OpenACC Summit 2020

Sep. 01. 2020

Accelerating Gyrokinetic Tokamak Simulation (GTS) Code using OpenACC

M.G. Yoo, C.H. Ma, S. Ethier, Jin Chen, W.X. Wang, E. Startsev

Princeton Plasma Physics Laboratory, Princeton, U.S.A.

Overview of porting GTS to GPU

GTS (Gyrokinetic Tokamak Simulation)

- A global gyrokinetic particle simulation code for micro-turbulence study in tokamak
- Particle-In-Cell algorithm (particles + grid-based field solve)
- Recently upgraded for physics studies associated with the thermal quench transport

Porting to GPU machine

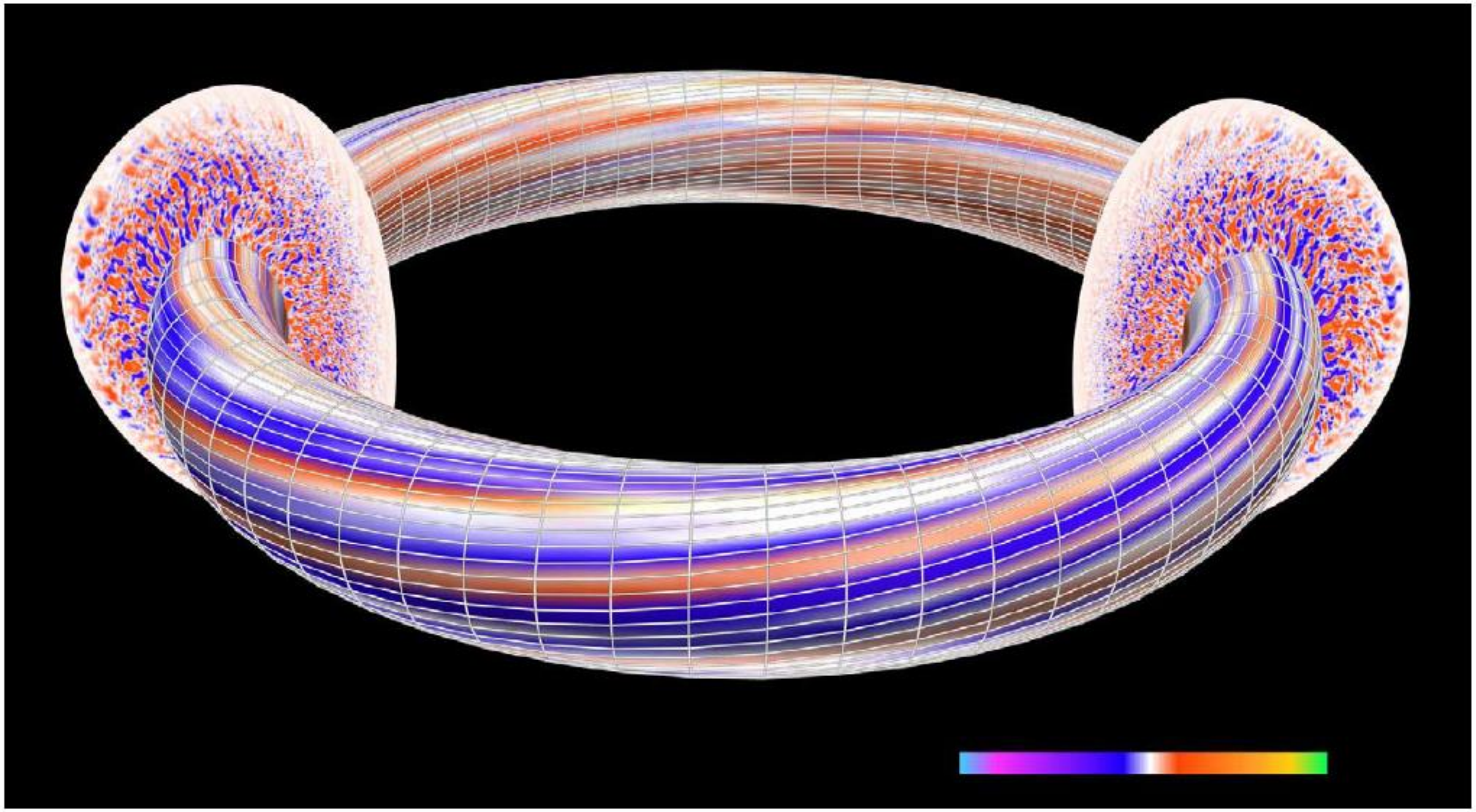
- Attended GPU Hackathon in 2019 at Princeton University (Mentor: Rueben Budiardja (ORNL))
- OpenACC directives ported the code to GPU keeping compatibilities with CPU machine
- GTS is now running ***production simulations*** on ***Traverse*** with a significant acceleration, making efficient use of the Traverse computational resource

  - Significant speed-up **(>20x)** for the particle parts

  - Field solve part (Poisson equation) will be ported to GPU via some libraries (ex. PETSc, Hypre, AMGx)

GTS (Gyrokinetic Tokamak Simulation)

- **A global gyrokinetic particle simulation code to study the micro turbulence physics of the fusion plasma in tokamaks**
  - δf particle-in-cell code in 3-dimensional curvilinear coordinate
  - Mainly written in Fortran, partly in C
  - Parallelized using MPI + OpenMP (previously), now using MPI+OpenACC



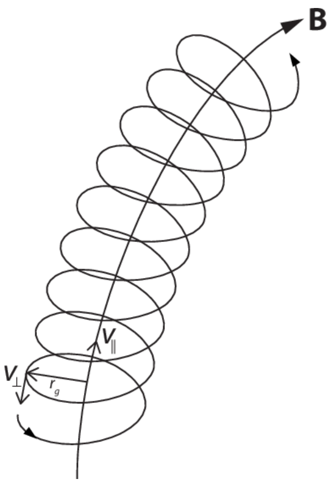
Gyrokinetic Equation

- **The gyrokinetic equation for particle distribution in 5-dimension phase space**

f_s: gyro-center distribution function

$$\frac{\partial f_s}{\partial t} + \frac{1}{B^*}\nabla_5 \cdot (\dot{\mathbf{Z}} B^* f_s) = \sum_b C[f_s, f_b]$$

$$\mathbf{Z} = \{\mathbf{R}, v_\parallel, \mu\} = \{a, \theta, \varphi, v_\parallel, \mu\}$$



