q + R(p) \\ p \end{pmatrix}}$$

$$\text{e.g. } R(p) = \frac{R_{56}}{E_0}\, p + \frac{R_{566}}{E_0^2}\, p^2 + \dots$$

- Linac Section
  - ultra-relativistic limit $\implies \partial q / \partial p = 0$
  - Cavity Kick + LSC Kick

$$\boxed{K[\Psi]^{\text{cav,LSC}} : \begin{pmatrix} q \\ p \end{pmatrix} \mapsto \begin{pmatrix} q \\ p + F[\Psi](q) \end{pmatrix}}$$

$$F[\Psi](q) = F^{\text{cav}}(q) + F[\Psi]^{\text{LSC}}(q)$$

$$F[\Psi]^{\text{LSC}}(q) = \int_{\mathbb{R}^2} G(q, q')\, \Psi\left(\vec{z}'\right)\, \mathrm{d}^2 \vec{z}'$$

Phase-Space Density Evolution

- Poisson-type collective terms can't be evaluated for arbitrary PSDs
  - See MV's poster (Tower) on a perturbative approximation
  - We still need numerics
- Simulation via PIC can be noisy at short wavelengths
- Better approach: Track $\Psi(\vec{z}; s)$ itself!
  - Consider invertible, measure-preserving map $\vec{M}_{s\leftarrow 0} : \vec{z}_0 \mapsto \vec{z}_s$
  - Via Liouvilles theorem $\Psi(\vec{z}_0; 0) = \Psi(\vec{z}_s; s)$

$$\boxed{\implies \Psi(\vec{z}; s) = \Psi\big(\vec{M}_{s\leftarrow 0}^{-1}(\vec{z}); 0\big) = \Psi\big(\vec{M}_{0\leftarrow s}(\vec{z}); 0\big)}$$

  - One can define the Perron-Frobenius operator $\mathcal{M} : \mathcal{L}_1(\mathbb{R}^2, \mathbb{R}) \to \mathcal{L}_1(\mathbb{R}^2, \mathbb{R})$ associated to $\vec{M}$ by $\big(\mathcal{M}\Psi\big)(\vec{z}) := \Psi(\vec{M}^{-1}(\vec{z}))$

$$\boxed{\Psi(\cdot\,; s) = \mathcal{M}_{s\leftarrow 0}\Psi(\cdot\,; 0)}$$

Model Properties – Drift Maps

$$\boxed{D : \begin{pmatrix} q \\ p \end{pmatrix} \mapsto \begin{pmatrix} q + R(p) \\ p \end{pmatrix}}$$

- $R \in C^1 \implies \partial D(\vec{z})/\partial \vec{z} \in \mathrm{Sp}(2,\mathbb{R})\,\forall \vec{z} \implies D \in \mathrm{Symp}(2,\mathbb{R})$
- Measure preserving and explicitly invertible
  - $D^{-1} : (q,p)^T \mapsto (q - R(p), p)^T$
- p is invariant under D
- Drift maps form an Abelian sub group of $\mathrm{Symp}(2,\mathbb{R})$

Model Properties – Kick Maps

$$\boxed{K^{\mathrm{cav}}_{l\leftarrow 0} : \begin{pmatrix} q \\ p \end{pmatrix} \mapsto \begin{pmatrix} q \\ p + F^{\mathrm{cav}}(q) \end{pmatrix}} \qquad \boxed{K^{\mathrm{LSC}}_{l\leftarrow 0}[\Psi(\cdot\,;0)] : \begin{pmatrix} q \\ p \end{pmatrix} \mapsto \begin{pmatrix} q \\ p + F[\Psi(\cdot\,;0);l](q) \end{pmatrix}}$$

- $F \in C^1 \implies K[\Psi] \in \mathrm{Symp}(2,\mathbb{R})$
- q is invariant under K
- Kick maps form an Abelian sub group of $\mathrm{Symp}(2,\mathbb{R})$
- $F[\Psi(\cdot;0)]$? But Ψ evolves along the long linac section!??

