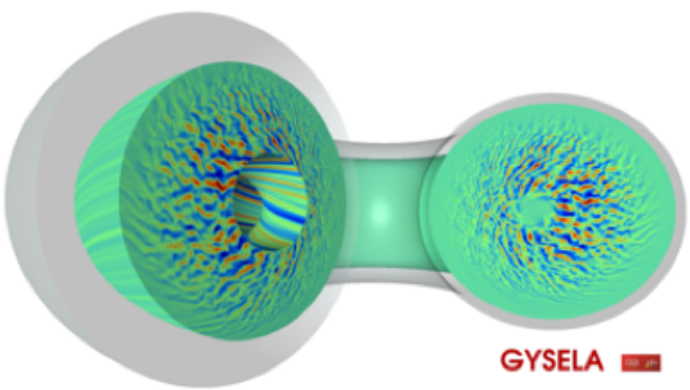
jh200053-MDHI

Supercomputer resources:
FX1000 @ Nagoya Univ, Tsubame3.0 @Tokyo Tech

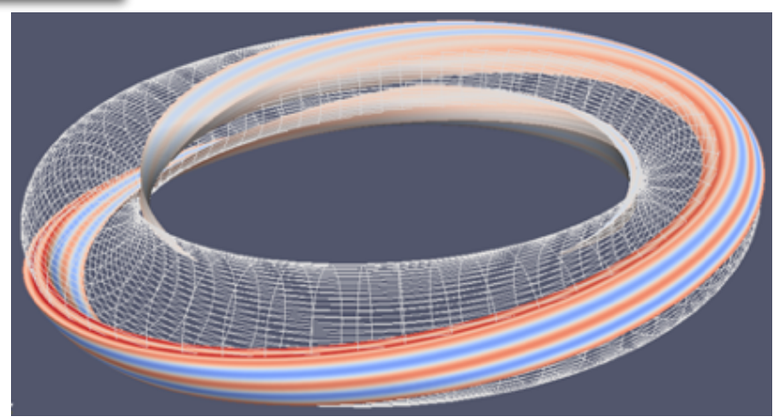
Preparing for Exa-systems: Performance portable implementation and scalable data analysis

Representative:	**Y. Asahi (JAEA)**	Code development
Deputy Representative:	S. Maeyama (Nagoya Univ.)	Plasma turbulence
Deputy Representative:	J. Bigot (MdS, France)	Scalable data analysis
Collaborating researcher:	X. Garbet (CEA, France)	Global plasma turbulence
Collaborating researcher:	G. Latu (CEA, France)	Performance portability
Collaborating researcher:	K. Fuji (Kyoto Univ.)	Machine learning
Collaborating researcher:	O. Kevin (CEA, France)	Global plasma turbulence
Collaborating researcher:	T.-H. Watanabe (Nagoya Univ.)	Local plasma turbulence
Collaborating researcher:	Y. Idomura (JAEA)	Large scale simulation
Collaborating researcher:	T. Aoki (Tokyo Tech.)	Optimization on GPU
Collaborating researcher:	T. Katagiri (Nagoya Univ.)	Optimization on CPU

Global code: GYSELA

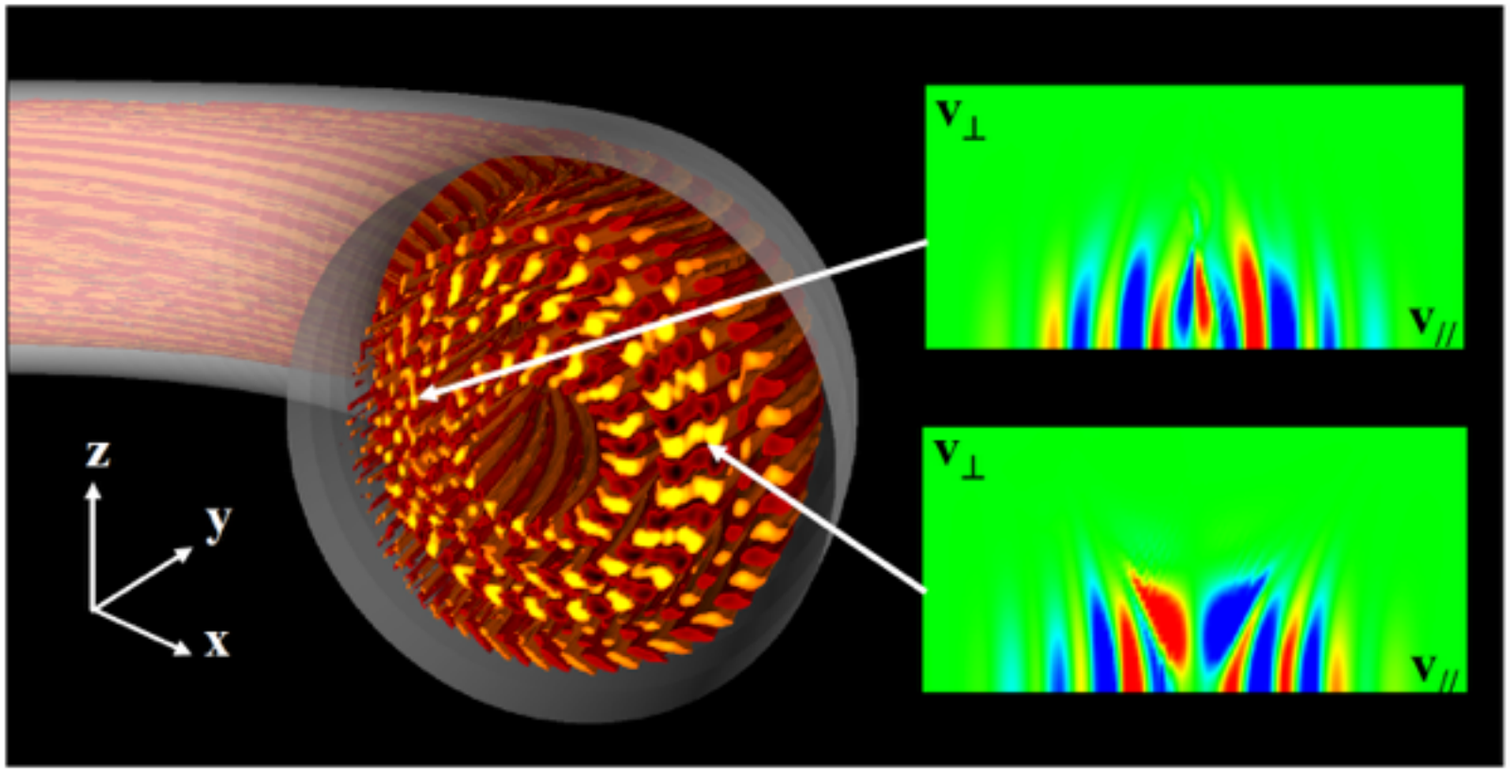


Local code: GKV



JHPCN 12th symposium, Shinagawa, Japan **Date: 9/July/2020**

Plasma turbulence simulation



Each grid point has structure in real space (x, y, z) and velocity space (v∥, v⊥)

→ **5D** stencil computations

[Idomura et al., Comput. Phys. Commun (2008); Nuclear Fusion (2009)]

First principle gyrokinetic model to predict plasma turbulence

- Confinement properties of fusion reactors (high temperature, non-Maxwellian)

Solving the machine scale problem (~m) with turbulence scale mesh (~cm)

- Degrees of freedom: $100^5 \sim 10^{10}$ Large scale data analysis (1D to **5D**)

Concerning the dynamics of kinetic electrons, complicated geometry, even more computational resource is needed

- **Accelerators** are key ingredients to satisfy huge computational demands at **reasonable energy consumption:** MPI + **'X'**

Objectives

Performance portability

- Investigating performance of directive based approach and abstraction based approach [1]★
- Scaling portable implementation with Kokkos and OpenMP4.5 (overlapping communication and computation using std::thread or OpenMP task)

Scalable data analysis

- Large scale data analysis based on Dask
- Analyzing the time series of 5D distribution function [2]
- In-situ machine learning to avoid saving the huge data

Exascale simulation and data analysis

[1] Y. Asahi et al., WACCPD 6, SC19
[2] Y. Asahi et al., to be submitted

★ Completed in JH190065 “Modernizing and accelerating fusion plasma turbulence codes targeting exa-scale systems”

Objectives

Performance portability

- Investigating performance of directive based approach and abstraction based approach [1]★
- Scaling portable implementation with Kokkos and OpenMP4.5 (overlapping communication and computation using std::thread or OpenMP task)

Scalable data analysis

- Large scale data analysis based on Dask
- Analyzing the time series of 5D distribution function [2]
- In-situ machine learning to avoid saving the huge data

Exascale simulation and data analysis

[1] Y. Asahi et al., WACCPD 6, SC19
[2] Y. Asahi et al., to be submitted

★ Completed in JH190065 “Modernizing and accelerating fusion plasma turbulence codes targeting exa-scale systems”

Masking transpose communication costs (jh180081-NAHI)

GKV and GYSELA employ transpose communications and 2D Operations

Sequential

Pack

1. Packing $\delta f_{k_x,k_y}$ to send buffer

Forward

2. Forward Transpose for $\delta f_{k_x,k_y}$

Conv2D

3. 2D IFFT in x, y for $ik_x\delta f_{k_x,k_y}, ik_y\delta f_{k_x,k_y}$
4. Multiplication $\mathcal{N}_{x,y} = \nabla_\perp \delta f_{x,y} \times \nabla_\perp \delta\psi_{x,y}$
5. 2D FFT in x, y for $\mathcal{N}_{x,y}$

