Porting NAQMD Kernels to GPU via OpenMP Offload

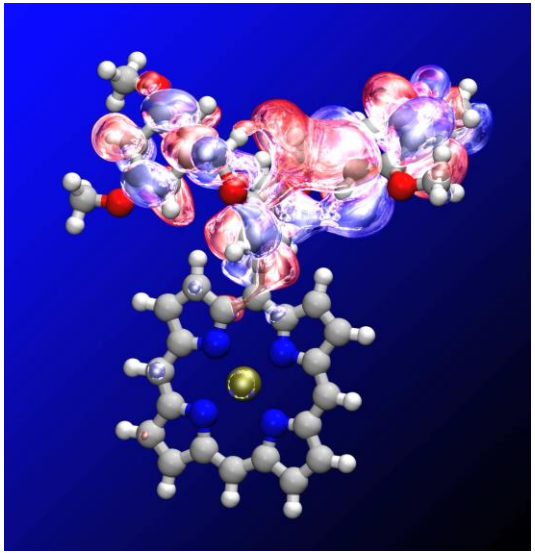
Pankaj Rajak, Ye Luo, Ken-ichi Namura and Aiichiro Nakano
Argonne National Laboratory

Intel eXtreme Performance User Group (IXPUG) Annual Meeting

OUTLINE

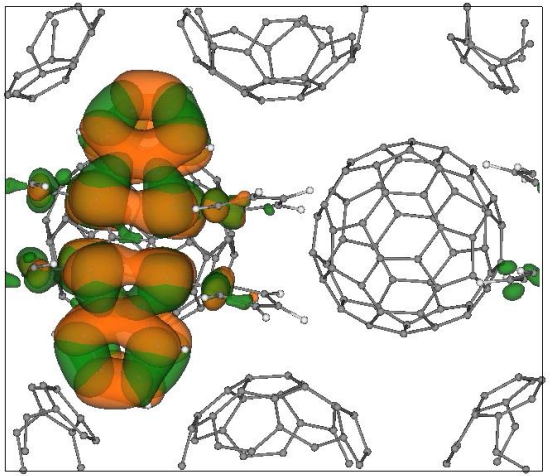
- Nonadiabatic Quantum Molecular Dynamics (NAQMD)
- Code optimization and GPU offload of NAQMD kernels via OpenMP offload on IBM+NVIDIA
- Performance of the NAQMD kernels on Intel-Gen9 GPU

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 NAQMD

- **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 LFD is potential field, and it return the electron density**

Time spent in various functions on a test system

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

```
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

Electron field 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][:][:][:][:] ← wrk[:][:][:][:]
} }
```

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

Electron field 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_0*psi[n][i][j][k][0] – al_1*psi[n][i][j][k][1] ;
            w[1]  = al_1*psi[n][i][j][k][1] + al_0*psi[n][i][j][k][0] ;
             …
            for (int s=0; s<2; s++) wrk[i][j][k][s] = w[s] ;
        }
     # update psi[n][:][:][:][:] ← wrk[:][:][:][:]
} }
```

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

Electron field solver: Kin_prop( ) Update-1

```
void kin_prop (int d, int p) {
float wrk[Nx+2][Norb][2], w[2];
for (int j=1; j <= Ny; j++)
  for (int k=1; k <= Nz; k++) {
    for (int i=1; i <= Nx; 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] ;
             …
             wrk[i][n][:] = w[:];
         }
     # update psi[:][j][k][:][:] ← wrk[:][:][:]
  } }
```

- Inefficient Memory access & loop structure
- By loop reordering, we can get rid of wrk

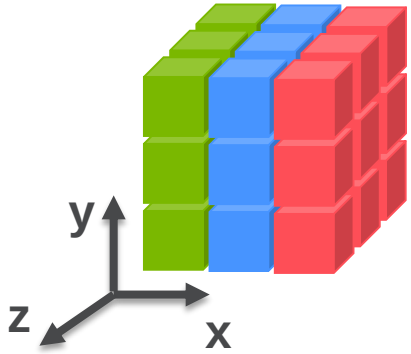
- Better memory usage, data locality and vectorization by changing data layout

psi[n,i,j,k,s]→ psi[i,j,k,n,s]

Electron field solver: Kin_prop( ) Update-2