Model Properties – Kick Maps

$$\boxed{K^{\mathrm{cav}}_{l\leftarrow 0}:\begin{pmatrix} q \\ p \end{pmatrix}\mapsto\begin{pmatrix} q \\ p+F^{\mathrm{cav}}(q) \end{pmatrix}} \qquad \boxed{K^{\mathrm{LSC}}_{l\leftarrow 0}[\Psi(\cdot\,;0)]:\begin{pmatrix} q \\ p \end{pmatrix}\mapsto\begin{pmatrix} q \\ p+F[\Psi(\cdot\,;0);l](q) \end{pmatrix}}$$

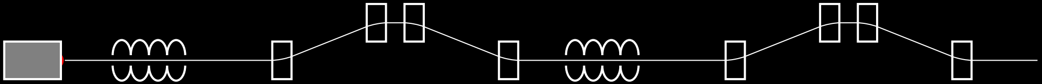
- $F \in C^1 \implies K[\Psi] \in \mathrm{Symp}(2,\mathbb{R})$
- q is invariant under K
- Kick maps form an Abelian sub group of $\mathrm{Symp}(2,\mathbb{R})$
- $F[\Psi(\cdot;0)]$? But Ψ evolves along the long linac section!??
- Spatial density is invariant under (even collective!) kick-type maps

$$\rho(q;s)=\int_{\mathbb{R}}\Psi(\vec{z};s)\,\mathrm{d}p=\int_{\mathbb{R}}\mathcal{K}[\Psi(\cdot\,;0)]_{s\leftarrow 0}\Psi(\vec{z};0)\,\mathrm{d}p=\rho(q;0)$$

- LSC fields can be integrated

$$F[\Psi(\cdot\,;s);s]=\int_0^s\int_{\mathbb{R}}G(q,q'\,;s)\rho(q'\,;s)\mathrm{d}q'\mathrm{d}s=F[\Psi(\cdot\,;0);s]$$

Time-discrete PSD evolution



$$\Psi(\cdot\,;0) \xrightarrow{\mathcal{K}^{\mathrm{cav,LSC}}_{s_1\leftarrow 0}} \Psi(\cdot\,;s_1) \xrightarrow{\mathcal{D}^{\mathrm{chic}}_{s_2\leftarrow s_1}} \Psi(\cdot\,;s_2) \xrightarrow{\mathcal{K}^{\mathrm{cav,LSC}}_{s_3\leftarrow s_2}} \Psi(\cdot\,;s_3) \xrightarrow{\mathcal{D}^{\mathrm{chic}}_{s_4\leftarrow s_3}} \Psi(\cdot\,;s_4)$$

- Model allows for an exact, time-discrete treatment of the beam-line components
- Traversal of a bunch-compressor consists of two simulation steps
- $\mathcal{K}/\mathcal{D}$ are the Perron-Frobenius operators associated to the Kick/Drift-Maps

1D Green's functions

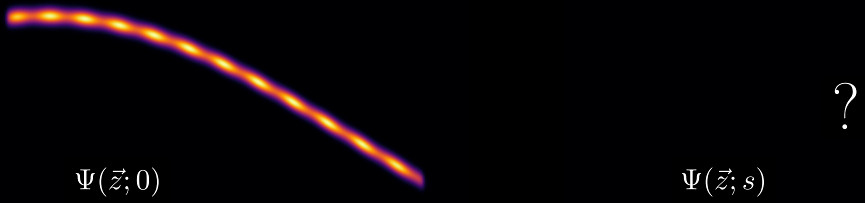
- Poisson's equation is inherently 3D: $\Delta\phi(\vec{x}) = \rho(\vec{x})$
  - 1D model needs assumptions about the other 2D
- We assume an axial symmetric bunch with radial shape $S(r)$
  - $\rho(r,\theta,q;s) = \rho_0/2\pi\, S(r)\, Z(q;s)$, with $Z(q;s) = \int_{\mathbb{R}} \Psi(\vec{z};s)\mathrm{d}p$
- Longitudinal force on a test charge distribution at position q with shape $\hat{S}(r)$ is then given by

$$\boxed{\bar{F}_z[Z,S,\hat{S}](z) \propto \int\limits_{-\infty}^{\infty} \mathrm{d}k\, k\, \mathrm{e}^{\mathrm{i}\,kz}\, \tilde{Z}(k)\, Q[S,\hat{S}](k)}$$

$$\boxed{Q[S,\hat{S}](k) = \iint\limits_{0}^{\infty} \mathrm{d}r\,\mathrm{d}r'\,r\,r'\,S(r')\,\hat{S}(r)\,I_0(k\,r_-)\,K_0(k\,r_+), \quad \text{with} \quad r_{+/-} = \max/\min(r,r')}$$