Backward

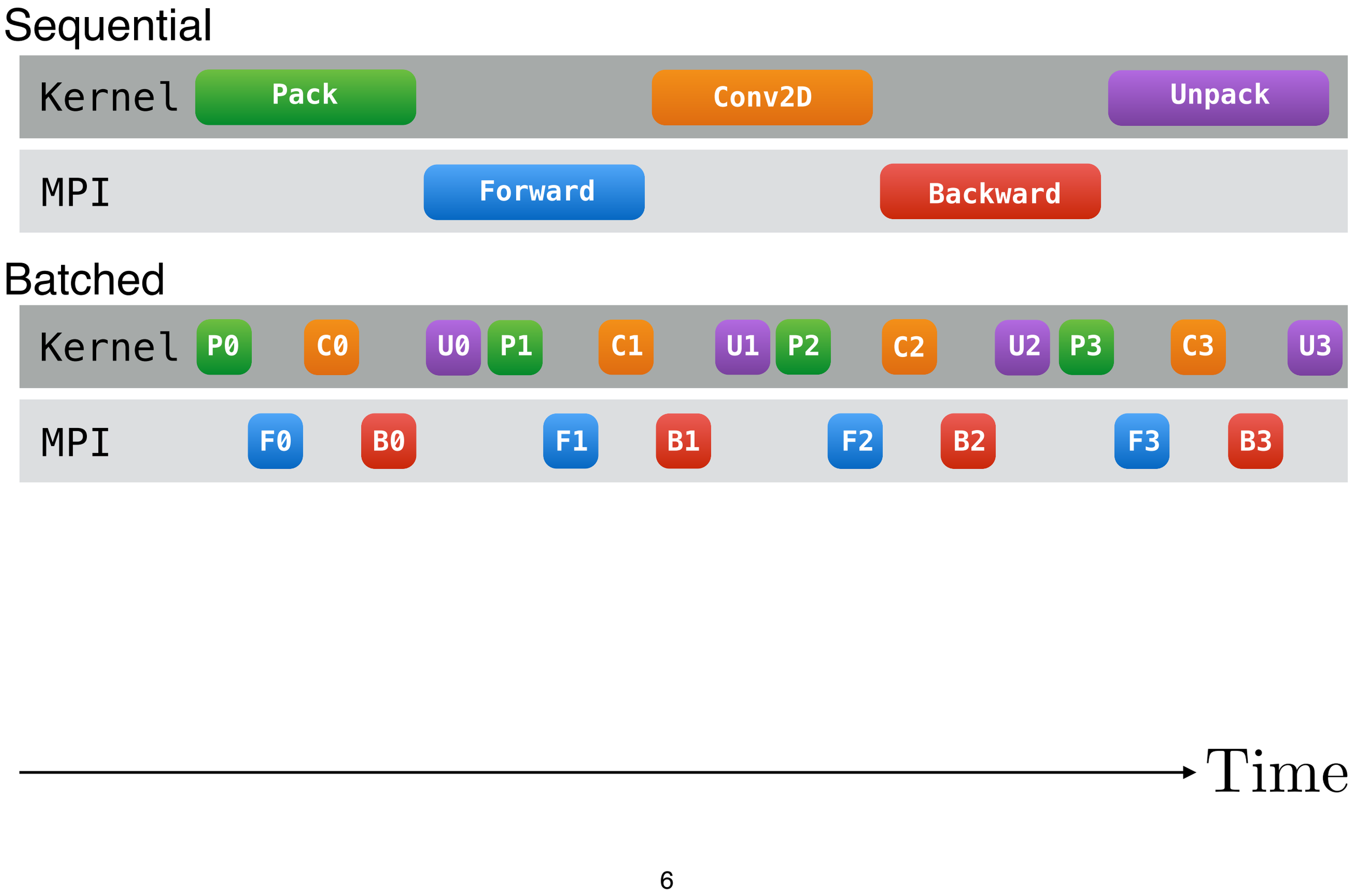
6. Backward Transpose for $\mathcal{N}_{k_x,k_y}$

Unpack

7. Unpacking $\mathcal{N}_{k_x,k_y}$ from receive buffer

Time

Masking transpose communication costs (jh180081-NAHI)

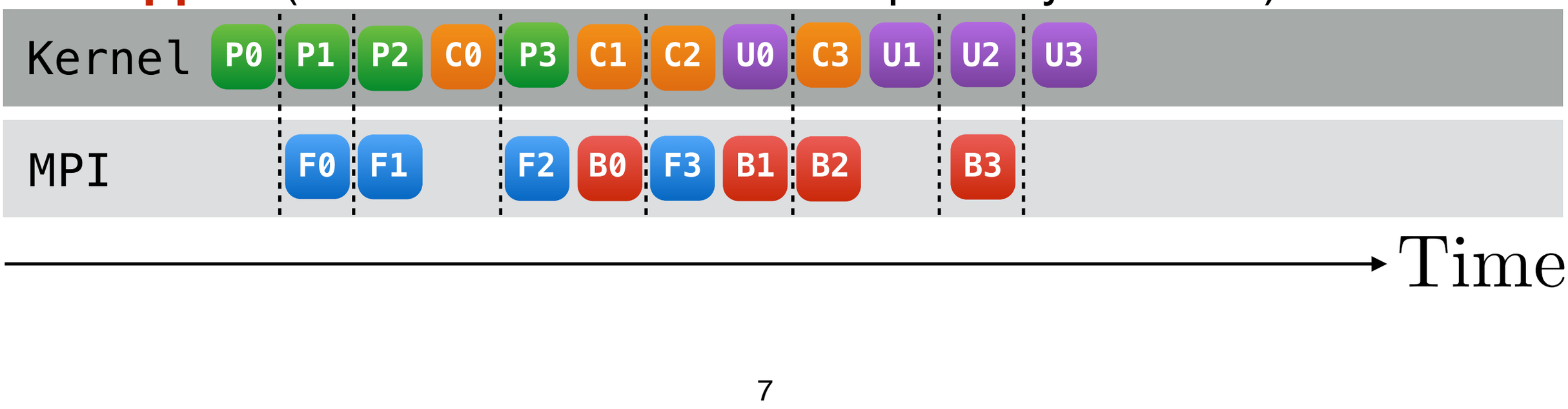


Masking transpose communication costs (jh180081-NAHI)

Sequential

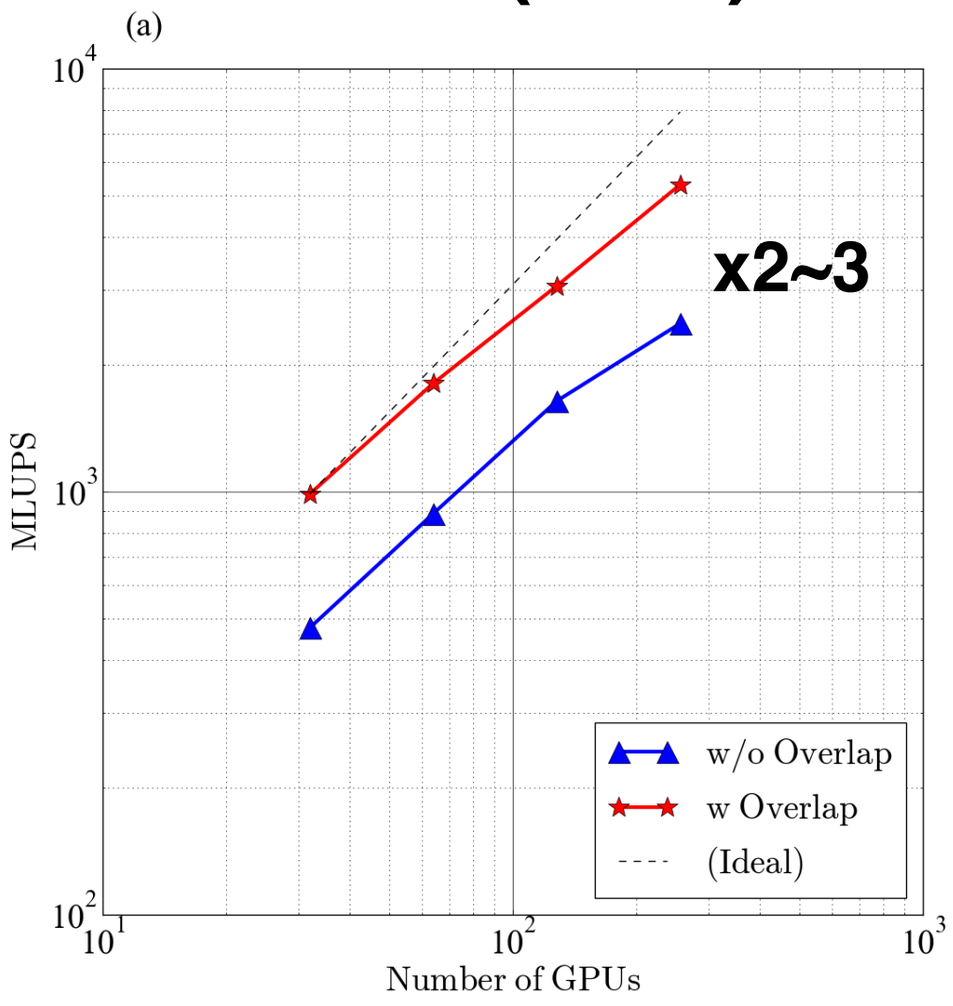
Batched

Overlapped (communication costs are partially masked)

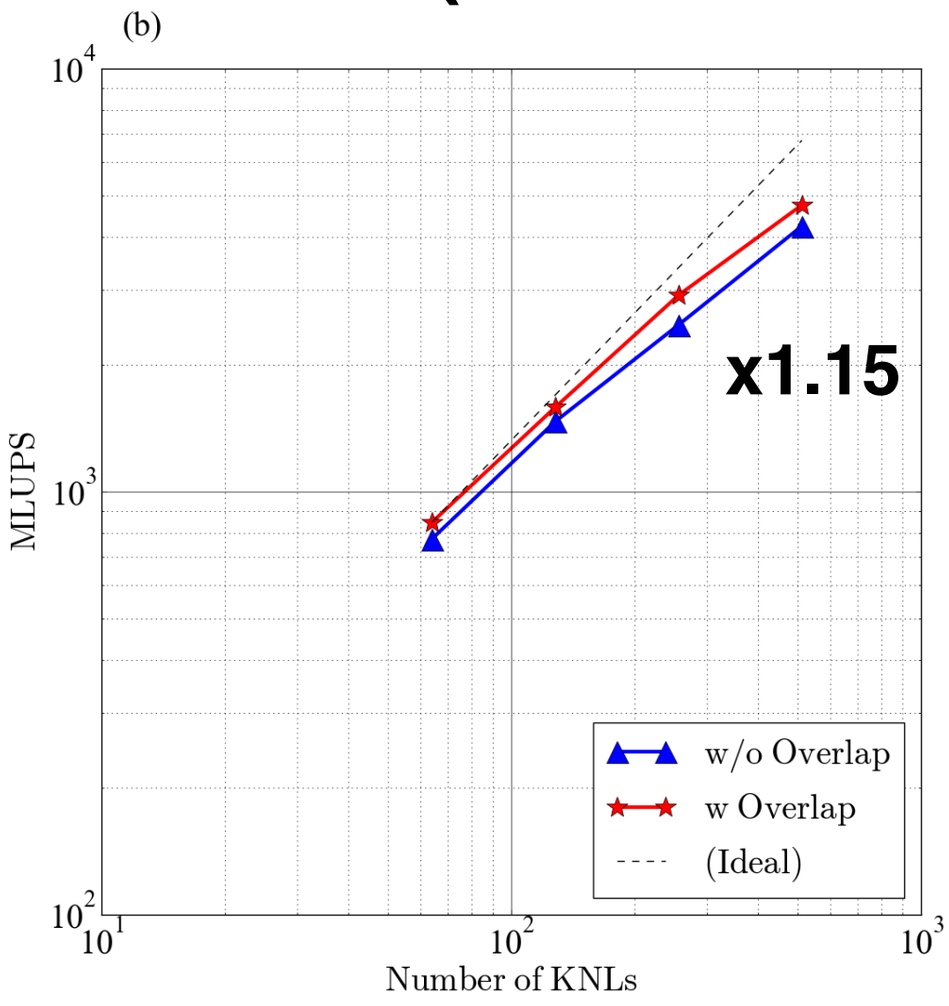


Strong scaling with overlapping (jh180081-NAHI)

GKV (GPU)



GYSELA (Xeon Phi KNL)



- Applying kernel optimization techniques [1] to GYSELA and GKV [2]
- Applying communication and computation overlapping [3]

Achievements

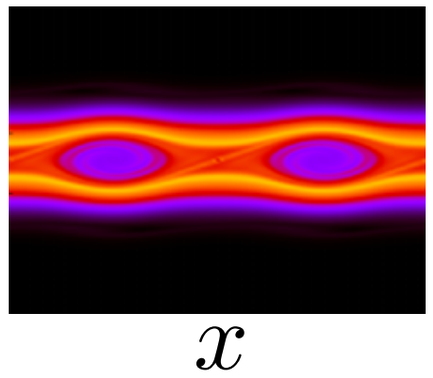
2x Speed up compared to the conventional CPUs (Broadwell~0.5TFlops, FX100~1TFlops)