[G. Hunt, Ph.D. Thesis, University of Leicester, 2016]

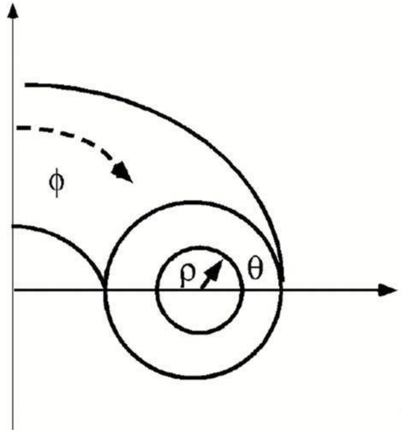
a (or ρ): radial coordinates (a flux surface label)

θ and ϕ: poloidal and toroidal angle

$v_\parallel$: parallel velocity

$\mu = m_s v_\perp^2 / 2B$: magnetic moment

$B_* = B + (m_s v_\parallel / e_s)\mathbf{b} \cdot \nabla \times \mathbf{b}$



Gyrokinetic Poisson Equation

- **Electron and Ion densities from the distribution function**

$$\delta\bar{n}_i(\mathbf{x}) = \int \delta f_i(\mathbf{R},\mu,v_\parallel)\delta(\mathbf{R}-\mathbf{x}+\rho_i)d\mathbf{R}d^3v,$$

$$\delta\bar{n}_e(\mathbf{x}) = \int \delta f_e(\mathbf{R},\mu,v_\parallel)\delta(\mathbf{R}-\mathbf{x}+\rho_e)d\mathbf{R}d^3v \approx \int \delta f_e d^3v,\ \ (\rho_e \to 0),$$

$$\text{GK transform } \Phi(\mathbf{x}) \to \bar{\Phi}(\mathbf{R},\mu): \quad \bar{\Phi}(\mathbf{R},\mu) = \frac{1}{2\pi}\int \phi(\mathbf{x})\delta(\mathbf{x}-\mathbf{R}-\rho)d\mathbf{x}d\Theta$$

- **Quasi-neutrality and gyrokinetic Poisson equation**

$$\sum_i \left[Z_i n_{i,0} + Z_i \nabla_\perp \cdot \frac{n_{i,0}}{B\Omega_i}\nabla_\perp\Phi + Z_i\delta\bar{n}_i\right] = n_{e,0} + \delta n_e$$

$$-\nabla_\perp \cdot \frac{Z_i n_{i,0}}{B\Omega_i}\nabla_\perp\Phi = Z_i\delta\bar{n}_i - \delta n_e$$

[Dubin, *et. al.*, Phys. Fluids 26, 3524 (1983)]

Particle-In-Cell Method

- **The distribution function $\boldsymbol{f_s}$ is represented by Marker Particles**

$$f_s \approx \sum_{i=1}^{N_M} w_i \frac{\delta(x - x_i)\delta(v - v_i)}{J(x_i)}$$

- **Equation of motion**

$$\frac{d\rho_\parallel}{dt} = \frac{(\mathbf{B}_0^* + \boldsymbol{\delta}\mathbf{B})}{\mathbf{B_0} \cdot (\mathbf{B}_0^* + \boldsymbol{\delta}\mathbf{B})} \cdot \left[-\frac{1}{q_s} \nabla H_0 \right]$$

$$\boldsymbol{v} = \frac{1}{\mathbf{B_0} \cdot (\mathbf{B}_0^* + \boldsymbol{\delta}\mathbf{B})} \left[\frac{1}{q_s} \frac{\partial H_0}{\partial \rho_\parallel} (\mathbf{B}_0^* + \boldsymbol{\delta}\mathbf{B}) + \frac{1}{q_s} \mathbf{B_0} \times \nabla H_0 \right]$$

$$\nabla H_0 = \left(\frac{m_s \left(v_\parallel^0\right)^2}{B_0} + \frac{\mu}{q} \right) \nabla B_0 + \nabla \overline{\Phi}$$

$$\rho_\parallel = \frac{m_s v_\parallel^0}{q_s B_0}$$

$$\frac{1}{q_s} \frac{\partial H_0}{\partial \rho_\parallel} = \boldsymbol{B_0} \cdot \boldsymbol{v} = v_\parallel^0$$