```
void kin_prop (int d, int p) {
for (int j=1; j <= Ny; j++)
  for (int k=1; k <= Nz; k++) {
    for (int i=1; i <= Nx; 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-1][j][k][n][0] – bl_1[i]*psi[i-1][j][k][n][1] ;
            w[1]  += bl_0[i]*psi[i-1][j][k][n][1] – bl_1[i]*psi[i-1][j][k][n][0] ;
             …                                                  use old psi
        }
    # update psi[:][j][k][:][:] ← wrk[:][:][:]
} }
```

- copy psi[i-1] to psi_old
- No need for temporary variable wrk



Electron field solver: Kin_prop( ) Update-3

```
void kin_prop (int d, int p) {
for (int j=1; j <= Ny; j++)
  for (int k=1; k <= Nz; k++) {
    for (int n=0 ; n <  Norb; n++)
        psi_old[n] = psi[yz_stride+n];
    for (int i=1; i <= Nx; 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 vector for GPU offload

Before

float psi[Nx+2][Ny+2][Nz+2][Norb][2]
float psi_old0[Norb],psi_old1[Norb]

After

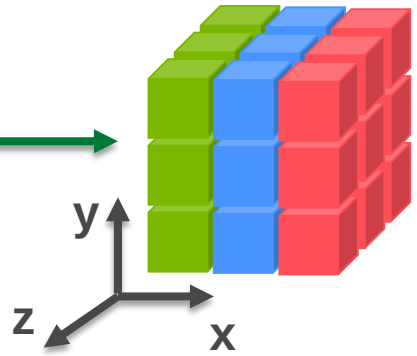
vector<complex<float>> psi
vector<complex<float>> psi_old

Electron field solver: Kin_prop( ) Offload

```
void kin_prop (int d, int p) {
#pragma omp team distribute  collapse(2)
for (int j=1; j <= Ny; j++)
  for (int k=1; k <= Nx; k++) {
    #pragma omp parallel for simd
     for (int n=0 ; n <  Norb; n++)
          psi_old[i] = psi[yz_stride+n];
     for (int i=1; i <= Nx; i++)
          #pragma omp parallel for simd
          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 field solver: Kin_prop( ) Offload Timing

```
void kin_prop (int d, int p) {
#pragma omp team distribute  collapse(2)
for (int j=1; j <= Ny; j++)
  for (int k=1; k <= Nz; k++) {
     #pragma omp parallel for simd
      for (int n=0 ; n <  Norb; n++)
          psi_old[i] = psi[yz_stride+n];
      for (int i=1; i <= Nx; i++)
          #pragma omp parallel for simd
           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
          }
  } }
```

Original

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

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)
```