[1] Y. Asahi et al., IEEE-TPDS, **28**, 7, 1974–1988 (2017)
[2] T.-H. Watanabe et al., Nucl. Fusion **46**, 24-32, (2006)
[3] Y. Asahi et al., CCPE (2020)

Performance portable implementation with Kokkos (jh190065)

4D Vlasov-Poisson equation （2D space、2D velocity space）

- Vlasov solver: Semi-Lagrangian, Strang splitting
- Poisson solver: 2D Fourier transform



Kokkos implementation of Poisson solver (a single codebase working on CPU/GPU)

```
53   // Forward 2D FFT (Real to Complex)
54   fft_->fft2(rho_.data(), rho_hat_.data());
56   // Solve Poisson equation in Fourier space
57   complex_view_2d ex_hat  = ex_hat_;
58   complex_view_2d ey_hat  = ey_hat_;
59   complex_view_2d rho_hat = rho_hat_;
63   view_1d filter = filter_;
65   Kokkos::parallel_for(nx1h, KOKKOS_LAMBDA (const int ix1) {
66     double kx = ix1 * kx0;
67     {
68       int ix2 = 0;
69       double kx = ix1 * kx0;
70       ex_hat(ix1, ix2) = -kx * I * rho_hat(ix1, ix2) * filter(ix1) * normcoeff;
71       ey_hat(ix1, ix2) = 0.;
72       rho_hat(ix1, ix2) = rho_hat(ix1, ix2) * filter(ix1) * normcoeff;
73     }
74
75     for(int ix2=1; ix2<nx2h; ix2++) {
76       double ky = ix2 * ky0;
77       double k2 = kx * kx + ky * ky;
78
79       ex_hat(ix1, ix2) = -(kx/k2) * I * rho_hat(ix1, ix2) * normcoeff;
80       ey_hat(ix1, ix2) = -(ky/k2) * I * rho_hat(ix1, ix2) * normcoeff;
81       rho_hat(ix1, ix2) = rho_hat(ix1, ix2) / k2 * normcoeff;
82     }
83
84     for(int ix2=nx2h; ix2<nx2; ix2++) {
85       double ky = (ix2-nx2) * ky0;
86       double k2 = kx*kx + ky*ky;
87
88       ex_hat(ix1, ix2) = -(kx/k2) * I * rho_hat(ix1, ix2) * normcoeff;
89       ey_hat(ix1, ix2) = -(ky/k2) * I * rho_hat(ix1, ix2) * normcoeff;
90       rho_hat(ix1, ix2) = rho_hat(ix1, ix2) / k2 * normcoeff;
91     }
92   });
94   // Backward 2D FFT (Complex to Real)
95   fft_->ifft2(rho_hat.data(), rho_.data());
96   fft_->ifft2(ex_hat.data(),  ex_.data());
97   fft_->ifft2(ey_hat.data(),  ey_.data());
```

Performance against SKL (OpenMP)

	Time [s]	Speedup
Skylake (OpenMP)	278	1.0
Skylake (Kokkos)	192	1.45
Arm (OpenMP)	589	0.47
Arm (Kokkos)	335	0.83
P100 (OpenACC)	21.5	12.95
P100 (Kokkos)	15.6	17.83
V100 (OpenACC)	10.0	27.8
V100 (Kokkos)	6.79	40.9

Achievements

Good performance portability keeping readability and productivity with Kokkos (Abstraction of memory and parallel operation)

[1] Y. Asahi et al., OpenACC meeting, September, Japan
[2] Y. Asahi et al., waccpd (SC19), November, US

Objectives

Performance portability

- Investigating performance of directive based approach and abstraction based approach [1]★
- Scaling portable implementation with Kokkos and OpenMP4.5 (overlapping communication and computation using std::thread or OpenMP task)

Scalable data analysis

- Large scale data analysis based on Dask
- Analyzing the time series of 5D distribution function [2]
- In-situ machine learning to avoid saving the huge data

Exascale simulation and data analysis

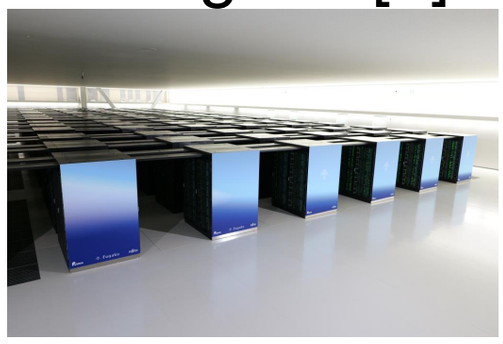
[1] Y. Asahi et al., WACCPD 6, SC19
[2] Y. Asahi et al., to be submitted

★ Completed in JH190065 “Modernizing and accelerating fusion plasma turbulence codes targeting exa-scale systems”

Demands for MPI + ‘X’ in our group

Portability

ARM machine
Fugaku [1]



GPU machine
SUMMIT [2]

Exa machine may be very **divergent**

Readability

OpenMP

```
#pragma omp parallel for
for(int i=0; i<n; i++)
  a[i] = b[i] + scalar * c[i];
```

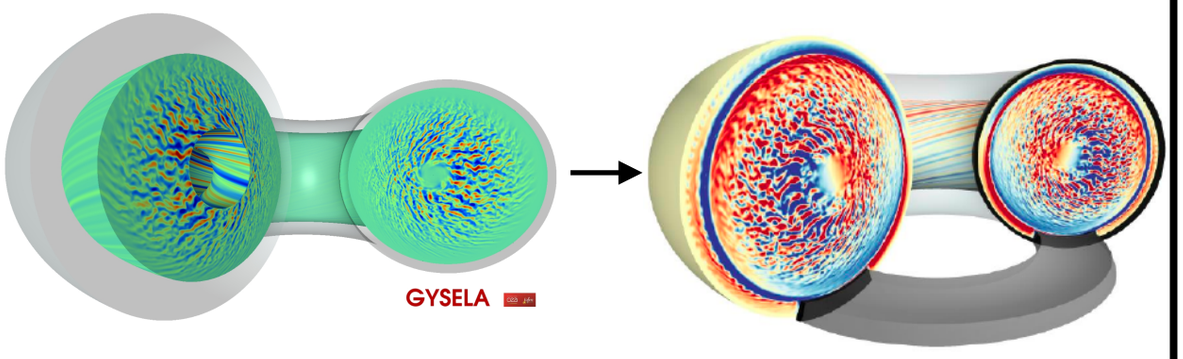
OpenACC

```
#pragma acc parallel loop
for(int i=0; i<n; i++)
  a[i] = b[i] + scalar * c[i];
```

Readable for physicists

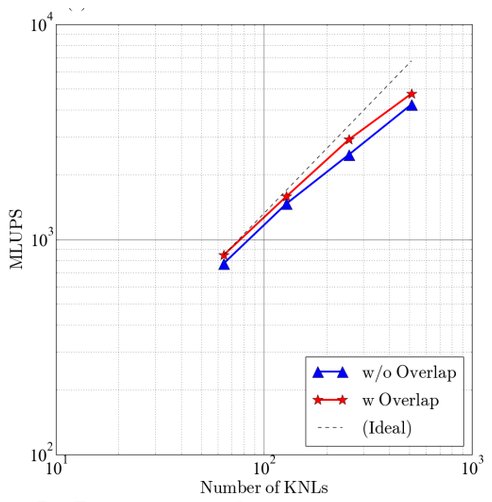
Productivity

Circular → **Limiter**



Advanced (realistic) physical model

High Performance



Strong scaling
of GYSELA
up to 512 KNLs
(MPI+OpenMP)

More than 100 M cpu hours/year

We need a **readable, portable,** and **high performance code** that is **easy to upgrade**!

[1] https://www.r-ccs.riken.jp/en/
[2] https://www.olcf.ornl.gov/summit/

Aim: explore performance portable implementation with the mini-app

Requirements

- Productivity: Easy to modify and maintenance
- Readability: Easy to read for developers from many different fields
- Portability: A single code runs on **many different devices**
- High performance: Good performance on a given device

Possible approaches

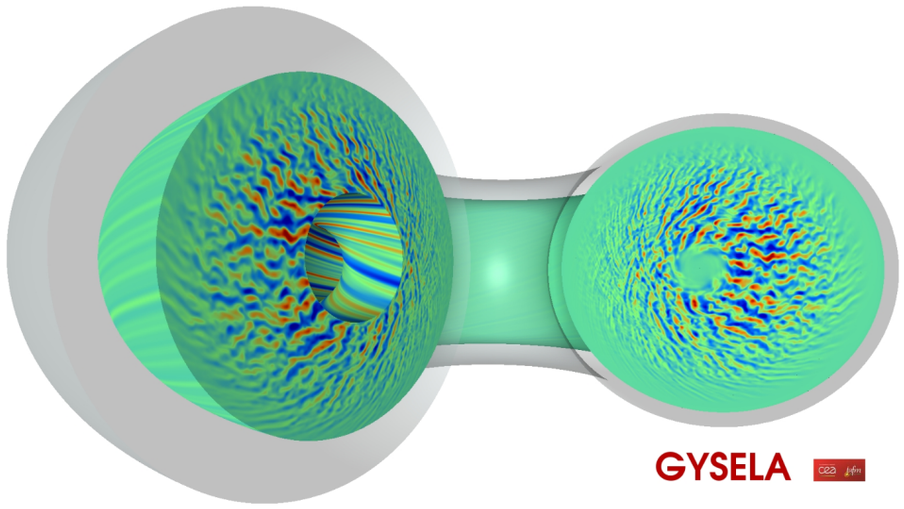
- Directive based approach: OpenMP, **OpenACC, OpenMP4.5**
- Higher level abstraction: **Kokkos**, RAJA, Alpaka

Methodology

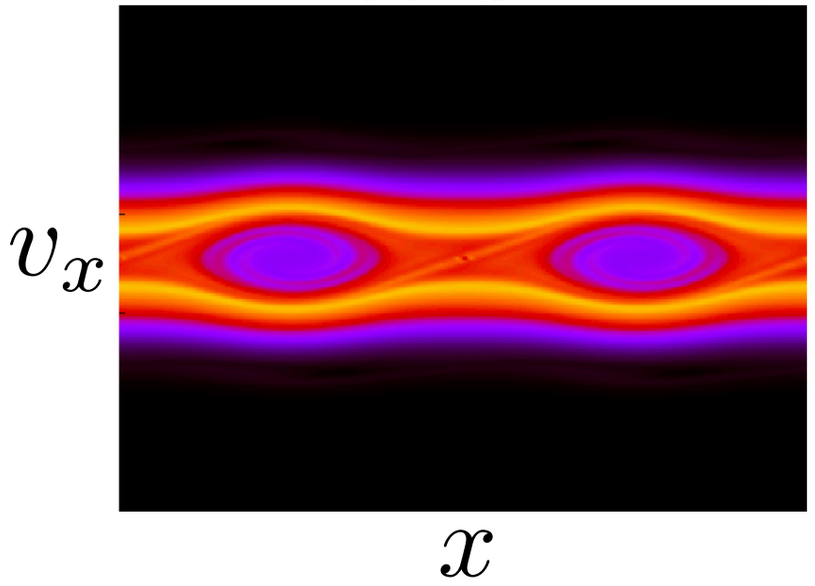
- Directive-based and abstraction-based implementation of mini-app
  Mixed OpenMP/OpenACC and Kokkos (minimize code duplication)
- Explore performance portable implementation over different devices:
  Nvidia GPUs, **Intel CPU**, **ARM CPU**
- Target mini-apps: GKV mini-app (transpose and FFT) and GYSELA mini-app (semi-Lagrangian)

