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
  - $\delta f$ 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
The gyrokinetic equation for particle distribution in 5-dimension phase space

- $f_s$: gyro-center distribution function

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

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

- $a$ (or $\rho$): radial coordinates (a flux surface label)
- $\theta$ and $\phi$: poloidal and toroidal angle
- $v_\parallel$: parallel velocity
- $\mu = m_s v_\perp^2 / 2B$: magnetic moment
- $B^* = B + (m_s v_\parallel / e_s) b \cdot \nabla \times b$
- **Electron and Ion densities from the distribution function**

\[
\delta n_i(x) = \int \delta f_i(R, \mu, \nu) \delta(R - x + \rho_i) dR d^3v,
\]

\[
\delta n_e(x) = \int \delta f_e(R, \mu, \nu) \delta(R - x + \rho_e) dR d^3v \approx \int \delta f_e d^3v, \quad (\rho_e \to 0),
\]

GK transform $\Phi(x) \to \Phi(R, \mu)$:

\[
\Phi(R, \mu) = \frac{1}{2\pi} \int \phi(x) \delta(x - R - \rho) dx d\Theta.
\]

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

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

\[
-\nabla_\perp \cdot \frac{Z_i n_{i,0}}{B\Omega_i} \nabla_\perp \Phi = Z_i \delta \tilde{n}_i - \delta n_e \quad \text{[Dubin, et. al., Phys. Fluids 26, 3524 (1983)]}
\]
The distribution function $f_S$ is represented by Marker Particles

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

Equation of motion

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

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

$\delta f$ weight evolution equation

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

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

$$\frac{d\delta f}{dt} = -\frac{df_0}{dt} = -\dot{Z} \cdot \nabla Z f_0 = -(\dot{Z}_0 + \dot{Z}_1) \cdot \nabla Z f_0$$
Particle-In-Cell Method

Assigning fields to particles
(Grid to Particles)

Move particles to new positions
(Particles only)

Charge deposition to Grid Nodes
(Particles to Grid)

Update potential & electric fields
(Grid only)

\[
\begin{align*}
\phi_{i,j+1}^{(n+1)} &= \phi_{i,j}^{(n+1)} \\
\phi_{i-1,j}^{(n+1)} &= \phi_{i+1,j}^{(n+1)} \\
\phi_{i,j-1}^{(n+1)} &= \phi_{i,j}^{(n+1)}
\end{align*}
\]
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
    
    \[
    !$acc \ declare \ create(P_x, P_v, P_w, P_E, P_B, P_{gradB}, \ldots)\]

Assigning fields to particles (Grid to Particles)

\[
 !$acc \ parallel \ loop \ present(P_x, P_E, \ldots) \\
 \text{do } m=1, Nm \\
 \quad ! \ Find \ a \ matching \ grid \ index \\
 \quad xid = \text{floor}\left((P_x(m) - x0)/dx\right) \\
 \quad ! \ Calculate \ a \ weight \ for \ linear \ interpolation \\
 \quad wx = (P_x(m) - Gx(xid))/dx \\
 \quad ! \ Calculate \ Fields \ at \ particle \ position \\
 \quad P_E(m) = (1-wx)E(xid) + wx*E(xid+1) \\
 \quad P_B(m) = \ldots \\
 \quad \ldots \\
 \quad \ldots \\
 \quad \text{enddo}
\]
### 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
endo
```

### 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
endo
```
Gyrokinetic Poisson Equation

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

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

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

\[
- \nabla \cdot \left[ \varepsilon_0 \vec{g} + \sum_s n_s m_s \left( \vec{g} - \frac{BB}{B^2} \right) \right] \cdot \nabla \Phi = e (\delta \vec{n}_i - \delta n_e)
\]

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

\[
Ax = b
\]

- 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

Lower is better
Speed-Up Factor

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

Higher is better

- 4 CPU (mz4np1)
- 32 CPU (mz4np8)
- 4CPU + 4GPU (mz4np1)
- 32CPU + 4GPU (mz4np8)
Full Power of 1 node on Traverse

32 CPU

- charge_eon: 22%
- push_eon: 22%
- collision_eon: 7%
- collision_ion: 8%
- smooth: 10%
- shift: 2%
- charge_ion: 12%
- push_ion: 11%
- shift: 1%
- charge_eon: 9%
- check_tracers: 24%
- poisson: 24%
- field: 1%
- collision_ion: 2%
- collision_eon: 2%
- push_eon: 5%
- shifte: 4%
- charge_eon: 3%
- check_tracers: 2%
- snapshot: 0%
- diagnostics: 2%

32 CPU + 4 GPU

- charge_eon: 29%
- push_eon: 4%
- collision_eon: 0%
- collision_ion: 0%
- smooth: 4%
- field: 5%
- shift: 3%
- charge_ion: 12%
- push_ion: 11%
- shift: 1%
- charge_eon: 9%
- check_tracers: 24%
- poisson: 66%
- field: 1%
- collision_ion: 2%
- collision_eon: 2%
- push_eon: 5%
- shifte: 4%
- charge_eon: 3%
- check_tracers: 2%
- snapshot: 0%
- diagnostics: 2%
Performance Comparison
(original Poisson vs new 3D Poisson)

Elapsed time/step

- ori. Tol=1e-3
- ori. Tol=1e-5
- New Poisson
Breakdown of Elapsed Time

Ori. Poisson (Tol=1e-3)

- poisson: 44%
- diagnostics: 5%
- push_ion: 3%
- shift: 2%
- charge_ion: 4%
- snapshot: 5%
- check_tracers: 10%
- charge_eon: 4%
- shifte: 7%
- push_eon: 8%
- collision_eon: 4%
- collisionion: 3%
- field: 3%
- smooth: 1%

Ori. Poisson (Tol=1e-5)

- di: 4%
- charge_ion: 4%
- snapshot: 1%
- check_tracers: 10%
- shifte: 2%
- push_ion: 2%
- shift: 1%
- charge_eon: 4%
- push_eon: 8%
- collision_eon: 4%
- collisionion: 3%
- field: 3%
- smooth: 1%

New Poisson 3D

- 71%
- 12%
- 4%
- 3%
- 6%
- 4%
- 5%
- 9%
- 10%
- 5%
- 5%
- 4%
- 1%
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
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)