Field Dynamics Solver: Field_prop () Offload

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

- Contains multiple loops which updates the individual point of the 4D vH[3][Nx][Ny][Nz] grid
- Pre-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<=Nx; i++)
    for (int j=1; j<=Ny; j++)
      for (int k=1; k<=Nz; k++)
          vH[2*dim_stride+ offset] = fx*vH[offset_rho-xyz_stride] +…
…
 #pragma omp target teams distribute parallel for simd collapse(3)
  for (int i=1; i<=Nx; i++)
    for (int j=1; j<=Ny; j++)
      for (int k=1; k<=Nz; k++){
          vH[1*dim_stride+ offset] += vH[2*dim_stride+ offset];
          vH[0*dim_stride+ offset] += vH[1*dim_stride+ offset];}
 …
}
```

Original

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

Updated timing

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

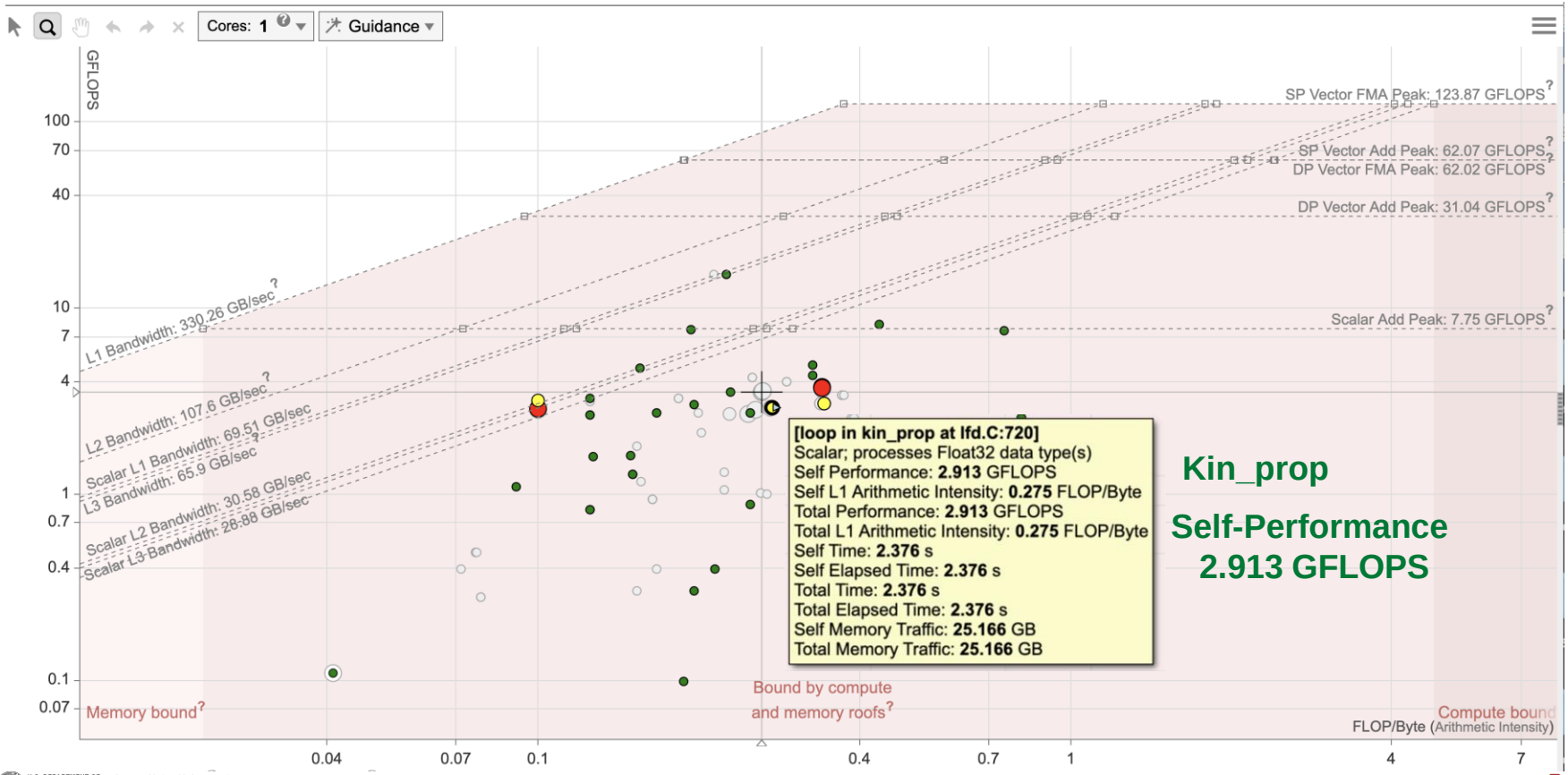
```
void field_prop () {
 #pragma omp target teams distribute parallel for simd collapse(3) nowait depend(inout:vH_ptr)
  for (int i=1; i<=Nx; i++)
    for (int j=1; j<=Ny; j++)
      for (int k=1; k<=Nz; k++)
          vH[2*dim_stride+ offset] = fx*vH[offset_rho-xyz_stride] +…
…
 #pragma omp target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

