2020 IXPUG ANNUAL MEETING

Argonne NATIONAL LABORATORY

A GEOMETRIC MULTIGRID METHOD KERNEL ON INTEL GPU W/ PERFORMANCE PORTABLE PROGRAMMING MODELS

JAEHYUK KWACK
ALCF Perf. Engr. Group
Argonne National Laboratory

October 14, 2020

GMG METHODS AND HPGMG [1]

- Geometric Multigrid solves partial differential equations (Lx = f) using a hierarchical approach
  - Coarse grids are constructed from fine grids by agglomeration of fine grid elements/cells
  - Provides O(N) computational complexity where N is number of unknowns.

- HPGMG: a HPC benchmark for full multigrid (FMG) algorithms
  - HPGMG-FE(Finite Element): compute-intensive and cache-intensive
  - HPGMG-FV(Finite Volume): memory bandwidth-intensive
    - Solving an elliptic problem on isotropic Cartesian grids with 4th order accuracy
    - 4× FP ops, 3× MPI messages, 2× MPI message size w/o DRAM data movement compared to 2th order HPGMG-FV
    - Employing the Full Multi-grid (FMG) F-cycle
    - A series of progressively deeper geometric multi-grid V-cycles

MG V-CYCLE

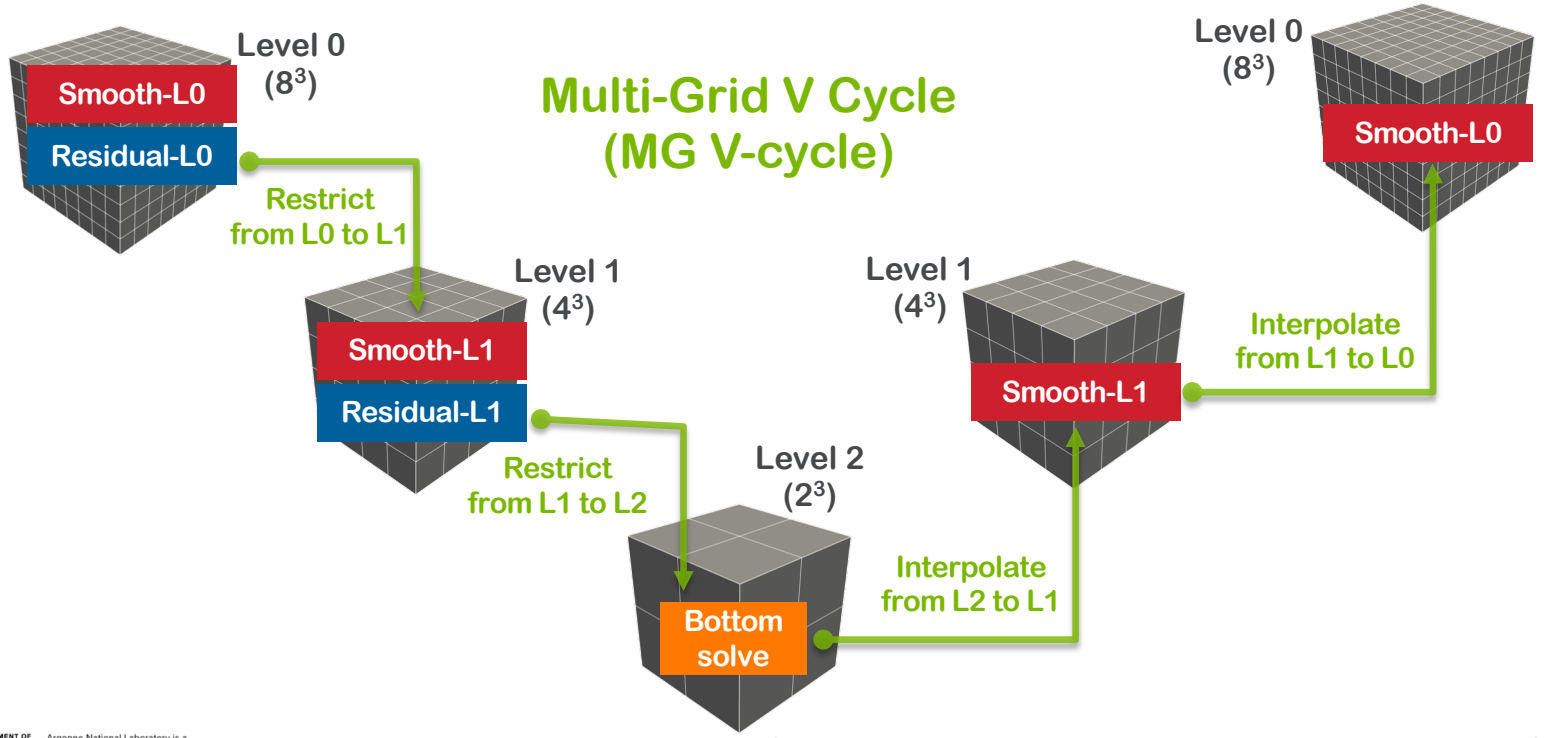
Multi-Grid V Cycle

Restrict from L1 to L2: $f^{L2} = restrict(r^{L1})$

Interpolate from L2 to L1: $u^{L1} += interpolate(u^{L2})$

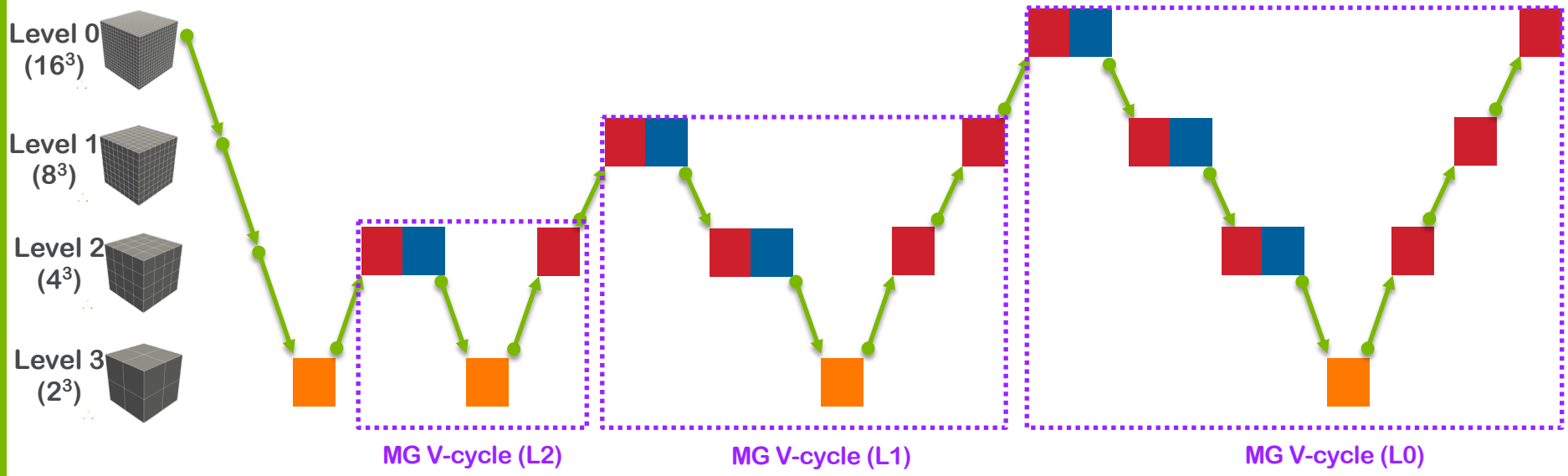
Smooth at L1: $Lu^{L1} = f^{L1}$

Residual at L1: $r^{L1} = f^{L1} - Lu^{L1}$



FMG CYCLE

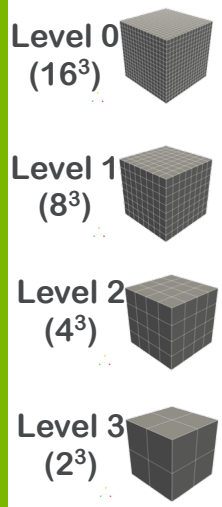
Full Geometric Multigrid cycle

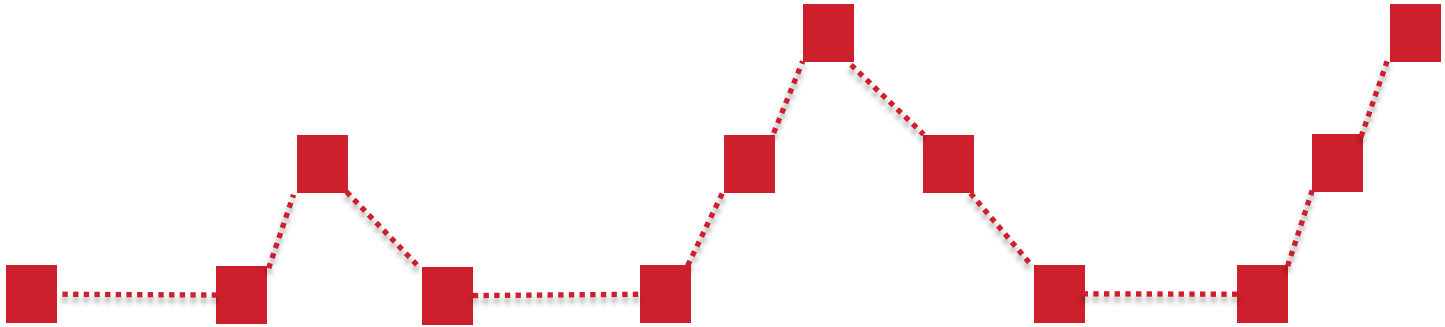


Smooth kernel: one of the most expensive kernels