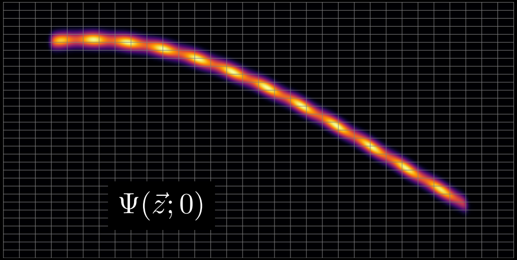
- Currently in use: $S(r) = \begin{cases} 1 & r \leq a \\ 0 & \text{else} \end{cases}$ and $\hat{S}(r) = r^{-1}\delta(r) \implies Q[S,\hat{S}] = \dfrac{1 - a\,k\,K_1(k\,a)}{k^2}$

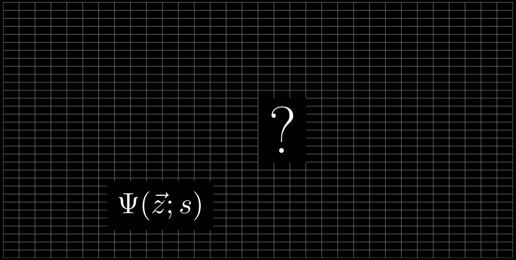
Naive Simulation Procedure – NOT PFTC2d



- Discretize $\Psi(\vec{z};0)$, store values on a grid $\Psi(\vec{z}_{ij};0)$
- To update time-forward grid at $\vec{z}_{ij}$ track back $\vec{z}_{ij}$ and evaluate old PSD $\Psi(\vec{z}_{ij};s) = \Psi(M_{0\leftarrow s}(\vec{z}_{ij});0)$ ($\Longrightarrow$ interpolation necessary)
- Problem for *exotic* PSDs: Most of the grid is empty! Prohibitively wasteful for high grid resolutions! $\Longrightarrow$ We can do better!

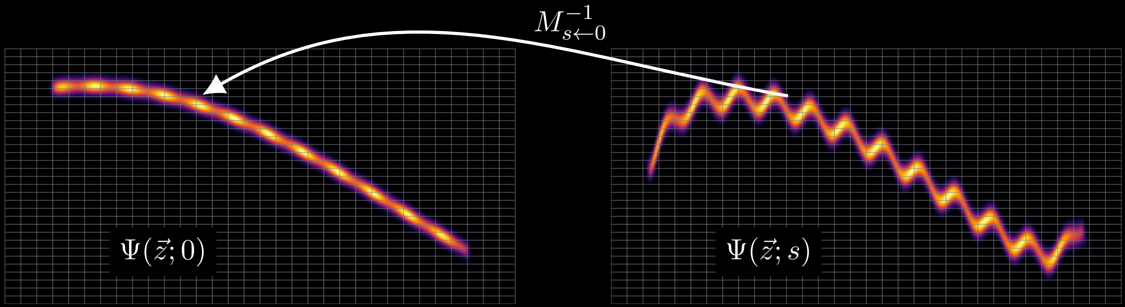
Naive Simulation Procedure – NOT PFTC2d





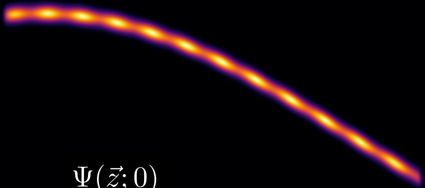
- Discretize $\Psi(\vec{z}; 0)$, store values on a grid $\Psi(\vec{z}_{ij}; 0)$
- To update time-forward grid at $\vec{z}_{ij}$ track back $\vec{z}_{ij}$ and evaluate old PSD $\Psi(\vec{z}_{ij}; s) = \Psi(M_{0\leftarrow s}(\vec{z}_{ij}); 0)$ ($\implies$ interpolation necessary)
- Problem for *exotic* PSDs: Most of the grid is empty! Prohibitively wasteful for high grid resolutions! $\implies$ We can do better!