Field_prop Timing

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

Intel Advisor Roofline Analysis on Intel Gen9 GPU

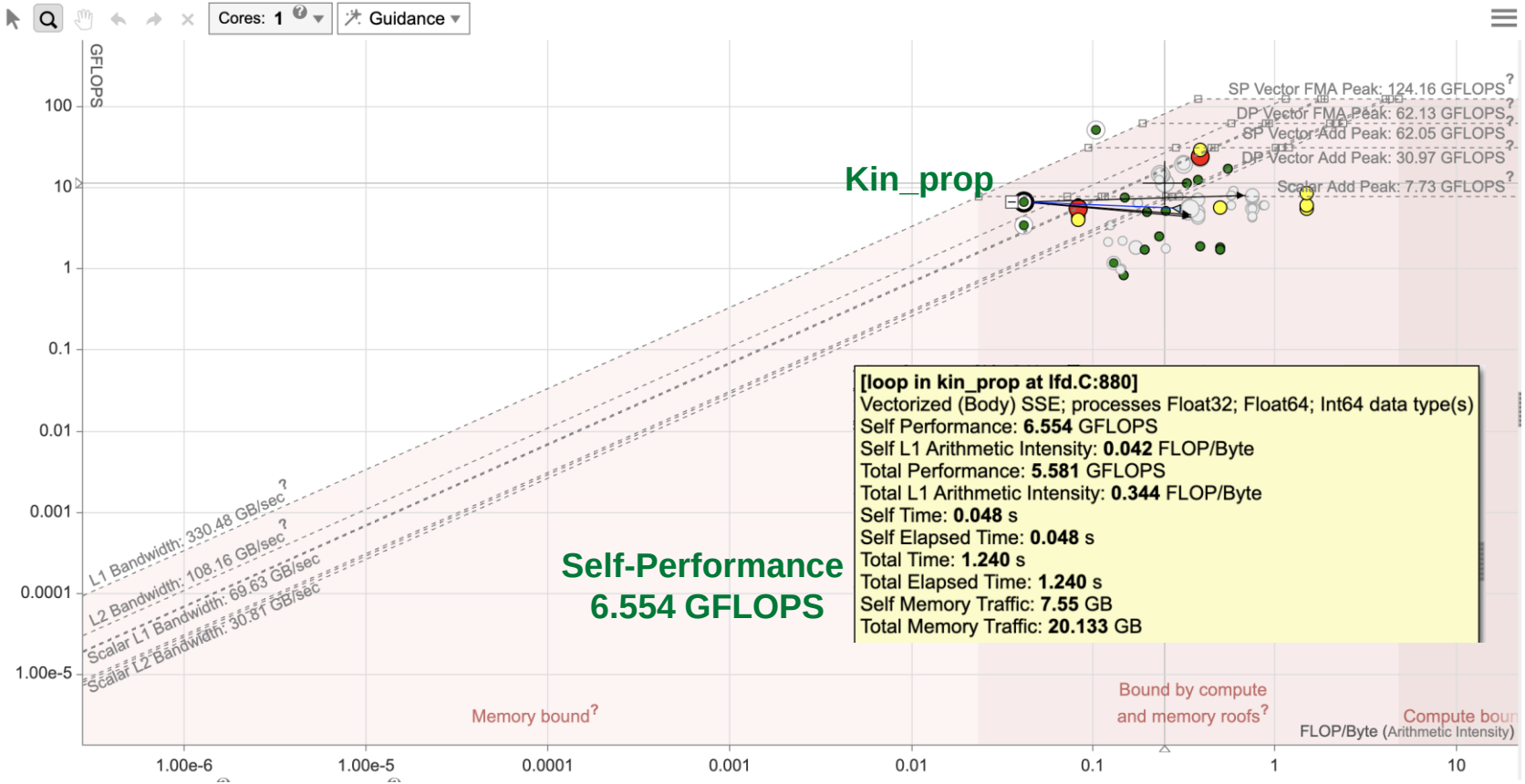
Roofline Analysis of LFD C++ Original Code