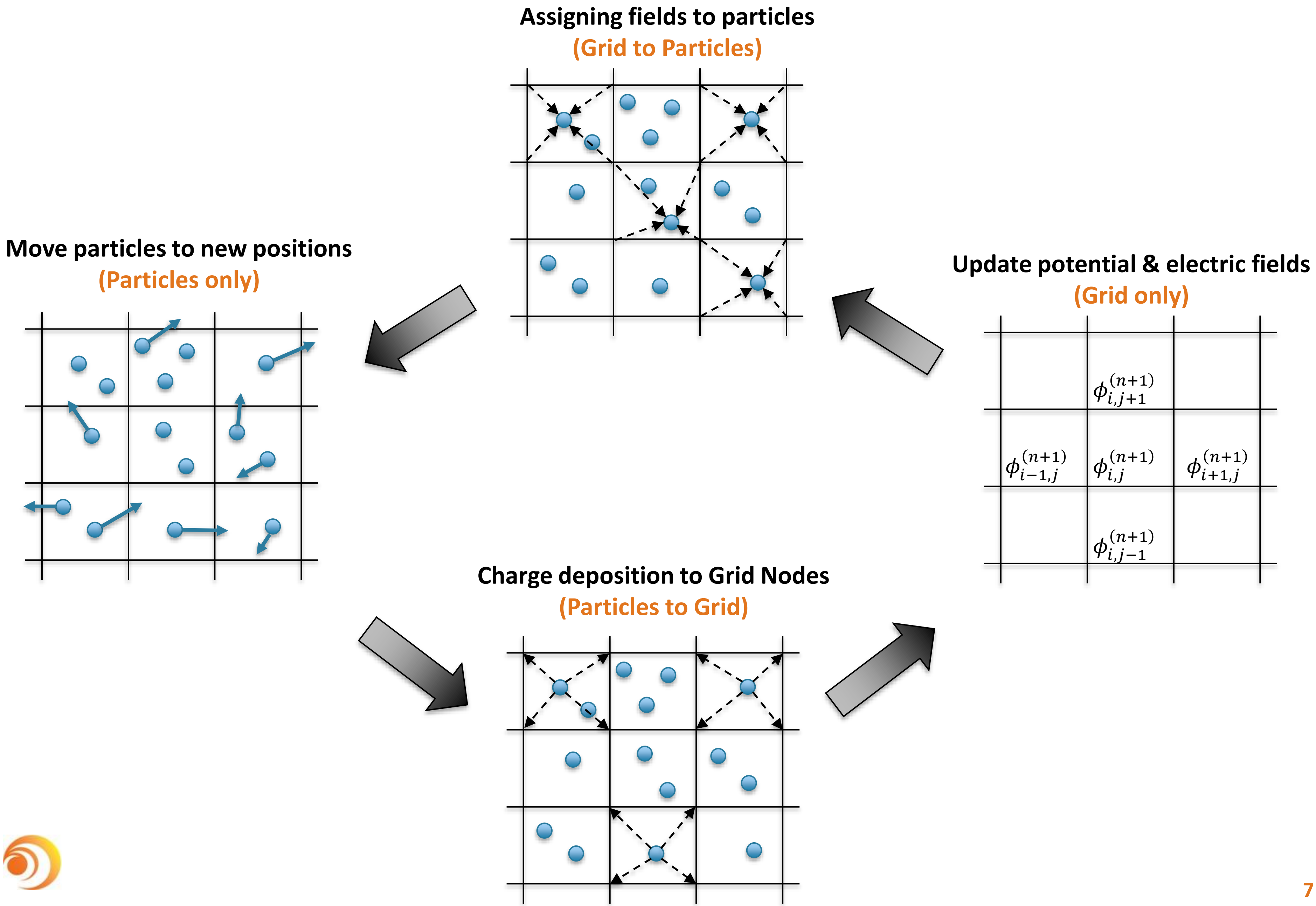
- **$\boldsymbol{\delta f}$ weight evolution equation**

$$\frac{df_{\text{tot}}}{dt} = \frac{\partial f_{\text{tot}}}{\partial t} + \dot{\mathbf{Z}} \cdot \nabla_{\mathbf{Z}} f = 0$$

$$f_{\text{tot}} = f_0 + \delta f$$

$$\Rightarrow \quad \frac{d\delta f}{dt} = -\frac{df_0}{dt} = -\dot{\mathbf{Z}} \cdot \nabla_{\mathbf{Z}} f_0 = -\left(\dot{\mathbf{Z}}_\mathbf{0} + \dot{\mathbf{Z}}_\mathbf{1}\right) \cdot \nabla_{\mathbf{Z}} f_0$$

Particle-In-Cell Method



Acceleration of Particle Parts by OpenACC

- **Particle parts are easily and efficiently parallelized by OpenACC**
  - Particles are independent from each other and has a single level loop
  - A simple acc parallel directives is very efficient for a huge number of marker particles ($N_M > 10^6$)
  - Particle arrays are global variables and created on device side to reduce the communication time

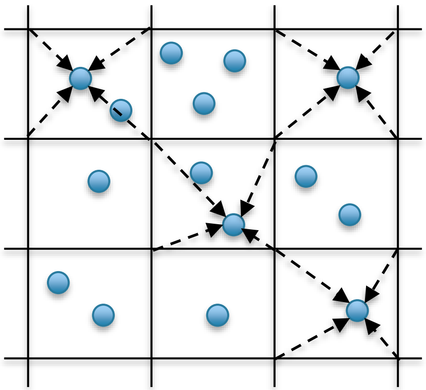
```
!$acc declare create(P_x,P_v,P_w,P_E,P_B,P_gradB, ...)
```

- **Assigning fields to particles (Grid to Particles)**

```
!$acc parallel loop present(P_x,P_E, ...)
do m=1, Nm
   ! Find a matching grid index
   xid = floor((P_x(m)-x0)/dx)

   ! Calculate a weight for linear interpolation
   wx = (P_x(m)-Gx(xid))/dx

   ! Calculate Fields at particle position
   P_E(m) = (1-wx)*E(xid) + wx*E(xid+1)
   P_B(m) = ...
   ...
   ...
enddo
```



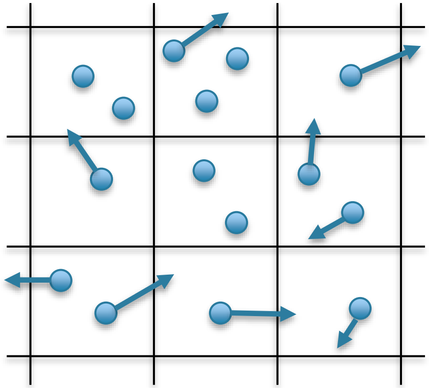
Acceleration of Particle Parts by OpenACC

- **Move particles to new positions (Particles only)**

```
!$acc parallel loop present(P_x,P_E,P_B, ...)
do m=1,Nm
   ! Calculate velocity and acceleration
   dxdt = f(P_B,P_gradB,P_E, ...)
   dvdt = g(P_B,P_gradB,P_E, ...)

   ! Calculate driving force for weight
   dwdt = h(P_B,P_gradB,P_E, ...)

   ! Update particle's information
   x(m) = x(m) + dxdt*dt
   v(m) = x(m) + dvdt*dt
   w(m) = w(m) + dwdt*dt
enddo
```

