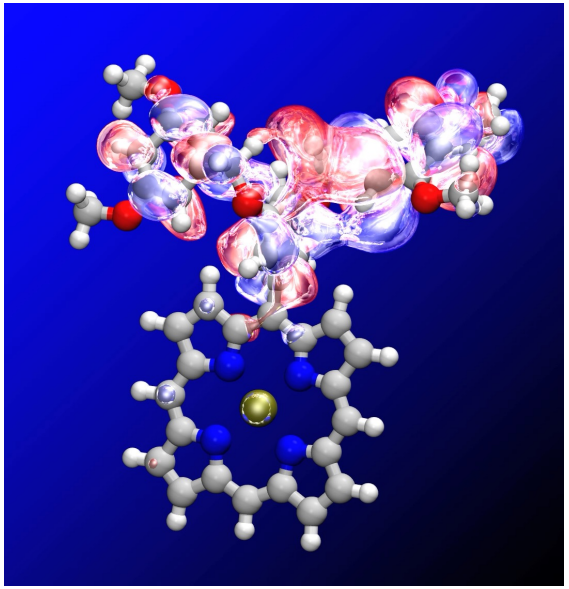
Argonne NATIONAL LABORATORY

PORTING LFD MINIAPP TO GPU VIA OPENMP OFFLOAD

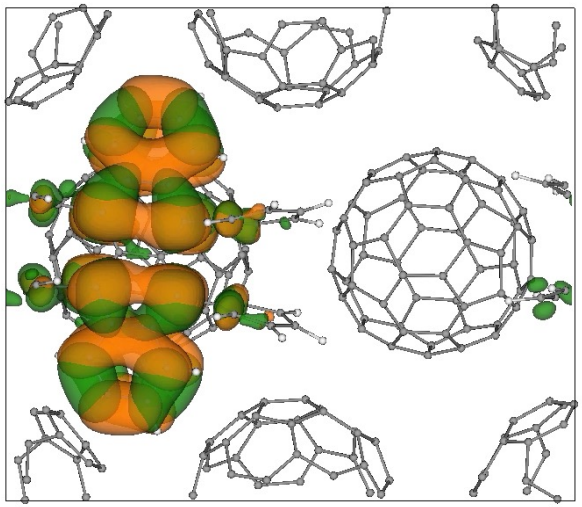
Pankaj Rajak, Ye Luo and Aiichiro Nakano
ESP Project : NAQMC_RMD_ESP
Argonne National Laboratory

QXMD : Scalable Quantum Molecular Dynamics (QMD)



Zn porphyrin

- Open source program for QMD with capabilities for nonadiabatic QMD (NAQMD) and multiscale shock
- Follow the trajectory of all atoms while computing interatomic interaction from first principles in the framework of density functional theory (DFT)



Rubrene/C_{60}

quasi-electron;
quasi-hole

- *SoftwareX 10, 100307: 1-5 (2019)*
- *Proceedings of International Conference on High Performance Computing in Asia-Pacific Region, HPCAsia2020 best paper award, 1-10 (2020)*

LFD Miniapp for QXMD

- **Local field dynamics (LFD) : key computational kernel of NAQMD**
- **LFD solves many-electron dynamics in the framework of real-time (RT) time-dependent density functional theory**
- **input to LDF is potential field, and it return the electron density**

```
Total walltime       = 314.458 (s)
Electron-propagation = 92.8069 (s)
Field-propagation    = 208.392 (s)
calc_energy function = 26.7224 (s)
```

Most expensive functions : electron and field dynamics solvers

- **Developed LFD mini-app in C++ for GPU offloading and integration with QXMD**

Time spent in various functions on a test system

```
========= Time spent in initilization:
  init_param           = 0.000279768 (s)
  init_wfnvloc         = 0.228392 (s)
  init_field           = 0.00852508 (s)
========= Time spent in indivudial functions:
  kin_prop             = 89.1699 (s)
  pot_prop             = 3.402 (s)
  set_prop             = 0.0519754 (s)
  compute_v            = 0.00362018 (s)
  compute_vxc          = 0.563744 (s)
  compute_rho          = 5.44145 (s)
  periodic_bc          = 0.234753 (s)
  spectral_field_solve = 0.0048161 (s)
Electron-propagation time ~  Kin_prop + pot_prop + periodic_bc
```