Encapsulate key GYSELA features into mini-app

GYSELA (3D torus) $(r, \theta, \phi, v_{\|}, \mu)$



Mini-app (periodic) (x, y, v_x, v_y)



	GYSELA	Mini-app ★	Mini-app MPI
System	5D Vlasov + 3D Poisson	4D Vlasov + 2D Poisson	4D Vlasov + 2D Poisson
Geometry	Realistic tokamak geoemtry	Periodic boundary conditions	Periodic boundary conditions
Scheme	Semi-Lagrangian (Spline) + Operator splitting	Semi-Lagrangian (Lagrange) + Operator splitting	Semi-Lagrangian (Spline) + Operator splitting
MPI	Yes	**No**	**Yes**
X	OpenMP	**OpenACC/OpenMP/Kokkos**	**Kokkos**
Language	Fortran 90	**C++**	**C++**
Lines of	More than 50k	About 5k	About 8k

- Extract the **Semi-Lagrangian + operator splitting** strategy for Vlasov solver
- Choose **Kokkos** for MPI version based on the experience with Mini-app [1]

★JH190065
[1] https://github.com/yasahi-hpc/vlp4d

Global algo. of GYSELA mini app (One time step)

```
18   // Exchange halo of the local domain in order to perform
19   // the advection afterwards (the interpolation needs the
20   // extra points located in the halo region)
21   comm.exchangeHalo(conf, fn, timers);
22
24   Spline::computeCoeff_xy(conf, fn);
25   Impl::deep_copy(fnp1, fn);
28
30   Advection::advect_2D_xy(conf, fn, 0.5 * dom->dt_);
33
35   field_rho(conf, comm, fn, ef);
36   field_poisson(conf, ef, dg, iter);
```

← **MPI Comm** Boundary condition

← spline coef (xy)

← copy fn to fnp1

← 2D advection (xy) fn updated

← compute phi using fn

```
39
41   Spline::computeCoeff_vxvy(conf, fnp1);
44
46   Advection::advect_4D(conf, ef, fnp1, fn, dom->dt_);
49
51   field_rho(conf, comm, fnp1, ef);
52   field_poisson(conf, ef, dg, iter);
```

← spline coef (xy vxvy)

← 4D advection (xy vxvy) fnp1 updated

← compute phi using fnp1

- GYSELA mini-app is parallelized with MPI (P2P and all reduce comms)
- Local spline is used for interpolation
- All reduce communication in poisson solver can be **masked by overlapping**

2D convolution by FFT (GKV hotspot)

Algorithm 2 2D convolution kernel

for *All grid points* $(\mu_m, v_{\|l}, z_k)$ **do**

$$\mathcal{N}^{n+1}_{k,l,m} = \sum_{k_\perp = k'_\perp + k''_\perp} \mathbf{b} \cdot k'_\perp \times k''_\perp \delta\psi_{k'_\perp,k,m} \delta f_{k''_\perp,k,l,m}$$
$$= \mathrm{FFT}[\ \mathrm{IFFT}[ik_x \delta f_{k_\perp}]\mathrm{IFFT}[ik_y \delta\psi_{k_\perp}] - \mathrm{IFFT}[ik_y \delta f_{k_\perp}]\mathrm{IFFT}[ik_x \delta\psi_{k_\perp}]\ ]$$
$$= \mathrm{FFT}[\nabla_\perp \delta f_{i,j,k,l,m} \times \nabla_\perp \delta\psi_{i,j,k,m}]$$

end for

GPU (transpose in (ky, z))

Data structure after operation

0. $\delta f_{k_x,k_y}$ $\quad (N_{k_x}, N_{k_y}/p_y, N_z)$

1. Forward Transpose for $\delta f_{k_x,k_y}$ $\quad (N_{k_x}, N_{k_y}, N_z/p_y)$

2. 2D IFFT in x, y for $ik_x \delta f_{k_x,k_y}, ik_y \delta f_{k_x,k_y}$ $\quad (N_x, N_y, N_z/p_y)$

3. Multiplication $\mathcal{N}_{x,y} = \nabla_\perp \delta f_{x,y} \times \nabla_\perp \delta\psi_{x,y}$ $\quad (N_x, N_y, N_z/p_y)$

4. 2D FFT in x, y for $\mathcal{N}_{x,y}$ $\quad (N_{k_x}, N_{k_y}, N_z/p_y)$

5. Backward Transpose for $\mathcal{N}_{k_x,k_y}$ $\quad (N_{k_x}, N_{k_y}/p_y, N_z)$

- Hot spot of GKV code consists of transpose communication and 2D FFT (convolution)
- GKV mini app (2D derivative with FFT): Transpose + 2D FFT

[Y. Asahi et al., CCPE (2020)]

GKV mini-app in Kokkos

GKV mini-app: 2D derivative with FFT and transpose（pipelined）

$$(N_x, N_y/p_y, N_z/N_{\mathrm{batch}}) \longrightarrow (N_x, N_y, N_z/N_{\mathrm{batch}}/p_y) \longrightarrow (N_x, N_y/p_y, N_z/N_{\mathrm{batch}})$$

Compute derivative in Fourier space

```
 99        for(int ib = 0; ib < Nbatch; ++ib) {
100          // Forward transpose (Nx, Ny/py, Nz/batch) => (Nx, Ny, Nz/batch/py)
101          trans.forward(in, xy, ib);
102
103          // Forward transform (Nx, Ny, Nz/batch/py) => (Nx/2+1, Ny, Nz/batch/py)
104          fft.fft2(xy.data(), kxky.data());
105
106          // Derivative in Fourier space
107          float64 normcoeff = 1./float64(Nx*Ny);
108          Kokkos::parallel_for("derivative", derivative_policy2d,
                                  KOKKOS_LAMBDA (const int jx, const int jy) {
109            complex128 ikx_tmp = ikx(jx, jy);
110            complex128 iky_tmp = iky(jx, jy);
111            for(int jz=0; jz<Nz_local; jz++) {
112              kxky(jx, jy, jz) = (ikx_tmp * kxky(jx, jy, jz) + iky_tmp * kxky(jx, jy, jz))
                                  * normcoeff;
113            }
114          });
115
116          // Backward transform (Nx/2+1, Ny, Nz/batch/py) => (Nx, Ny, Nz/batch/py)
117          fft.ifft2(kxky.data(), xy.data());
118
119          // Backward transpose (Nx, Ny, Nz/batch/py) => (Nx, Ny/py, Nz/batch)
120          trans.backward(xy, out, ib);
121        }
```

std::thread with Kokkos for Overlap

```
        for(int ib = 0; ib < Nbatch + 4; ++ib) {
101       // Shallow copy
102       RealView4D send_forward0  = (ib % 2 == 0) ? trans.send_forward0_  : trans.send_forward1_;
103       RealView4D recv_forward0  = (ib % 2 == 0) ? trans.recv_forward0_  : trans.recv_forward1_;
          …
110
111       threads.emplace_back(
112         [&]() {
113           // Packing
114           if(ib < Nbatch) {
115             trans.forwardPack(in, send_forward0, ib);
116           }
118           // Unpacking, 2D operation, Packing
119           if(ib >= 2 && ib < Nbatch + 2) {
120             trans.forwardUnpack(recv_forward1, xy);
122             fft.fft2(xy.data(), kxky.data());
123             float64 normcoeff = 1./float64(Nx*Ny);
124             Kokkos::parallel_for("derivative", derivative_policy2d, KOKKOS_LAMBDA (const int jx, const int jy) {
125               complex128 ikx_tmp = ikx(jx, jy);
126               complex128 iky_tmp = iky(jx, jy);
127               for(int jz=0; jz<Nz_local; jz++) {
128                 kxky(jx, jy, jz) = (ikx_tmp * kxky(jx, jy, jz) + iky_tmp * kxky(jx, jy, jz)) * normcoeff;
129               }
130             });
131             fft.ifft2(kxky.data(), xy.data());
132
133             trans.backwardPack(xy, send_backward0);
134           }
136           // Unpacking (backward transpose)
137           if(ib >= 4 && ib < Nbatch + 4) {
138             trans.backwardUnpack(recv_backward1, out, ib);
139           }
140         }
141       );
143       // Communications
144       if(ib >= 1 && ib < Nbatch + 1) {
145         trans.comm(send_forward1, recv_forward0);
146       }
147       if(ib >= 3 && ib < Nbatch + 3) {
148         trans.comm(send_backward1, recv_backward0);
149       }
151       for(auto &th: threads) th.join();
152       std::vector<std::thread>().swap(threads); // cleanup
153     }
```

Kokkos kernel launched by std::thread

Communication/Computation overlap by pipelining [1]

[1] Y. Asahi, CCPE (2020)

Objectives

Performance portability

- Investigating performance of directive based approach and abstraction based approach [1]★
- Scaling portable implementation with Kokkos and OpenMP4.5 (overlapping communication and computation using std::thread or OpenMP task)

Scalable data analysis

- Large scale data analysis based on Dask
- Analyzing the time series of 5D distribution function [2]
- In-situ machine learning to avoid saving the huge data

Exascale simulation and data analysis

[1] Y. Asahi et al., WACCPD 6, SC19
[2] Y. Asahi et al., to be submitted

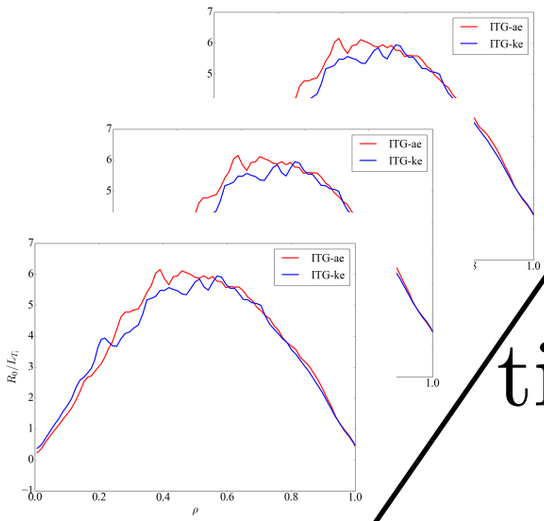
★ Completed in JH190065 “Modernizing and accelerating fusion plasma turbulence codes targeting exa-scale systems”