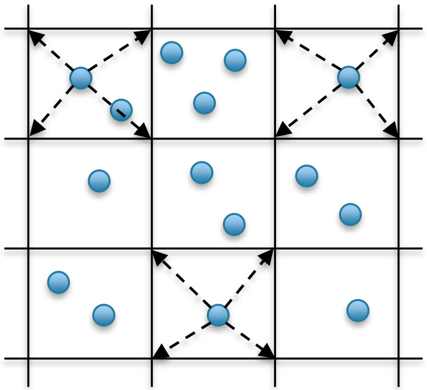


- **Charge deposition to Grid Nodes (Particles to Grid)**

```
!$acc parallel loop present(P_x,P_E,P_B, ...)
do m=1,Nm
   ! Find a matching grid index
   xid = floor((P_x(m)-x0)/dx)

   ! Calculate weight evolution
   wx = (P_x(m)-G_x(xid))/dx

   !$acc atomic update
   G_n(xid)   = G_n(xid)   + (1-wx)*P_w
   !$acc atomic update
   G_n(xid+1) = G_n(xid+1) + wx*P_w
enddo
```



Gyrokinetic Poisson Equation

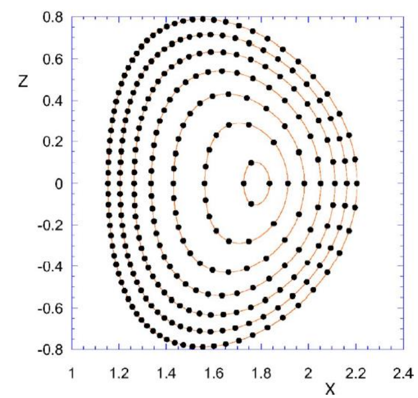
- **Coupled (2D+1D) field equations on Unstructured Mesh (use FEM)**

$$\alpha\delta\Phi + \beta\nabla_\perp\cdot\left(\sum_i g_i \nabla_\perp\delta\Phi\right) = b - \beta\nabla_\perp\cdot\left(\sum_i g_i \frac{d\langle\Phi\rangle}{da}\nabla a\right)$$

Solved iteratively

$$\frac{d}{da}\left[\sum_i \langle g_i g^{aa}\rangle \mathcal{V}_a' \frac{d\langle\Phi\rangle}{da}\right] = -\frac{d}{da}\left[\sum_i\langle g_i\nabla_\perp\delta\Phi\cdot\nabla a\rangle\mathcal{V}_a'\right] + \frac{\mathcal{V}_a'}{\beta}\langle b\rangle$$

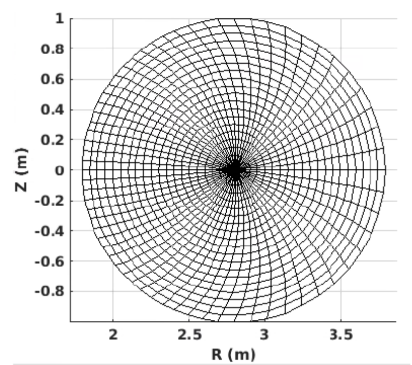
Unstructured grid



- **A direct 3D field equation on Structured Mesh (use FDM)**

$$-\nabla\cdot\left[\epsilon_0\overleftrightarrow{\mathbf{g}} + \sum_s \frac{n_s m_s}{B^2}\left(\overleftrightarrow{\mathbf{g}} - \frac{\boldsymbol{BB}}{B}\right)\right]\cdot\nabla\Phi = e(\delta\bar{n}_i - \delta n_e)$$

Structured grid



- **Algebraic Multigrid solver is used to invert linearized equations**

$$\boldsymbol{Ax = b}$$

- BoomerAMG method in Hypre library through PETSc interface

- (CPU version) vs (cuda version)

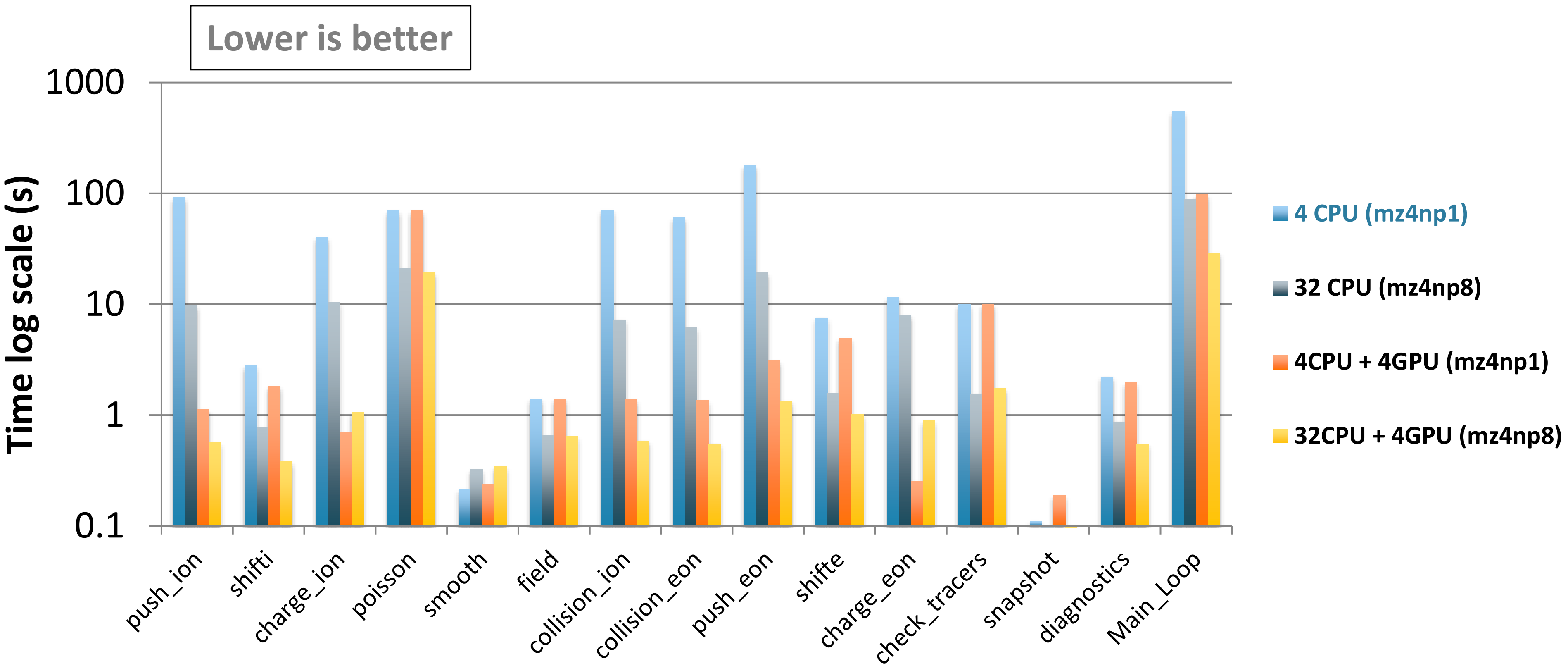
Traverse Cluster at Princeton University