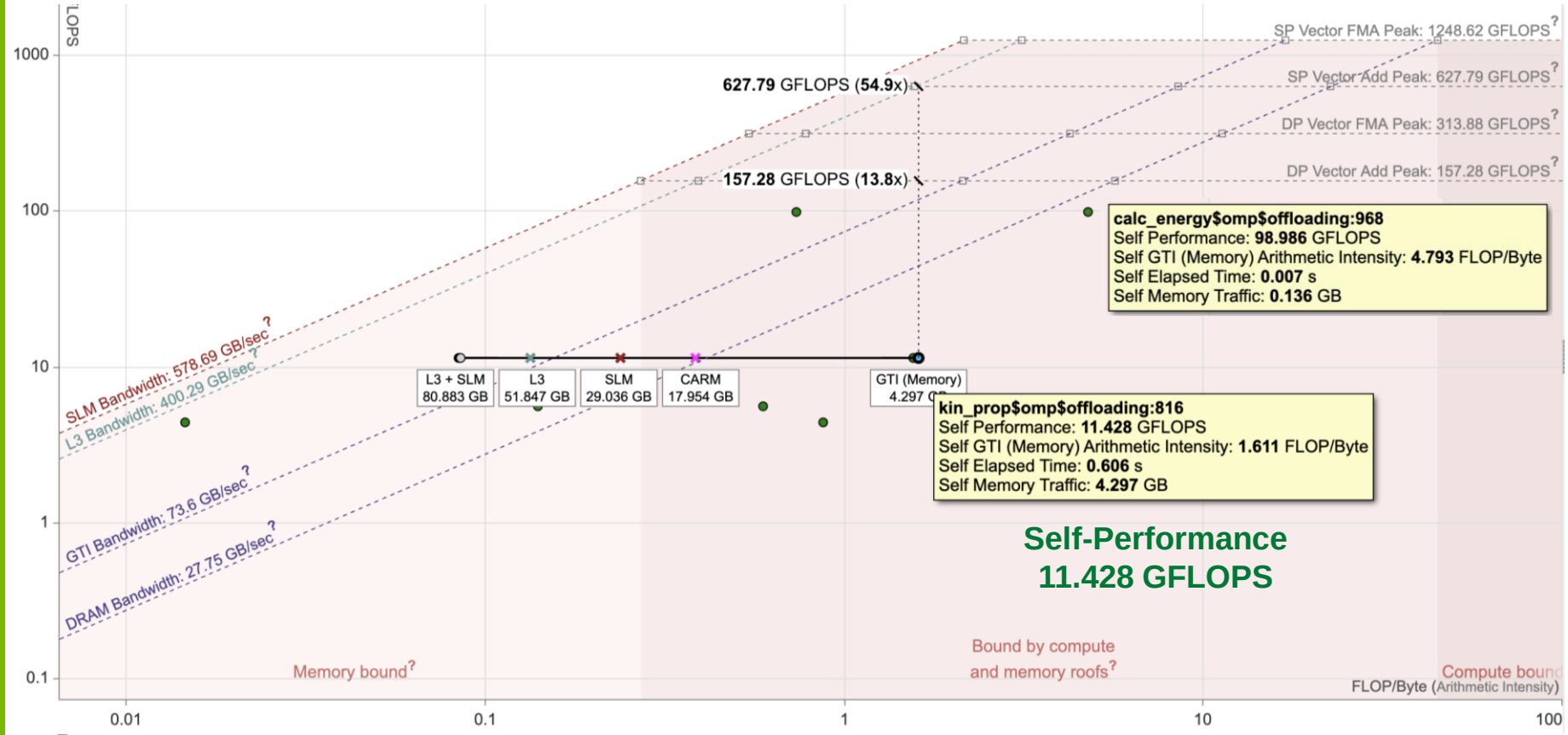
Kin_prop

Self-Performance 2.913 GFLOPS

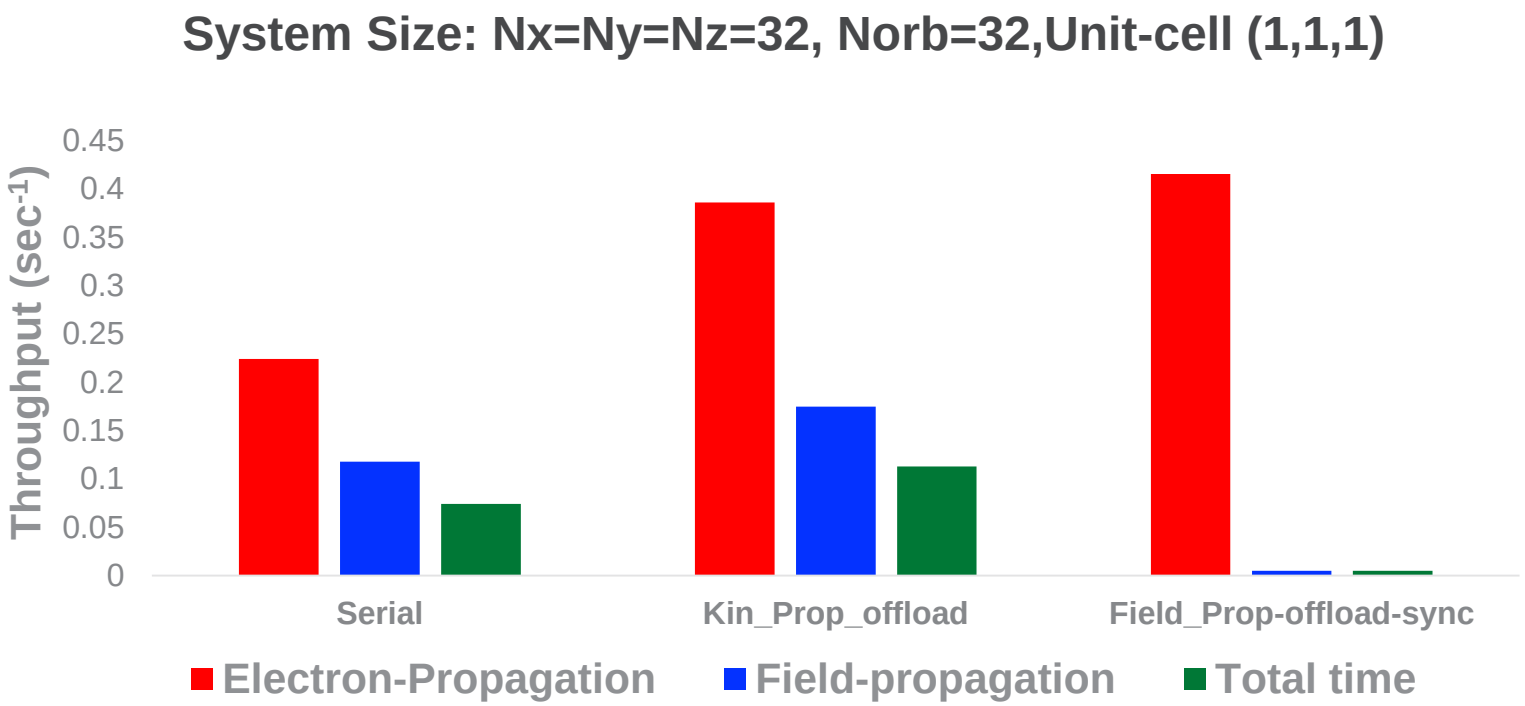
Roofline Analysis of LFD C++ Updated Code



Roofline Analysis of LFD C++ Offload (Kin_prop)



Benchmark Results on Intel Gen9-GPU



Summary

- Refactor the CPU code with vector friendly algorithm
- Change data layout for better vectorization and data locality
- GPU offload by hierarchical or flat parallelism
- Reduce data movement between device and host
- Leverage asynchronous offload to reduce runtime cost