Analyzing 5D gyrokinetic simulation data

Conventional study

1D time series

Structures of radial profile



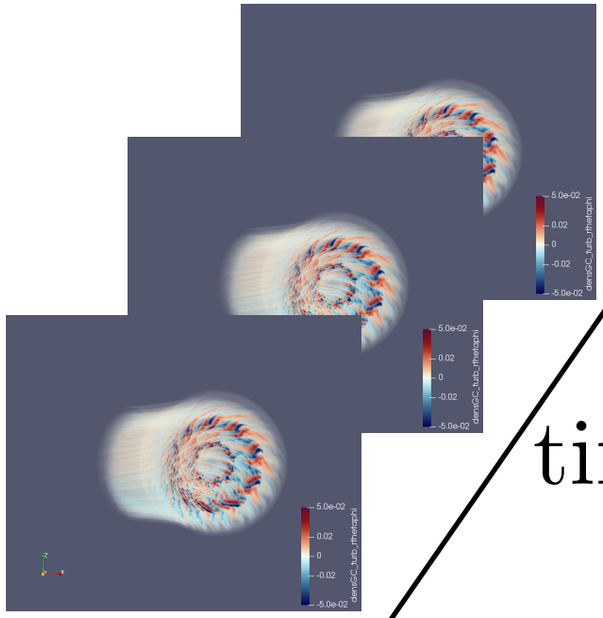
time

ρ

~kB

3D time series

Structures of Fluid moments δn



time

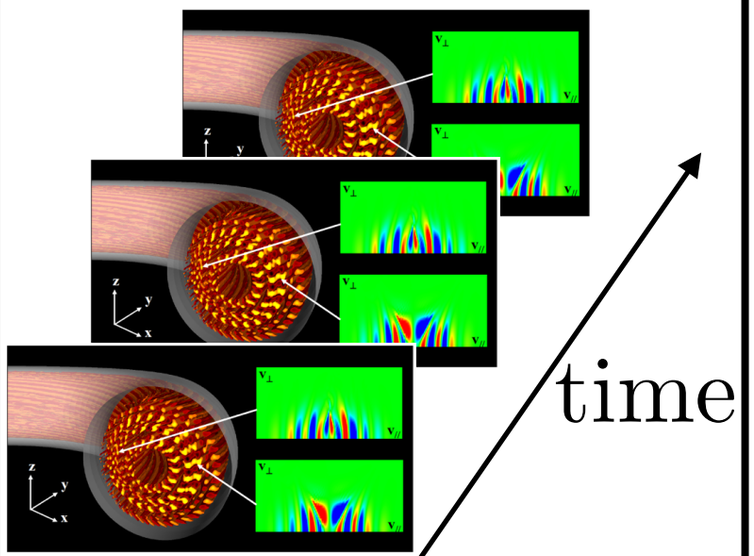
~10MB

This work

5D time series

Phase structure

δf



time

~10GB

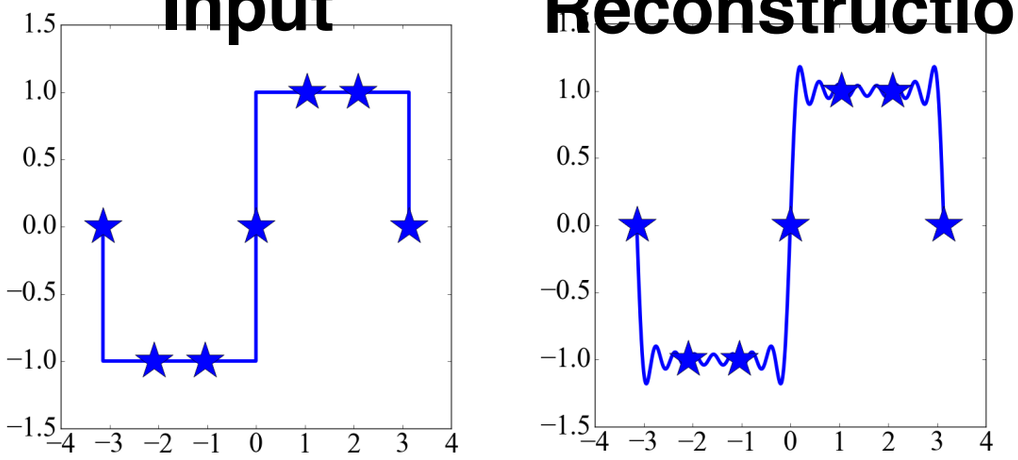
High dimensional + huge data

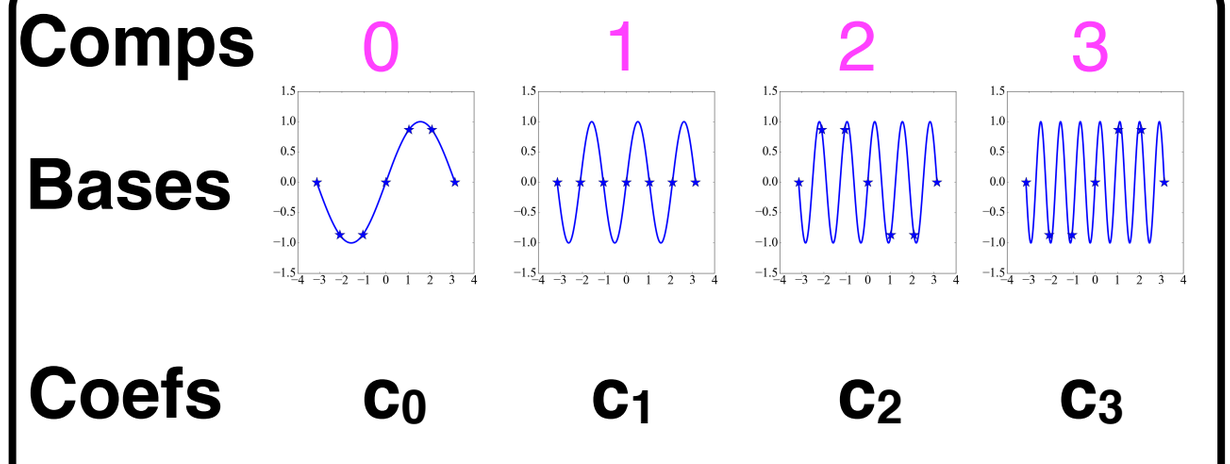
Conventional Study: 3D structures (like convective cells), 1D structures (stair case, stiffness in temperature gradient)

This work: Extracting phase space structure from the time series of 5D distribution function (pattern formation in phase space)

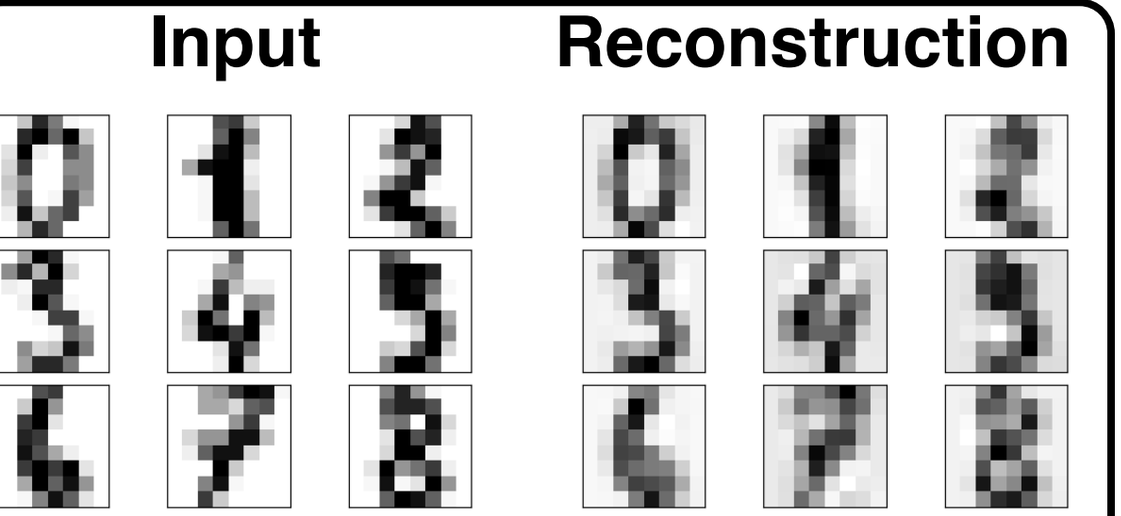
PCA and Fourier Transform

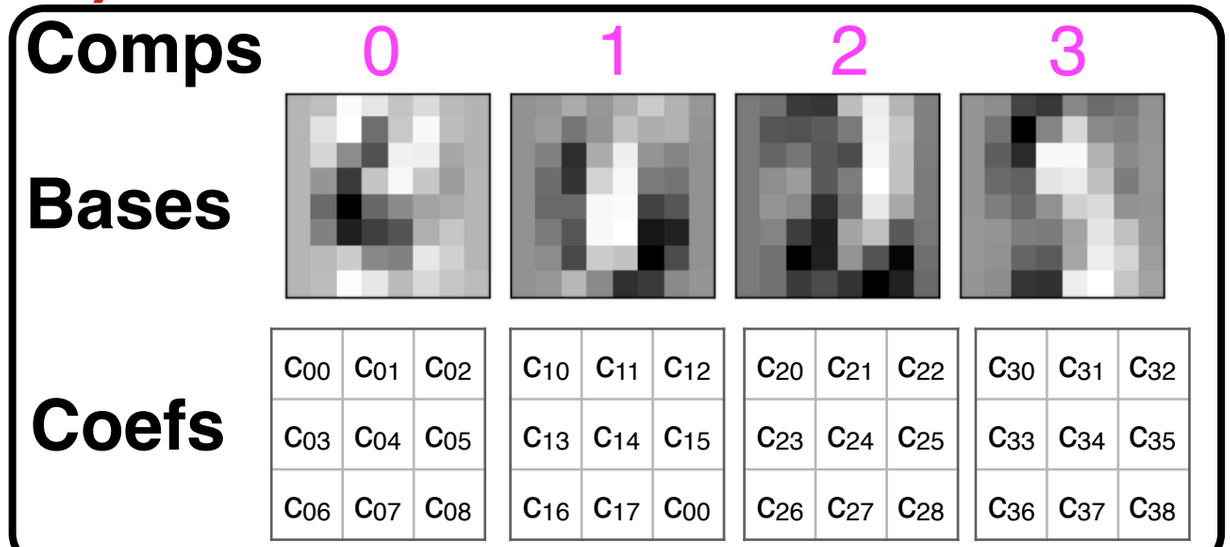
Fourier decomposition on signals





PCA (principal component analysis) on hand-written numbers





	Input	Bases	Coefficients	Reconstruction
FFT on signals	$N_{\text{signals}} \times (\text{scalar})$	$(N_{\text{signals}}, N_{\text{basis}}) \times (\text{scalar})$	$N_{\text{basis}} \times (\text{scalar})$	$\text{Sig}(m) = \sum_n C_n I_n(m)$
PCA on numbers	$N_{\text{numbers}} \times (\text{width}, \text{height})$	$N_{\text{basis}} \times (\text{width}, \text{height})$	$(N_{\text{numbers}}, N_{\text{basis}}) \times (\text{scalar})$	$\text{Img}(m, x, y) = \sum_n C_{n,m} I_n(x, y)$

Dimensionality reduction keeping important features in the data

Principal component analysis of distribution function

Analyzing 6D (3D space x 2D velocity x time) **Terabyte** data using Dask+Xarray

- Easily manage **out-of-memory data** (> 1TB) without MPI parallelization

Random sampling 3D phase space data $(\varphi, v_\parallel, w)$ from $(\mathrm{time}, r, \theta)$