Traverse consists of:

- 46 IBM AC922 Power 9 nodes, with each node having
  - 2 IBM Power 9 processors (sockets)
    - 16 cores per processor
    - 4 hardware threads per core
  - 32 cores per node
  - 256 GB of RAM per node
  - 4 NVIDIA V100 GPUs (2 per socket) with 32GB of memory each
  - 3.2TB NVMe (solid state) local storage (not shared between nodes)
  - EDR InfiniBand, 1:1 per rack, 2:1 rack to rack interconnect
  - GPFS high performance parallel scratch storage: 2.9PB raw
  - Globus transfer node (10 GbE external, EDR to storage)
- InfiniBand Network
  - EDR InfiniBand (100 Gb/s)
  - Fully non-blocking (1:1) within a chassis, 2:1 oversubscription between chassis.

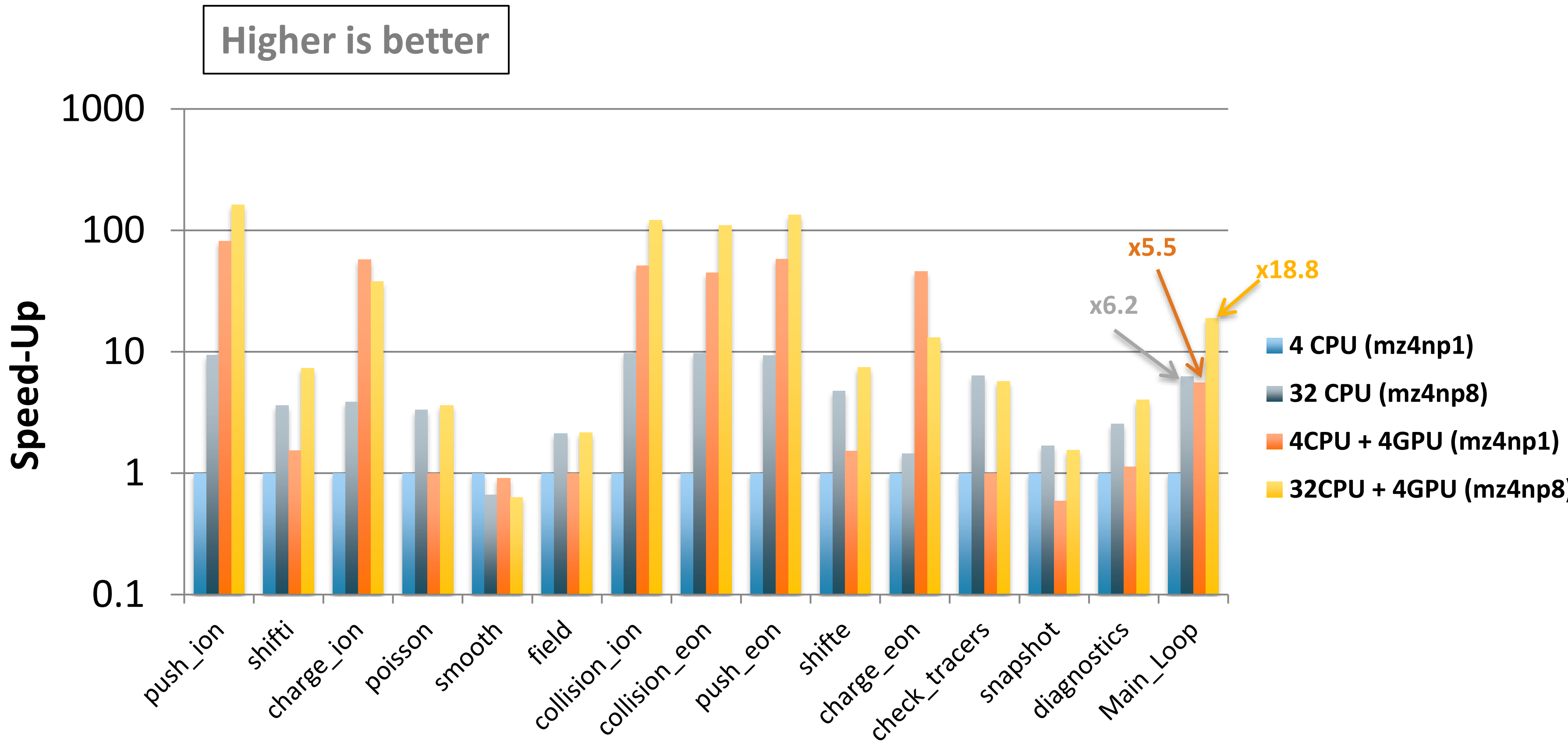
Elapsed Time

micell=50, MGRID= 45,137
mpsi=100 MISUM= 9,007,200



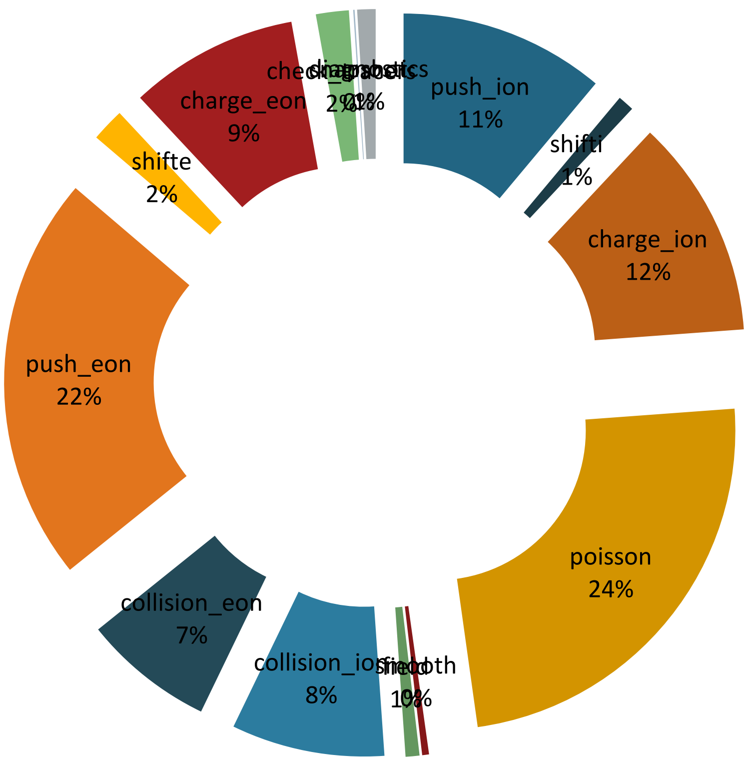
Speed-Up Factor

micell=50, MGRID= 45,137
mpsi=100 MISUM= 9,007,200

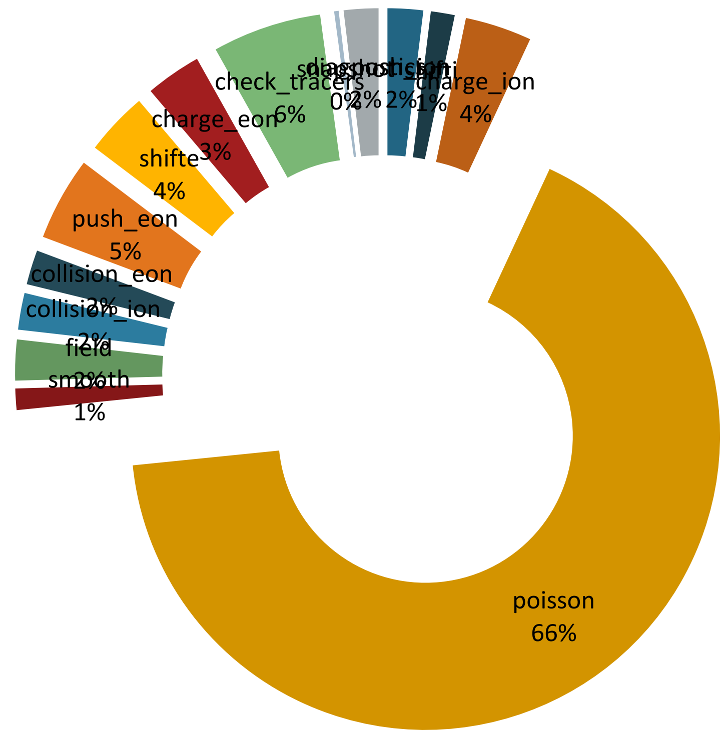