Naive Simulation Procedure – NOT PFTC2d



- Discretize $\Psi(\vec{z};0)$, store values on a grid $\Psi(\vec{z}_{ij};0)$
- To update time-forward grid at $\vec{z}_{ij}$ track back $\vec{z}_{ij}$ and evaluate old PSD $\Psi(\vec{z}_{ij};s) = \Psi(M_{0\leftarrow s}(\vec{z}_{ij});0)$ ($\implies$ interpolation necessary)
- Problem for *exotic* PSDs: Most of the grid is empty! Prohibitively wasteful for high grid resolutions! $\implies$ We can do better!

Quadtree Domain Decomposition – PFTC2d



$\Psi(\vec{z};0)$

$?$

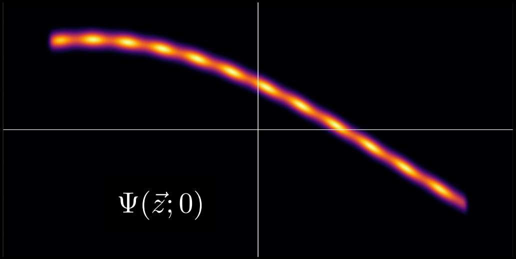
$\Psi(\vec{z};s)$

- Recursively refine the mesh, where $\Psi(\vec{z}) > \epsilon$
- Results in a (Quad)-Tree structure with depth r
- Store function values only on a grid on the smallest leafs $\implies$ memory efficient
- Evaluate by determining the appropriate leaf $\mathcal{O}(r)$

Update Step

1. Track boundary forward to determine new outer box
2. Refine box at one point to desired resolution
3. Starting from there, create & update neighboring boxes (flood fill)

Quadtree Domain Decomposition – PFTC2d



?

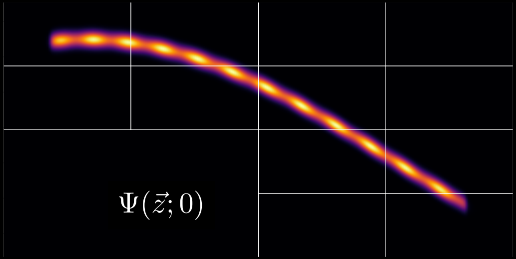
$\Psi(\vec{z};s)$

- Recursively refine the mesh, where $\Psi(\vec{z}) > \epsilon$
- Results in a (Quad)-Tree structure with depth r
- Store function values only on a grid on the smallest leafs $\implies$ memory efficient
- Evaluate by determining the appropriate leaf $\mathcal{O}(r)$

Update Step

1. Track boundary forward to determine new outer box
2. Refine box at one point to desired resolution
3. Starting from there, create & update neighboring boxes (flood fill)

Quadtree Domain Decomposition – PFTC2d



?

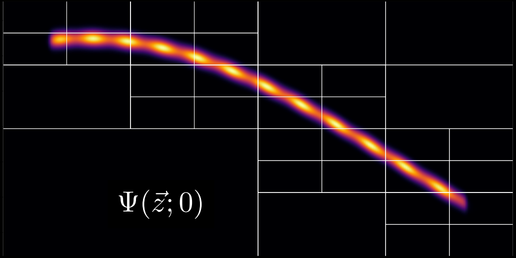
$\Psi(\vec{z}; s)$

- Recursively refine the mesh, where $\Psi(\vec{z}) > \epsilon$
- Results in a (Quad)-Tree structure with depth r
- Store function values only on a grid on the smallest leafs $\implies$ memory efficient
- Evaluate by determining the appropriate leaf $\mathcal{O}(r)$

Update Step

1. Track boundary forward to determine new outer box
2. Refine box at one point to desired resolution
3. Starting from there, create & update neighboring boxes (flood fill)

Quadtree Domain Decomposition – PFTC2d



?

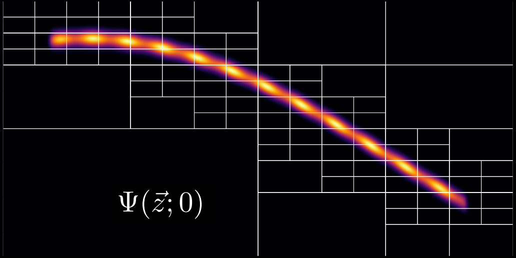
$\Psi(\vec{z};s)$

- Recursively refine the mesh, where $\Psi(\vec{z}) > \epsilon$
- Results in a (Quad)-Tree structure with depth r
- Store function values only on a grid on the smallest leafs $\Longrightarrow$ memory efficient
- Evaluate by determining the appropriate leaf $\mathcal{O}(r)$

Update Step

1. Track boundary forward to determine new outer box
2. Refine box at one point to desired resolution
3. Starting from there, create & update neighboring boxes (flood fill)

Quadtree Domain Decomposition – PFTC2d



?