Electron filed solver: Kin_prop( )

```
void kin_prop (int d, int p) {
float wrk[Nx+2][Ny+2][Nz+2][2], w[2];
for (int n=0; n < Norb; n++) {
  for (int i=1; I <= Nr[0]; i++)
    for (int j=1; j <= Nr[1]; j++)
        for (int k=1; k <= Nr[2]; k++) {
            w[0]  = al[d][p][0]*psi[n][i][j][k][0] - al[d][p][1]*psi[n][i][j][k][1] ;
            w[1]  = al[d][p][0]*psi[n][i][j][k][1] + al[d][p][1]*psi[n][i][j][k][0] ;
             ...
            for (int s=0; s<2; s++) wrk[i][j][k][s] = w[s] ;
        }
    # update psi[n][i][j][k][s] ← wrk[i][j][k][s]
} }
```

- Inefficient Memory usage & loop structure
- By loop reordering, we can get rid of wrk
- al doesn't depend on n, j and k. (can be cached)

Electron filed solver: Kin_prop( ) Update-1

```
void kin_prop (int d, int p) {
float w[2];
for (int j=1; j < Nr[1]; j++)
  for (int k=1; I <= Nr[2]; k++)
    for (int i=1; j <= Nr[0]; i++)
      for (int n=0; n < Norb; n++) {
        w[0]  = al_0*psi[i][j][k][n][0] – al_1*psi[i][j][k][n][1] ;
        w[1]  = al_1*psi[i][j][k][n][1] + al_0*psi[i][j][k][n][0] ;
        ...
        # update psi[n][i][j][k][s] ← w[s]
      }
}
```

- Inefficient Memory usage & loop structure
- By loop reordering, we can get rid of wrk
- al doesn't depend on n, j and k. (can be cached)

- Better memory usage and data locality by changing data layout

psi[n,i,j,k,s]→ psi[i,j,k,n,s]

Electron filed solver: Kin_prop( ) Update-2

```
void kin_prop (int d, int p) {
for (int j=1; j < Nr[1]; j++)
  for (int k=1; I <= Nr[2]; k++)
    for (int i=1; j <= Nr[0]; i++)
      for (int n=0; n < Norb; n++) {
          w[0]  = al_0*psi[i][j][k][n][0] – al_1*psi[i][j][k][n][1] ;
          w[1]  = al_1*psi[i][j][k][n][1] + al_0*psi[i][j][k][n][0] ;
          w[0]  += bl_0[i]*psi[i-i][j][k][n][0] – bl_1[i]*psi[i-i][j][k][n][1] ;
          w[1]  += bl_0[i]*psi[i-i][j][k][n][1] – bl_1[i]*psi[i-i][j][k][n][0] ;
          …
          # update psi[n][i][j][k][s] ← w[s]
      }
}
```

- use old psi

Electron filed solver: Kin_prop( ) Update-2

```
void kin_prop (int d, int p) {
for (int j=1; j < Nr[1]; j++)
  for (int k=1; I <= Nr[2]; k++) {
   for (int n=0; n <= Norb; n++) {
        psi_old0[n] = psi[0][j][k][n][0];
        psi_old1[n] = psi[0][j][k][n][1];  }
    for (int i=1; j <= Nr[0]; i++)
    for (int n=0; n < Norb; n++) {
          w[0]  = al_0*psi[i][j][k][n][0] – al_1*psi[i][j][k][n][1] ;
          w[1]  = al_1*psi[i][j][k][n][1] + al_0*psi[i][j][k][n][0] ;
          w[0]  += bl_0[i]*psi_old0[n] – bl_1[i]*psi_old1[n] ;
          w[1]  += bl_0[i]*psi_old1[n] – bl_1[i]*psi_old0[n] ;
          …
          # update psi_old0 ← psi[n][i][j][k][0] and psi_old1 ← psi[n][i][j][k][s]
          # update psi[n][i][j][k][s] ← w[s]
       }
} }
```

- copy old psi to reduce scratch space
- Involves complex operation
- convert psi, psi_old into 1D complex variable

Electron filed solver: Kin_prop( ) Update-3

```
void kin_prop (int d, int p) {
for (int j=1; j < Nr[1]; j++)
  for (int k=1; l <= Nr[2]; k++) {
    for (int n=0 ; n <  Norb; n++)
        psi_old[i] = psi[yz_stride+n];
        for (int i=1; j <= Nr[0]; i++)
        for (int n=0; n < Norb; n++) {
            w  = al*psi[stride+n] + bl[i]*psi_old[n] + …;
           # update psi_old0[n] ← psi[stride+n]
           # update psi[stride+n]← w
        }
  }
}
```