Full Power of 1 node on Traverse

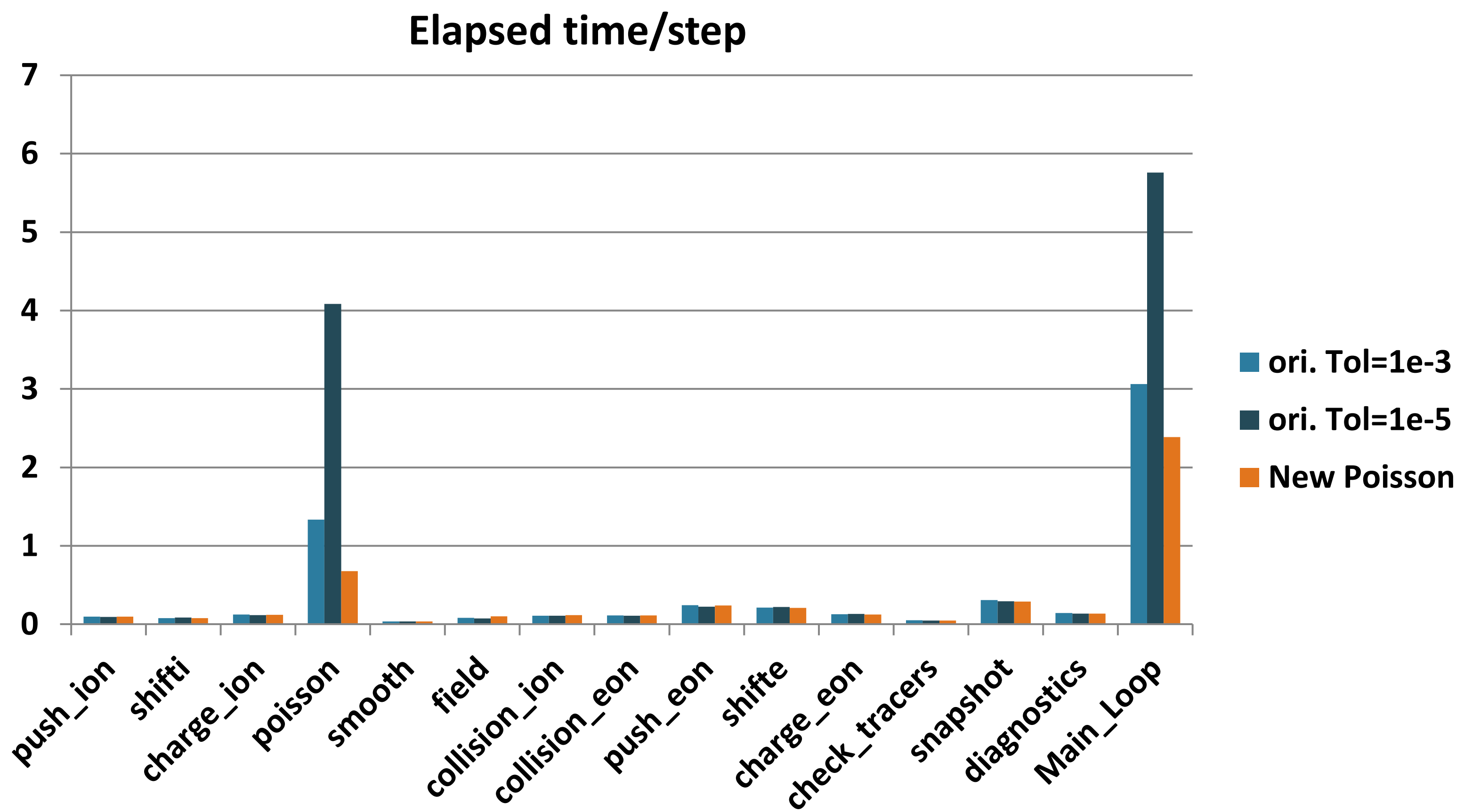
32 CPU



32 CPU + 4 GPU

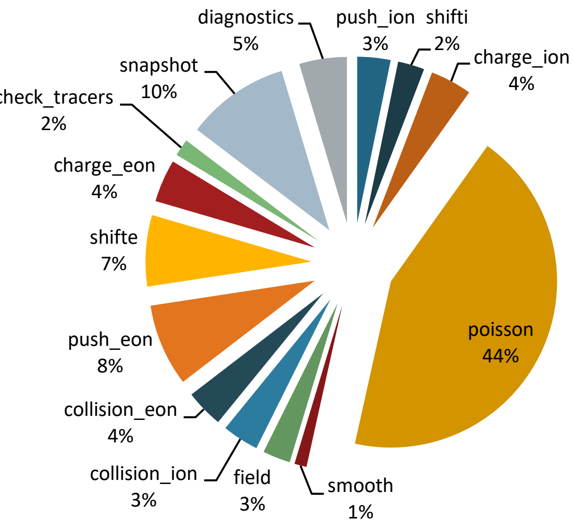


Performance Comparison
(original Poisson vs new 3D Poisson)

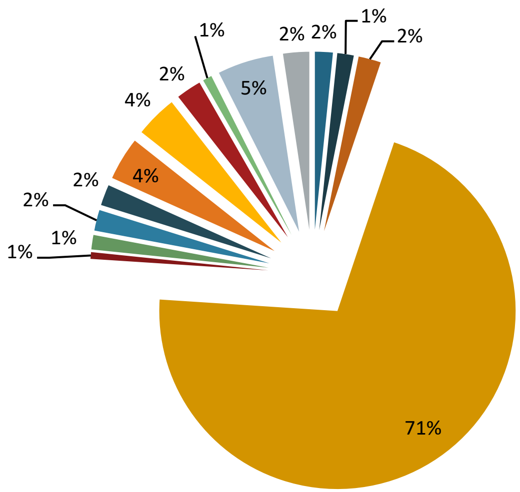


Breakdown of Elapsed Time

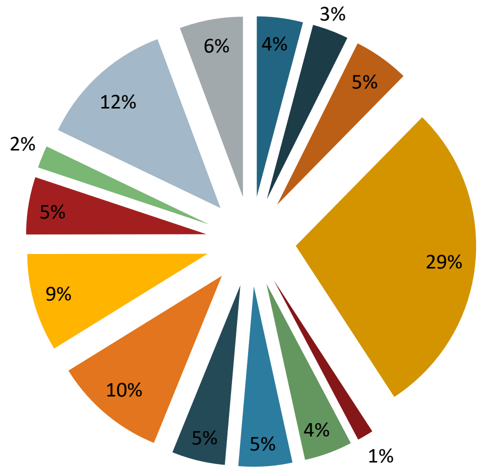
Ori. Poisson (Tol=1e-3)



Ori. Poisson (Tol=1e-5)



New Poisson 3D



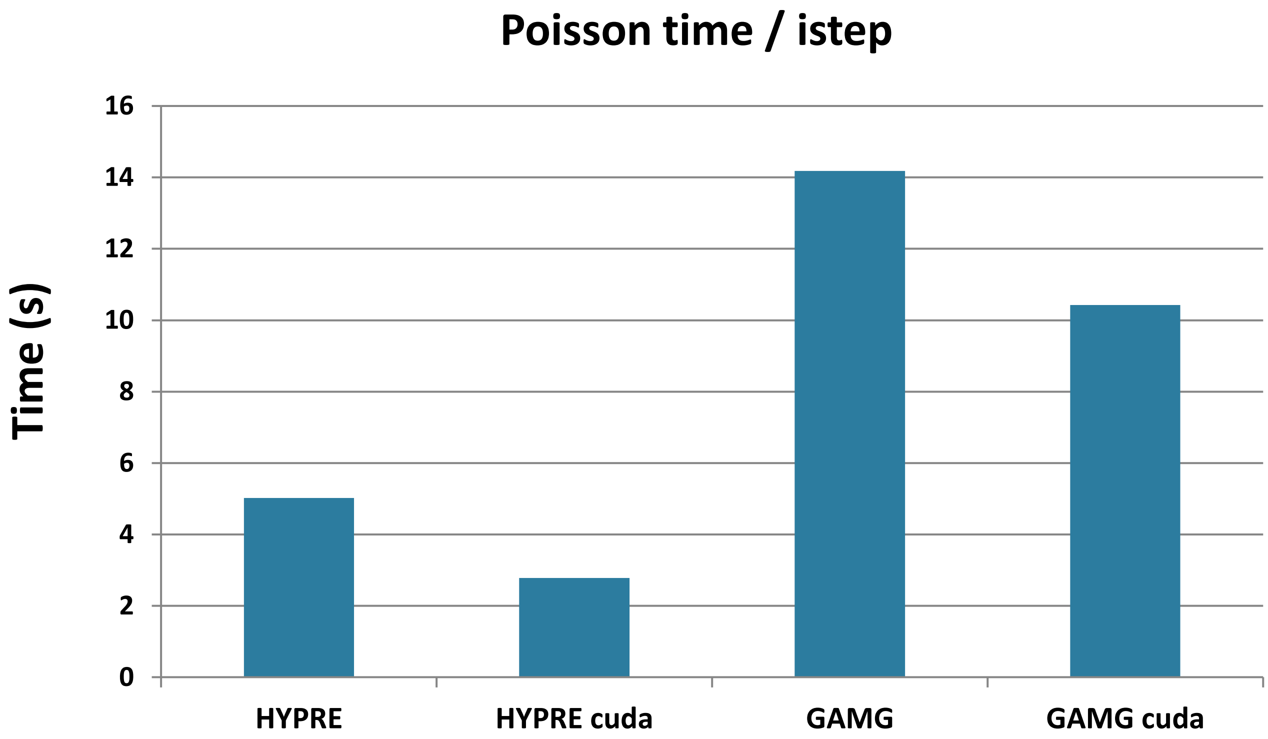
Algebraic Multigrid Solver: (HYPRE) vs (GAMG)