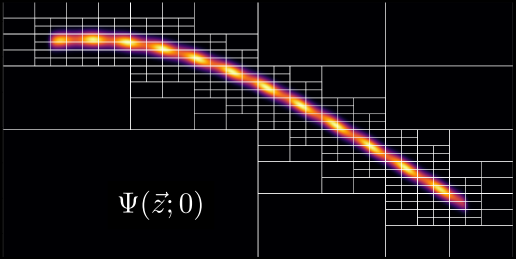
$\Psi(\vec{z};s)$

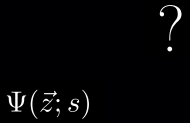
- Recursively refine the mesh, where $\Psi(\vec{z}) > \epsilon$
- Results in a (Quad)-Tree structure with depth r
- Store function values only on a grid on the smallest leafs $\implies$ memory efficient
- Evaluate by determining the appropriate leaf $\mathcal{O}(r)$

Update Step

1. Track boundary forward to determine new outer box
2. Refine box at one point to desired resolution
3. Starting from there, create & update neighboring boxes (flood fill)

Quadtree Domain Decomposition – PFTC2d



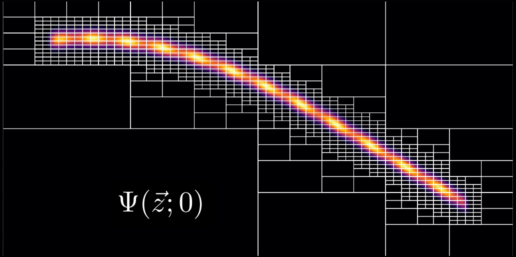


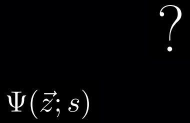
- Recursively refine the mesh, where $\Psi(\vec{z}) > \epsilon$
- Results in a (Quad)-Tree structure with depth r
- Store function values only on a grid on the smallest leafs $\implies$ memory efficient
- Evaluate by determining the appropriate leaf $\mathcal{O}(r)$

Update Step

1. Track boundary forward to determine new outer box
2. Refine box at one point to desired resolution
3. Starting from there, create & update neighboring boxes (flood fill)

Quadtree Domain Decomposition – PFTC2d



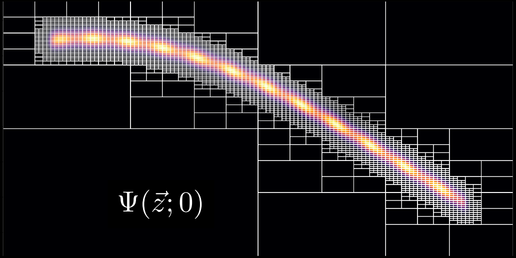


- Recursively refine the mesh, where $\Psi(\vec{z}) > \epsilon$
- Results in a (Quad)-Tree structure with depth r
- Store function values only on a grid on the smallest leafs $\implies$ memory efficient
- Evaluate by determining the appropriate leaf $\mathcal{O}(r)$

Update Step

1. Track boundary forward to determine new outer box
2. Refine box at one point to desired resolution
3. Starting from there, create & update neighboring boxes (flood fill)

Quadtree Domain Decomposition – PFTC2d



$?$

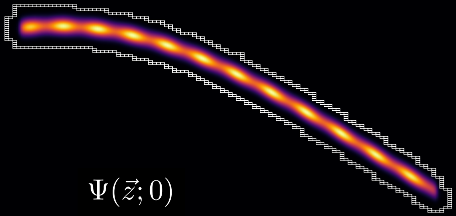
$\Psi(\vec{z}; s)$

- Recursively refine the mesh, where $\Psi(\vec{z}) > \epsilon$
- Results in a (Quad)-Tree structure with depth r
- Store function values only on a grid on the smallest leafs $\implies$ memory efficient
- Evaluate by determining the appropriate leaf $\mathcal{O}(r)$

Update Step

1. Track boundary forward to determine new outer box
2. Refine box at one point to desired resolution
3. Starting from there, create & update neighboring boxes (flood fill)

Quadtree Domain Decomposition – PFTC2d



$\Psi(\vec{z}; 0)$

?