- Involves complex operation
- convert psi, psi_old into 1D complex variable

Before

std:: float psi[Nx+2][Ny+2][Nz+2][Norb]
std:: float psi_old[Ny+2][Nz+2][Norb]

After

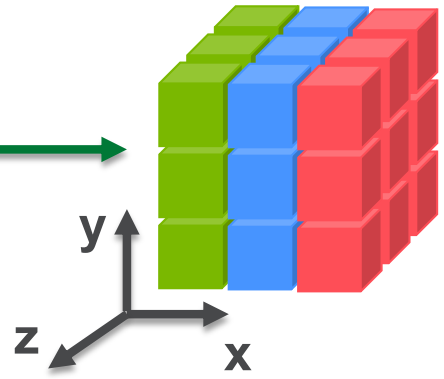
std::complex<float> psi
std::complex<float> psi_old

Electron filed solver: Kin_prop( ) Offload

```
void kin_prop (int d, int p) {
#pragma omp team distribute  collapse(2)
for (int j=1; j < Nr[1]; j++)
  for (int k=1; I <= Nr[2]; k++) {
   #pragma omp parallel for simd nowait
    for (int n=0 ; n <  Norb; n++)
         psi_old[i] = psi[yz_stride+n];
         for (int i=1; j <= Nr[0]; i++)
        #pragma omp parallel for simd nowait
        for (int n=0; n < Norb; n++) {
             w  = al*psi[stride+n] + bl[i]*psi_old[n] + ...;
            # update psi_old0[n] ← psi[stride+n]
            # update psi[stride+n]← w
        }
 } }
```

Hierarchical parallelism

- Coarse grain parallelism via *omp team distribute* on outer loops
- Fine grain parallelism on inner Norb loop by omp parallel for
- Typical size of Nr is 256 and Norb 100



Electron filed solver: Kin_prop( ) Offload Timing

```
void kin_prop (int d, int p) {
#pragma omp team distribute  collapse(2)
for (int j=1; j < Nr[1]; j++)
  for (int k=1; I <= Nr[2]; k++) {
   #pragma omp parallel for simd nowait
    for (int n=0 ; n <  Norb; n++)
         psi_old[i] = psi[yz_stride+n];
         for (int i=1; j <= Nr[0]; i++)
        #pragma omp parallel for simd nowait
         for (int n=0; n < Norb; n++) {
              w  = al*psi[stride+n] + bl[i]*psi_old[n] + ...;
             # update psi_old0[n] ← psi[stride+n]
             # update psi[stride+n]← w
         }
  } }
```

Updated timing

```
Total wall time            = 208.29 (s)
Electron-propagation time  = 1.44 (s)
Field-propagation time     = 206.08 (s)
calc_energy function time  = 18.81 (s)
```

Original

```
Total walltime       = 314.458 (s)
Electron-propagation = 92.8069 (s)
Field-propagation    = 208.392 (s)
calc_energy function = 26.7224 (s)
```

➢ ***Using xlr complier***

Field Dynamics Solver: Field_prop ()

```
void field_prop () {
  for (int i=1; i<=Nr[0]; i++)
    for (int j=1; j<=Nr[1]; j++)
      for (int k=1; k<=Nr[2]; k++)
        vH[2][i][j][k] = fx*vH[0][i-1][j][k] +…. + rho*rho[i][j][k]
  …
  for (int i=1; i<=Nr[0]; i++)
    for (int j=1; j<=Nr[1]; j++)
      for (int k=1; k<=Nr[2]; k++){
        vH[1][i][j][k] += vH[2][i][j][k];
        vH[0][i][j][k] += vH[1][i][j][k];  }
  …
}
```

- Contains multiple loops which updates the individual point of the 4D vH[3][Nx][Ny][Nz] grid

Field Dynamics Solver: Field_prop () Offload

```
void field_prop () {
 #pragma omp target teams distribute parallel for simd collapse(3)
  for (int i=1; i<=Nr[0]; i++)
    for (int j=1; j<=Nr[1]; j++)
      for (int k=1; k<=Nr[2]; k++)
          vH[2*dim_stride+ stride] = fx*vH[offset_rho-xyz_stride] +…
…
 #pragma omp target teams distribute parallel for simd collapse(3)
  for (int i=1; i<=Nr[0]; i++)
    for (int j=1; j<=Nr[1]; j++)
      for (int k=1; k<=Nr[2]; k++){
          vH[1*dim_stride+ stride] += vH[2*dim_stride+ stride];
          vH[0*dim_stride+ stride] += vH[1*dim_stride+ stride];}
 …
}
```