micell=50,
mpsi=100

MGRID= 45,137
MISUM= 9,007,200

1 Node 4 MPI ranks
mzetamax=4
npartdom=1

1 CPU / 1 GPU



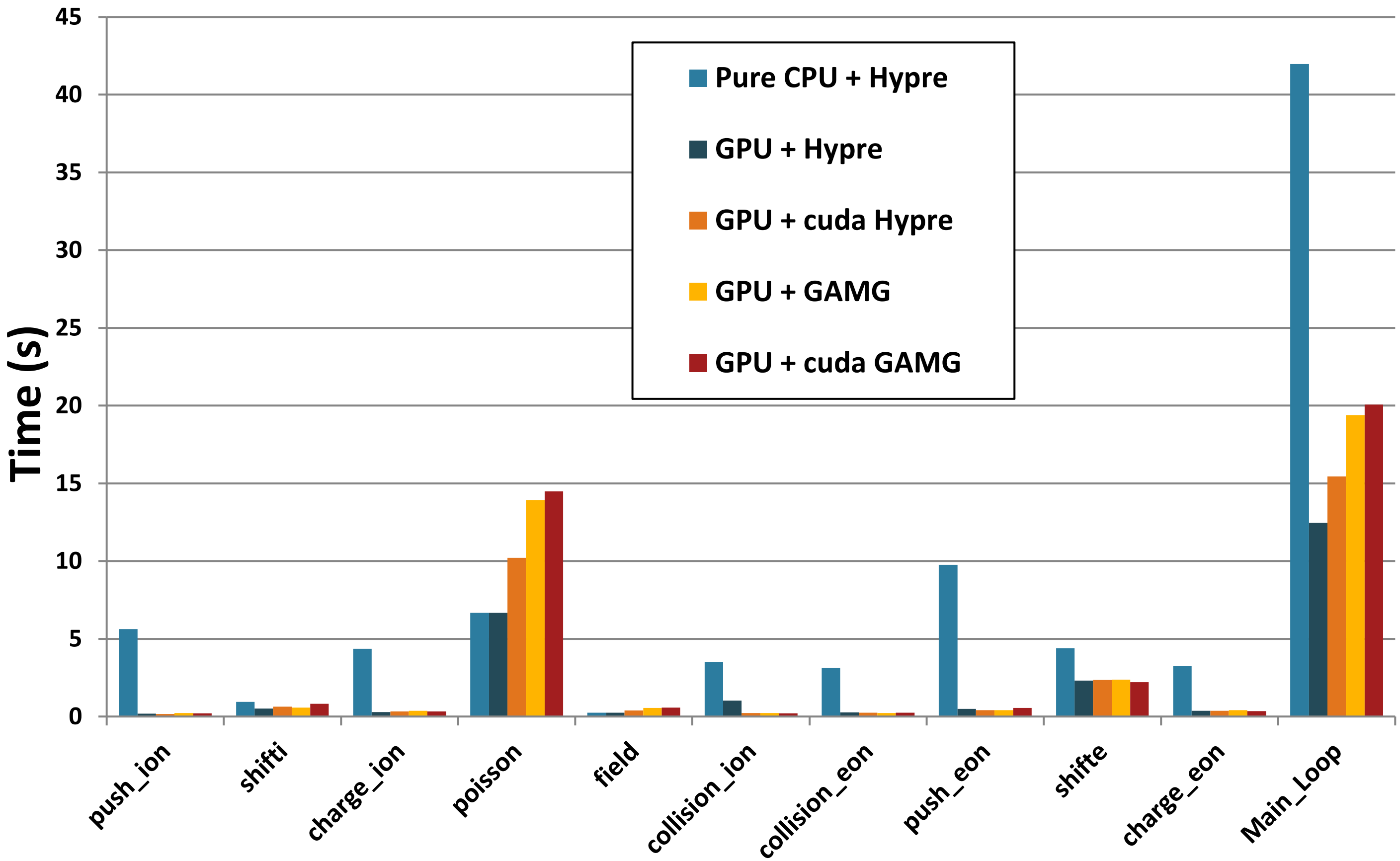
HYPRE vs GAMG in Production Run case

micell=100
mpsi=150

MGRID= 113,185*16
MISUM= 180,854,400

4 Node 128 MPI
mzetamax=16
npartdom=8

4 CPU / 1 GPU



Summary

GTS (Gyrokinetic Tokamak Simulation)

- A global gyrokinetic particle simulation code for micro-turbulence study in tokamak
- Particle-In-Cell algorithm (particles + grid-based field solve)
- Recently upgraded for physics studies associated with the thermal quench transport

Porting to GPU machine

- Attended GPU Hackathon in 2019 at Princeton University (Mentor: Rueben Budiardja (ORNL))
- OpenACC directives ported the code to GPU keeping compatibilities with CPU machine
- GTS is now running ***production simulations*** on ***Traverse*** with a significant acceleration, making efficient use of the Traverse computational resource

  - Significant speed-up **(>20x)** for the particle parts

  - Field solve part (Poisson equation) will be ported to GPU via some libraries (ex. PETSc, Hypre, AMGx)