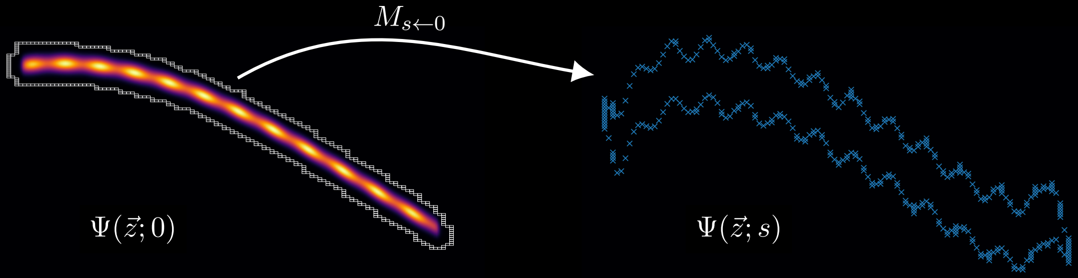
$\Psi(\vec{z}; s)$

- Recursively refine the mesh, where $\Psi(\vec{z}) > \epsilon$
- Results in a (Quad)-Tree structure with depth r
- Store function values only on a grid on the smallest leafs $\implies$ memory efficient
- Evaluate by determining the appropriate leaf $\mathcal{O}(r)$

Update Step

1. Track boundary forward to determine new outer box
2. Refine box at one point to desired resolution
3. Starting from there, create & update neighboring boxes (flood fill)

Quadtree Domain Decomposition – PFTC2d

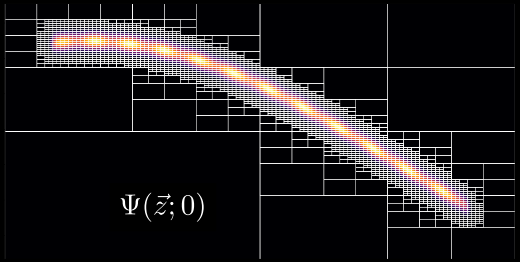


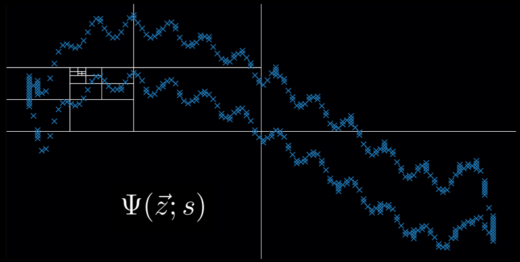
- Recursively refine the mesh, where $\Psi(\vec{z}) > \epsilon$
- Results in a (Quad)-Tree structure with depth r
- Store function values only on a grid on the smallest leafs $\implies$ memory efficient
- Evaluate by determining the appropriate leaf $\mathcal{O}(r)$

Update Step

1. Track boundary forward to determine new outer box
2. Refine box at one point to desired resolution
3. Starting from there, create & update neighboring boxes (flood fill)

Quadtree Domain Decomposition – PFTC2d



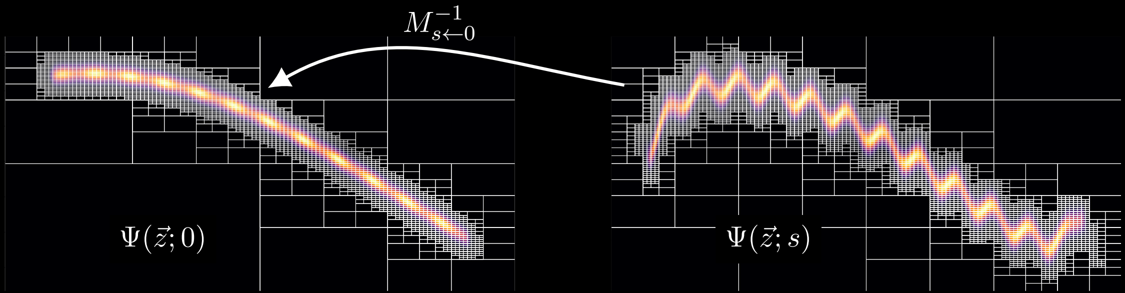


- Recursively refine the mesh, where $\Psi(\vec{z}) > \epsilon$
- Results in a (Quad)-Tree structure with depth r
- Store function values only on a grid on the smallest leafs $\implies$ memory efficient
- Evaluate by determining the appropriate leaf $\mathcal{O}(r)$