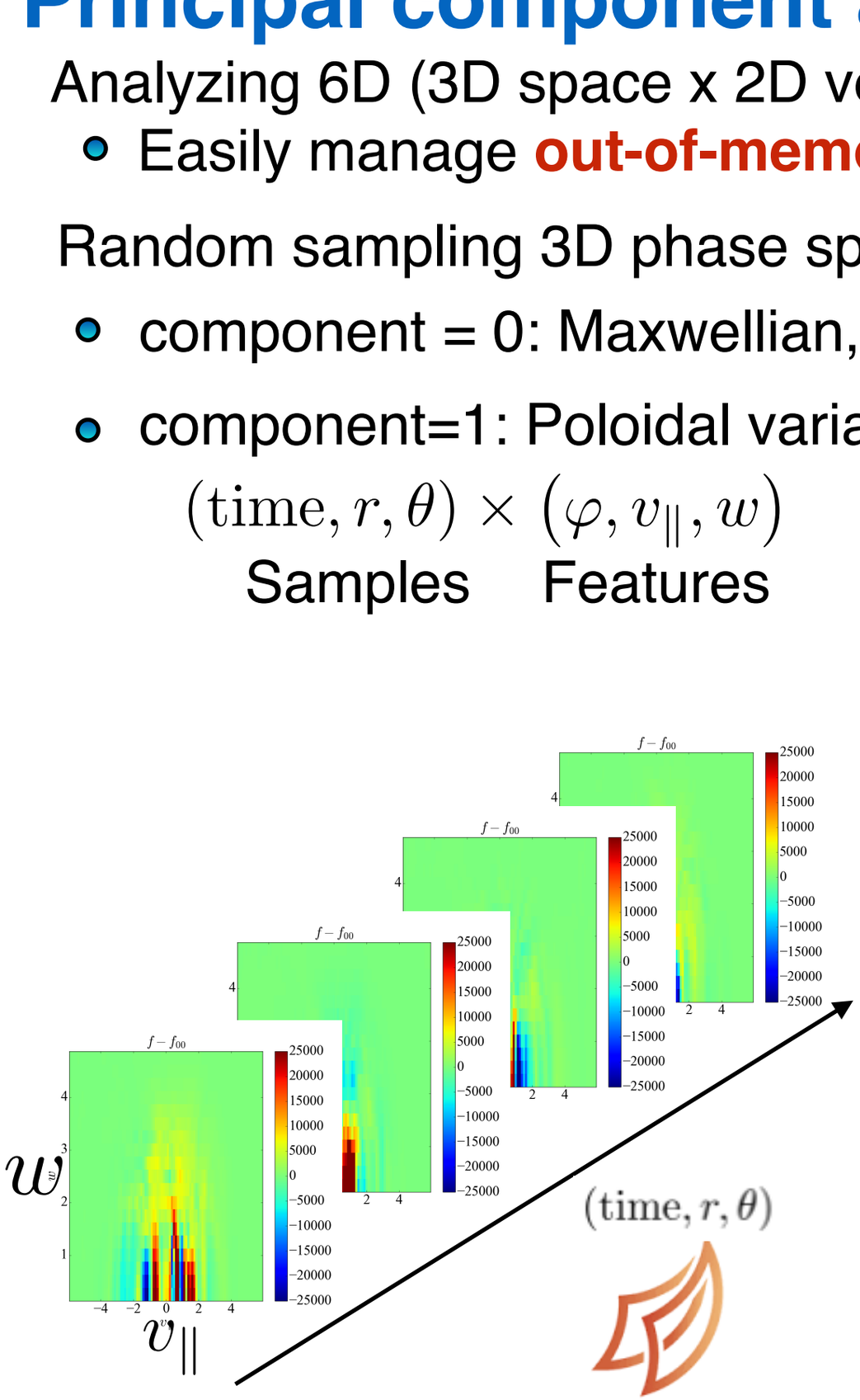
- component = 0: Maxwellian, representing radial structure in temperature
- component=1: Poloidal variation of distribution function

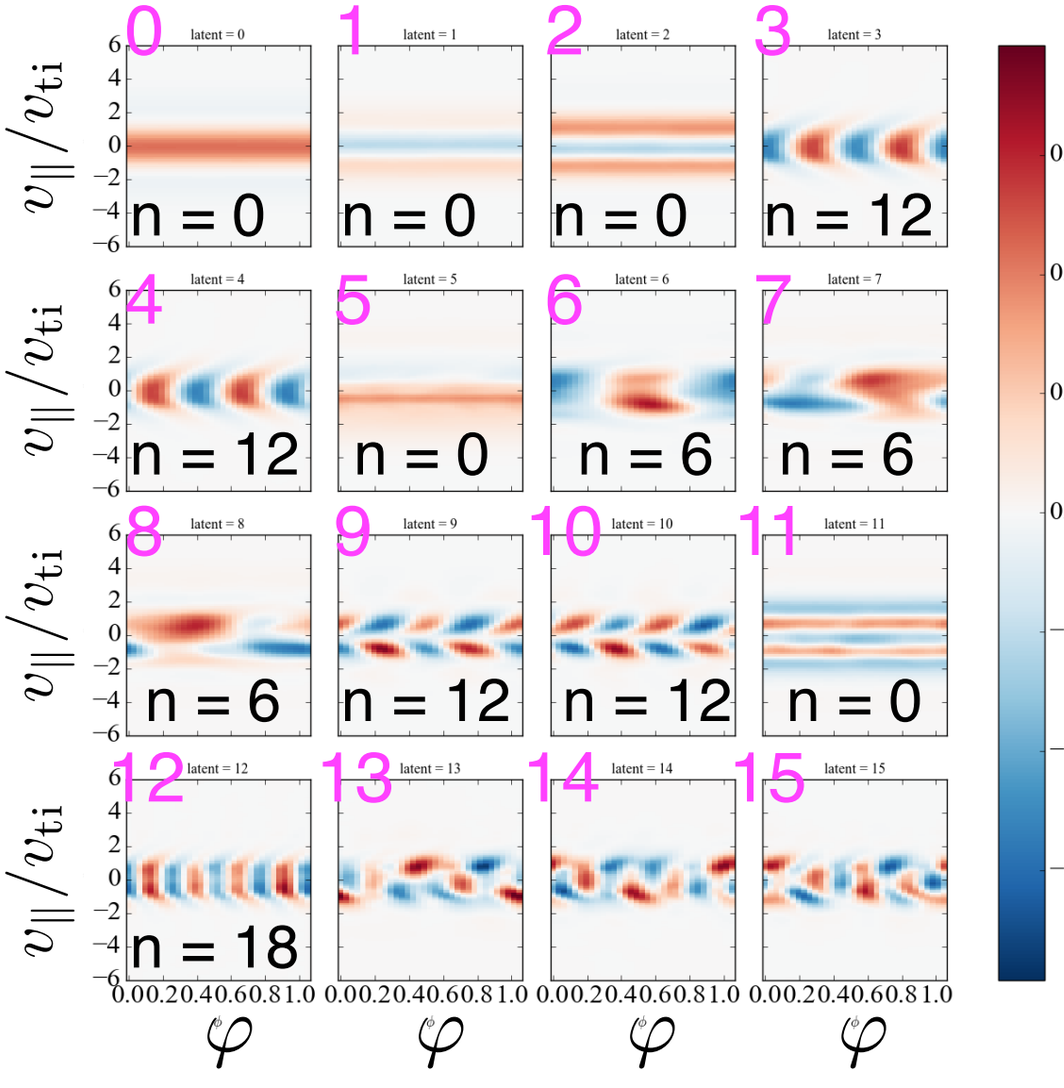
$$(\mathrm{time}, r, \theta) \times (\varphi, v_\parallel, w)$$

Samples Features

$$(\mathrm{components}) \times (\varphi, v_\parallel, w)$$

Eigen distribution function (basis)



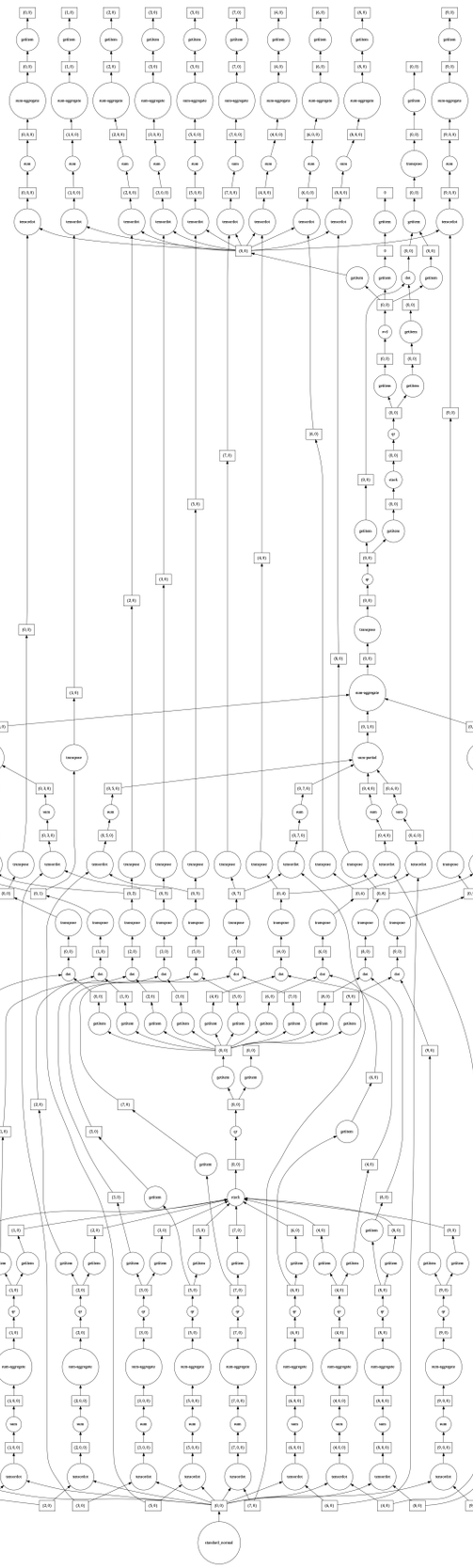


Task level parallelization with Dask.distributed

Huge data
(larger than memory)

Chunking into on memory tasks
which are managed by schedule

Task graph

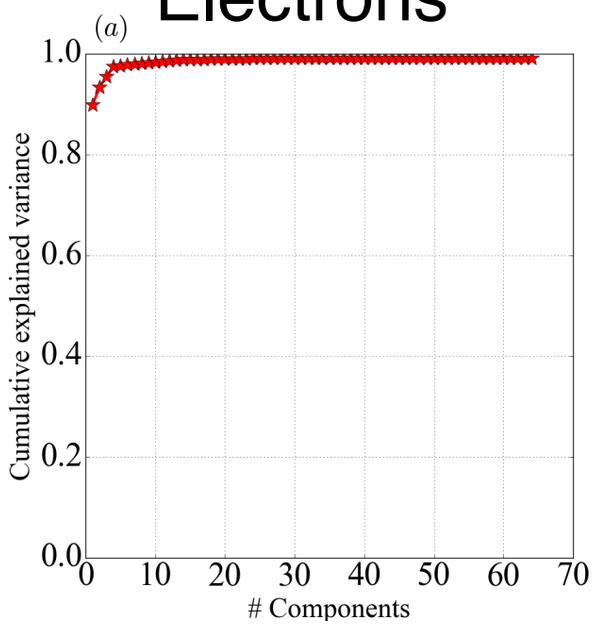


```
22      cluster = SLURMCluster(processes=1,
23                             cores=nb_cores,
24                             memory="150GB",
25                             project='GT5DSTC',
26                             name='worker',
27                             queue='S-M',
28                             walltime='00:30:00',
29                             interface='ipogif0')
30
31      # Typical size in incremental PCA
32      X = da.random.random((10000, 100000), chunks=(1000, 100000)).persist()
33      cluster.scale(nb_workers)
34      client = Client(cluster)
35
36      start = time.time()
37      u, s, v = da.linalg.svd_compressed(X, k=4)
38
39      future = u.compute()
40      end = time.time()
```