- Convert vH[3][Nx][Ny][Nz] into 1D dynamic array vH
- Allocate vH on device to minimize data movement between host and device
- Flat parallelism for omp offload

Field_prop () Offload Timing

```
void field_prop () {
 #pragma omp target teams distribute parallel for simd collapse(3)
  for (int i=1; i<=Nr[0]; i++)
    for (int j=1; j<=Nr[1]; j++)
      for (int k=1; k<=Nr[2]; k++)
          vH[2*dim_stride+ stride] = fx*vH[offset_rho-xyz_stride] +…
…
 #pragma omp target teams distribute parallel for simd collapse(3)
  for (int i=1; i<=Nr[0]; i++)
    for (int j=1; j<=Nr[1]; j++)
      for (int k=1; k<=Nr[2]; k++){
          vH[1*dim_stride+ stride] += vH[2*dim_stride+ stride];
          vH[0*dim_stride+ stride] += vH[1*dim_stride+ stride];}
 …
}
```

Updated timing

```
Total walltime       = 113.608 (s)
Electron-propagation = 1.55114 (s)
Field-propagation    = 111.271 (s)
```

Original

```
Total walltime       = 314.458 (s)
Electron-propagation = 92.8069 (s)
Field-propagation    = 208.392 (s)
calc_energy function = 26.7224 (s)
```

GPU Activity time of field_prop

```
33.77%  9.84747s   1100000
19.38%  5.65139s   1100000
19.38%  5.65079s   1100000
19.34%  5.64034s   1100000
```

Field_prop () Asynchronous Offload

```
void field_prop () {
 #pragma omp target teams distribute parallel for simd collapse(3) nowait depend(inout:vH_ptr)
  for (int i=1; i<=Nr[0]; i++)
    for (int j=1; j<=Nr[1]; j++)
      for (int k=1; k<=Nr[2]; k++)
          vH[2*dim_stride+ stride] = fx*vH[offset_rho-xyz_stride] +…
 …
 #pragma omp target teams distribute parallel for simd collapse(3) nowait depend(inout:vH_ptr)
  for (int i=1; i<=Nr[0]; i++)
    for (int j=1; j<=Nr[1]; j++)
      for (int k=1; k<=Nr[2]; k++){
          vH[1*dim_stride+ stride] += vH[2*dim_stride+ stride];
          vH[0*dim_stride+ stride] += vH[1*dim_stride+ stride];}
  …
}
```

Field_prop Timing

Original = 208.392 s
Sync. = 111.271 s
Async. = 26.722 s

Benchmark Results

System Size: Nx=Ny=Nz=32, Norb=64,Unit-cell (1,2,1)

Branch	Electron-Propagation		Field-propagation		Total time	
	Xlr_c	ifx	Xlr_c	ifx	Xlr_c	ifx
Master	92.8069 s	8.841 s	208.39 s	41.88 s	314.45 s	51.52 s
Kin_offload	1.4400 s	2.06 s	206.08 s	28.03 s	208.29 s	31.17 s
Kin_Field_sync_offload	1.5511 s	Error	111.27 s	Error	113.60 s	Error
Kin_Field_async_offload	0.8317 s	-	27.46 s	-	29.03 s	-

Summary

1) System Size: Nx=Ny=Nz=32, Norb=32,Unit-cell (1,1,1)

Branch	Electron-propagation (s)	Field-propagation (s)	Total Time (s)
Master	46.7649	42.0905	95.7997
Kin_offload	0.79	41.72	43.23
Kin_Field_sync_offload	0.752694	22.3408	23.8729
Kin_Field_async_offload	0.449227	5.56158	6.74293

2) System Size: Nx=Ny=Nz=32, Norb=64,Unit-cell (1,2,1)

Branch	Electron-propagation (s)	Field-propagation (s)	Total Time (s)
Master	92.8069	208.392	314.458
Kin_offload	1.44	206.08	208.29
Kin_Field_sync_offload	1.55114	111.271	113.608
Kin_Field_async_offload	0.831752	27.457	29.0229