Update Step

1. Track boundary forward to determine new outer box
2. Refine box at one point to desired resolution
3. Starting from there, create & update neighboring boxes (flood fill)

Quadtree Domain Decomposition – PFTC2d

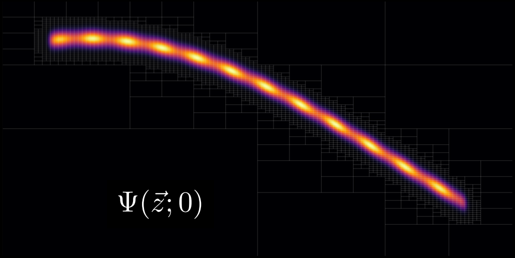


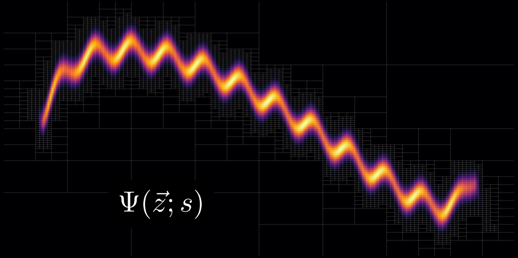
- Recursively refine the mesh, where $\Psi(\vec{z}) > \epsilon$
- Results in a (Quad)-Tree structure with depth r
- Store function values only on a grid on the smallest leafs $\implies$ memory efficient
- Evaluate by determining the appropriate leaf $\mathcal{O}(r)$

Update Step

1. Track boundary forward to determine new outer box
2. Refine box at one point to desired resolution
3. Starting from there, create & update neighboring boxes (flood fill)

Quadtree Domain Decomposition – PFTC2d





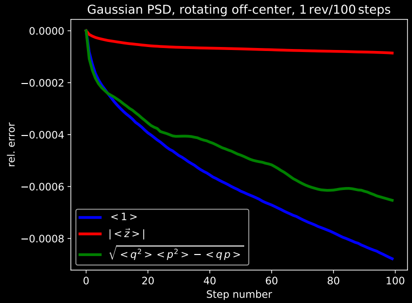
- Recursively refine the mesh, where $\Psi(\vec{z}) > \epsilon$
- Results in a (Quad)-Tree structure with depth r
- Store function values only on a grid on the smallest leafs $\implies$ memory efficient
- Evaluate by determining the appropriate leaf $\mathcal{O}(r)$

Update Step

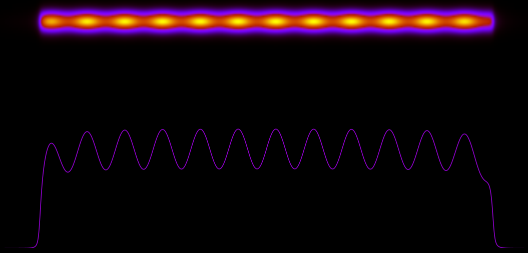
1. Track boundary forward to determine new outer box
2. Refine box at one point to desired resolution
3. Starting from there, create & update neighboring boxes (flood fill)

PFTC2d

- `PFTC2d` is implemented in `C++`
- Trees allow for efficient implementation of important operations:
  - Projections $\int_{\mathbb{R}} \Psi(\vec{z}; s)\mathrm{d}p$
  - Expectation values $E[f](s) = \int_{\mathbb{R}^2} f(\vec{z})\, \Psi(\vec{z}; s)\mathrm{d}^2\vec{z}$
- Underlying principles and data structures are N-Dim-ready: Octrees, Sedecatrees,...



PFTC2d



Summary

- Multi-stage bunch compression leads to interesting non-linear effects
- Because of exotic FEL PSDs, the efficient simulation is not straight-forward
- `PFTC2d` is under development: A Perron-Frobenius code, utilizing the quadtree domain decomposition technique

Thank you
for your attention!