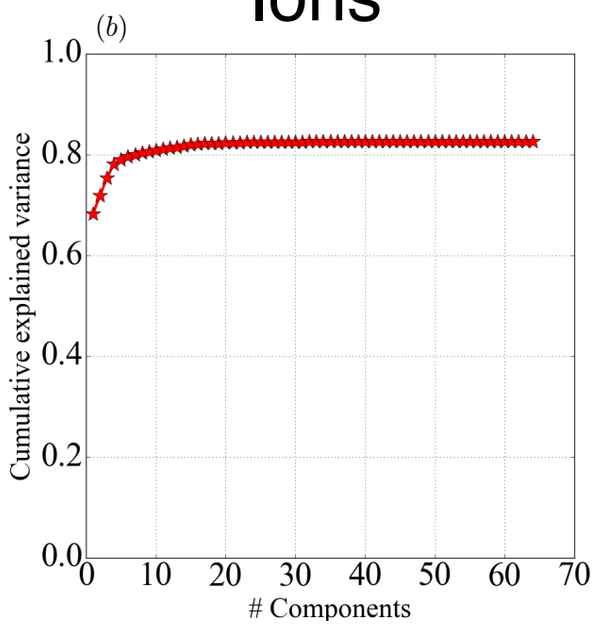
```
It took 28.68612051010132  [s] with nb_workers 1, nb_cores 1
It took 20.1727883815765 38 [s] with nb_workers 2, nb_cores 1
It took 17.562381744384766 [s] with nb_workers 4, nb_cores 1
It took 12.512330770492554 [s] with nb_workers 8, nb_cores 1
```

Large scale PCA over 16 TB data

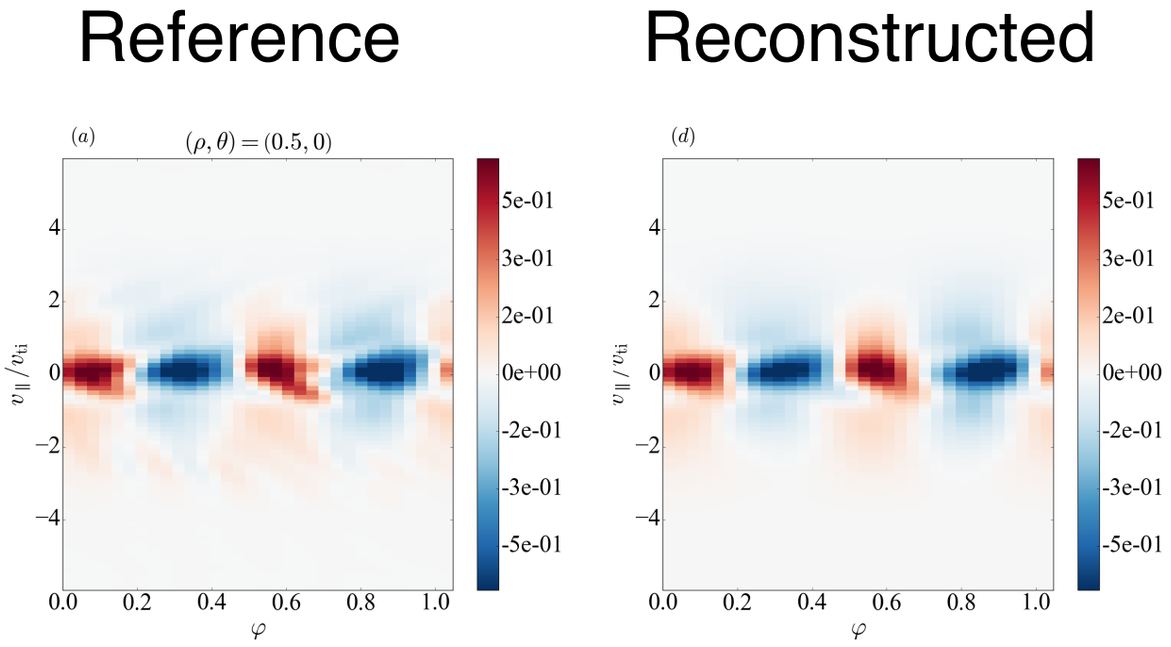
Electrons



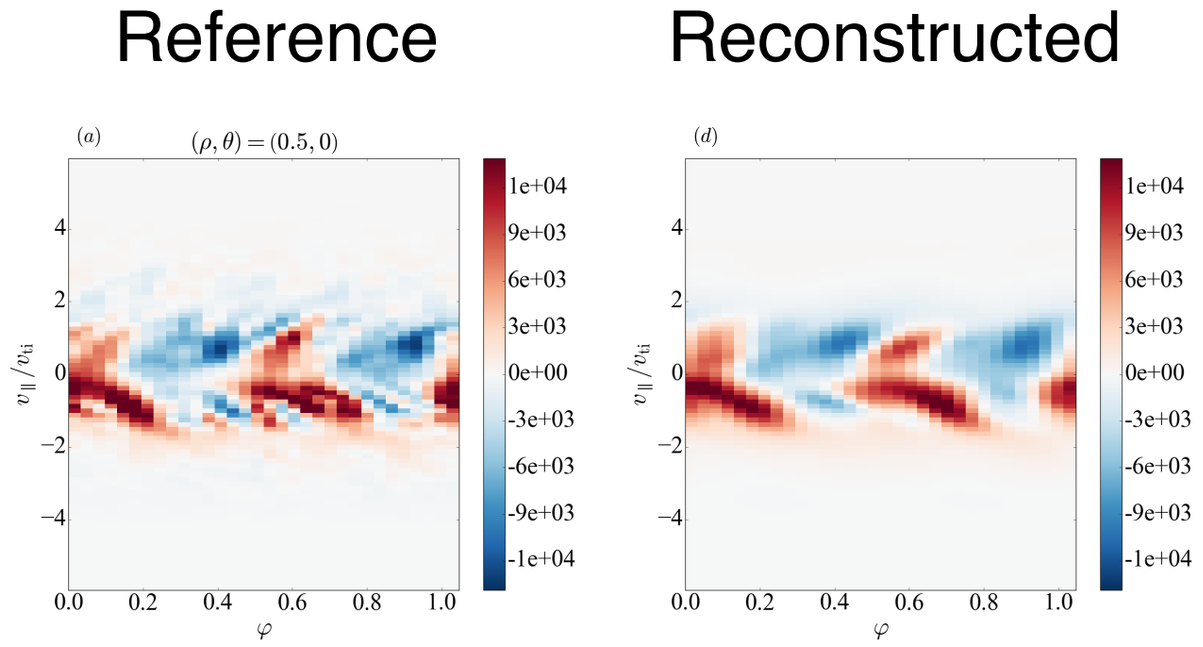
Ions



Reference Reconstructed



Reference Reconstructed

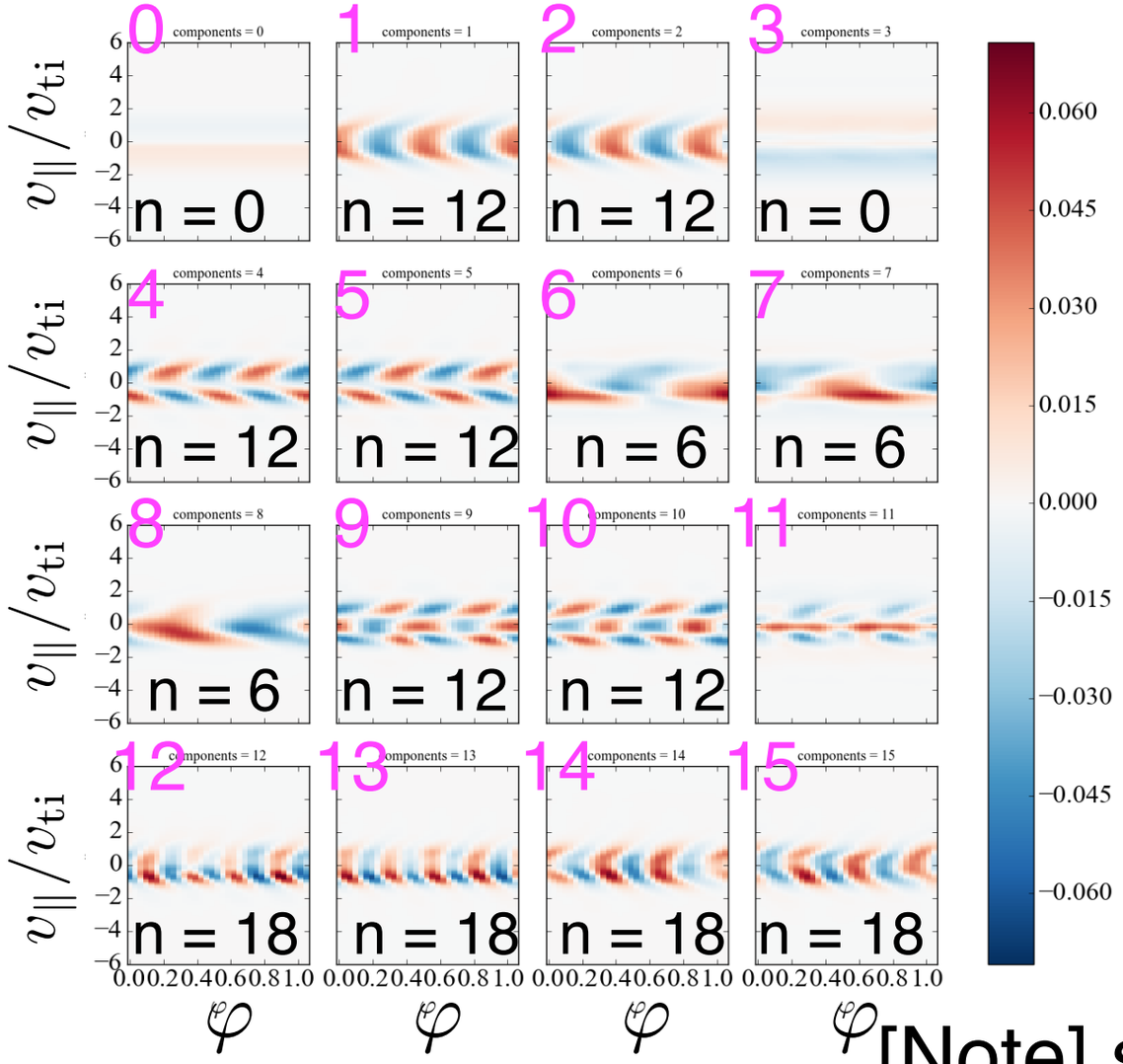


- Electron distribution function can be expressed with few components, while ion distribution function needs much more components
- **16 TB reduced into 7GB with 83 % of cumulative explained variance**

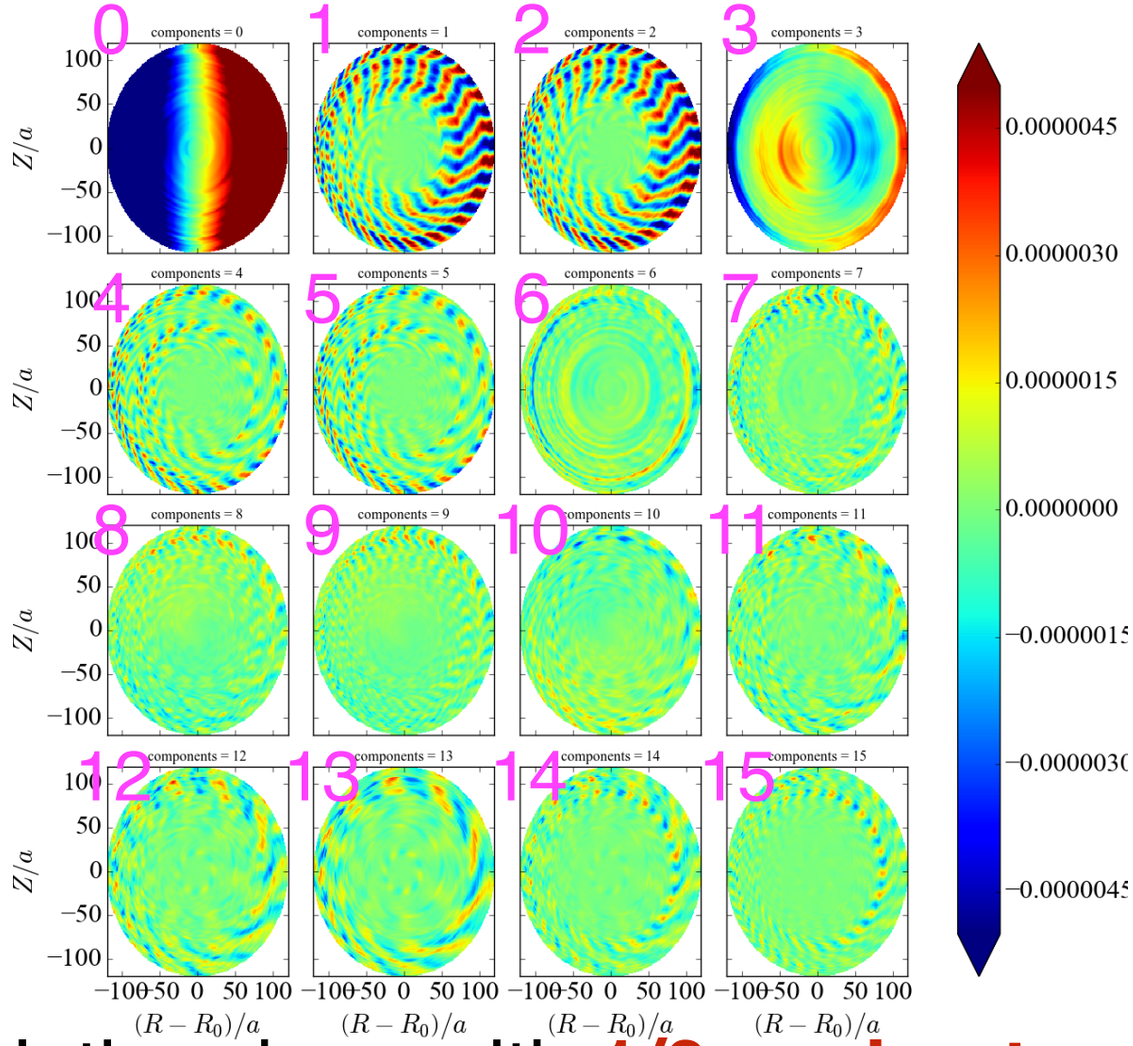
$$\underbrace{(\mathrm{time}, r, \theta)}_{\text{Samples}} \times \underbrace{(\varphi, v_{||}, w)}_{\text{Features}}$$

Decomposing turbulence into sub-physics

Phase space basis (w = 0.125)



Coefficients (r, theta)



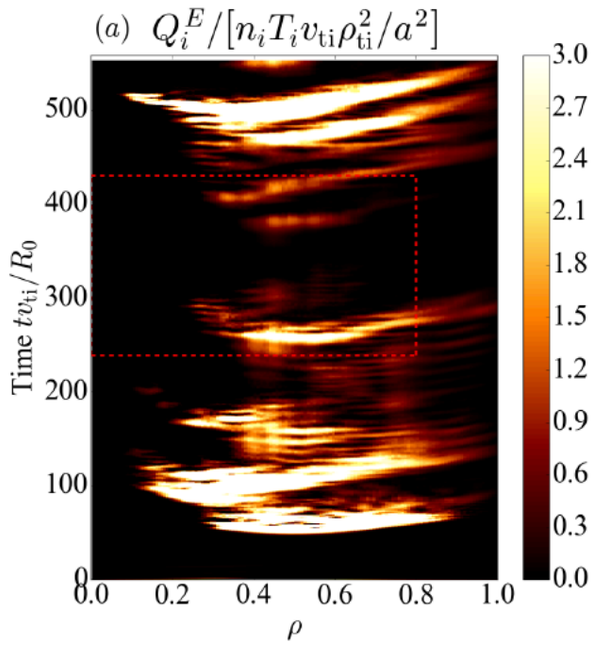
[Note] simulation done with **1/6 wedge torus**

0: Magnetic geometry	**1**: Ballooning (n=12)	**2**: Ballooning (n=12)	**3**: Convective cells
4: Ballooning (n=12)	**5**: Ballooning (n=12)	**6**: ballooning (n=6)	**7**: ballooning (n=6)
8: Ballooning (n=6)	**9**: Ballooning (n=12)	**10**: ballooning (n=12)	**11**: Turbulent
12: Ballooning (n=18)	**13**: Ballooning (n=18)	**14**: Ballooning (n=18)	**15**: Ballooning (n=18)

- Turbulence decomposed into **geometry** (0), **ballooning**, convective cells (3)

Energy flux recovered from reduced data

Reference



Approximated energy flux

$$\begin{aligned}\hat{Q}_i^E &= \int dv_\parallel d\mu 2\pi m_i^2 B_\parallel^* \left(\mathbf{v}_{E\times B}\cdot\nabla r\right)\left(\frac{m_i v_\parallel}{2}+\mu B\right)\hat{f} \\ &= \hat{Q}_{00}+\hat{Q}_{\rm mean}+\sum_j \hat{Q}_j,\end{aligned}$$

$$\begin{aligned}\hat{Q}_{00} &= \int dv_\parallel d\mu 2\pi m_i^2 B_\parallel^* \left(\mathbf{v}_{E\times B}\cdot\nabla r\right)\left(\frac{m_i v_\parallel}{2}+\mu B\right) f_{00} \\ \hat{Q}_{\rm mean} &= \int dv_\parallel d\mu 2\pi m_i^2 B_\parallel^* \left(\mathbf{v}_{E\times B}\cdot\nabla r\right)\left(\frac{m_i v_\parallel}{2}+\mu B\right)\overline{f} \\ \hat{Q}_j &= \int dv_\parallel d\mu 2\pi m_i^2 B_\parallel^* \left(\mathbf{v}_{E\times B}\cdot\nabla r\right)\left(\frac{m_i v_\parallel}{2}+\mu B\right)\mathbf{p}_j x_j\end{aligned}$$

Energy flux by PCs

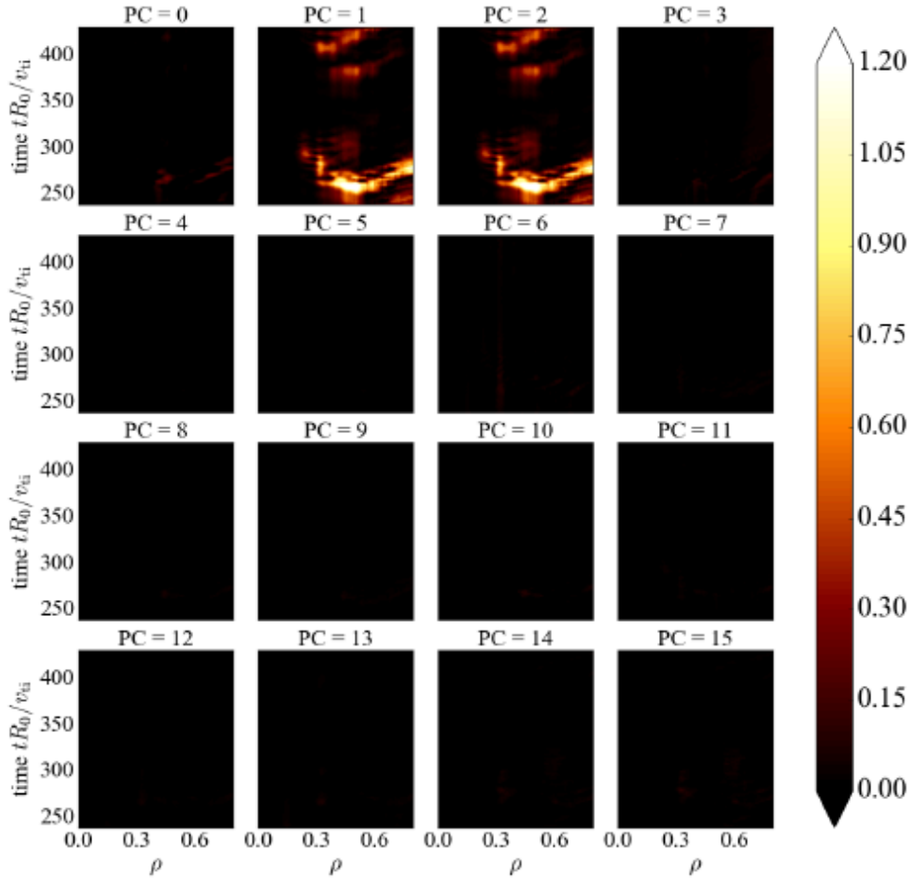


Figure 13: Spatio-temporal evolution of the reconstructed ion turbulent energy flux $\hat{Q}_i^E / \left[n_i T_i v_{\rm ti} \rho_{\rm ti}^2/a^2\right]$ driven by the first 16 principal components (PCs).

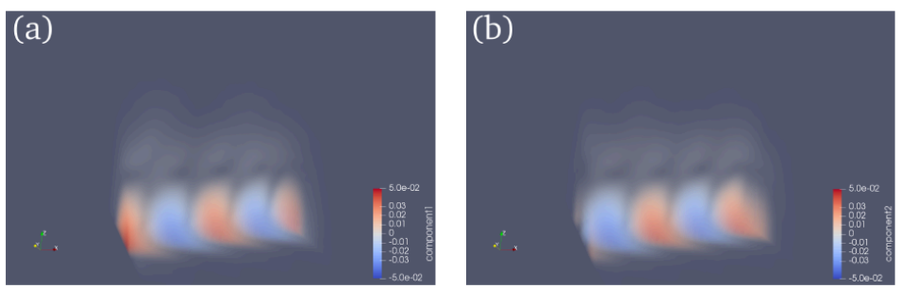


Figure 14: The principal components 1 (a) and 2 (b) of phase space bases $\mathbf{p}_j$. The directions x, y, and z correspond to the directions φ, $v_\parallel$, and w, respectively.

- 3 order reduction of the data size, still keeping the important properties like avalanche like transport [1]

[1] Y. Asahi, to be submitted to PoP

Current plans/targets

MPI version of GYSELA/GKV mini apps with Kokkos

- Goal: Submit a paper to a SC workshop (P3H3PC or WACCPD2020)
- Good to show scalable approach over multiple platforms (including Fugaku)
  Using std::thread + Kokkos and OpenMP4.5 offloading + task

Scalable data analysis based on Dask

- Goal: Submit a paper to computer science and/or physics journal
- Integrate Dask into GYSELA diags through PDI (developed by J. Bigot)
- In situ machine learning (starting with PCA?)

Target Conferences/Workshops

- SC20, 15 November, Atlanta, US

  P3H3PC: Performance Portability, and Productivity

  WACCPD2020: Directive based implementation needed