e.g., for 9 levels (512^3), smooth kernel time / Total = 73.6%

FMG SMOOTH MINI-APP

Extracted the FMG cycle of HPGMG-FV





FMG smooth mini-app

Initialization
(e.g., grids for levels, memory allocations, a series of levels for smooth in a FMG cycle, and so on)

Loop for iterations (to measure sustained performance)

Loop for a series of levels for smooth in a FMG cycle (e.g., L2-L2-L1-L2-L2-L1-L0-L1-L2-L2-L1-L0 for 4 levels)

Loop for GSRB smooth (i.e. RBRBRB)

Finalization
(e.g., reporting smooth kernel time per level, memory deallocation, and so on)

```
int block,s;
for(s=0;s<2*NUM_SMOOTHS;s++){ // there are two sweeps per GSRB smooth
   double _timeStart = getTime();

   // loop over all block/tiles this process owns...
   for(block=0;block<level->num_my_blocks;block++){
      const int ilo = ilo_levels[ilevel][block];
      const int jlo = jlo_levels[ilevel][block];
      const int klo = klo_levels[ilevel][block];
      const int ihi = ihi_levels[ilevel][block];
      const int jhi = jhi_levels[ilevel][block];
      const int khi = khi_levels[ilevel][block];

      int i,j,k;
      const double h2inv = h2inv_levels[ilevel];
      const int jStride = jStride_levels[ilevel];
      const int kStride = kStride_levels[ilevel];
      // is element 000 red or black on *THIS* sweep
      const int color000 = (lowi_levels[ilevel]^lowj_levels[ilevel]^lowk_levels[ilevel]^s)&1;

      const double * __restrict__ rhs   = vectors[       rhs_id] + ghosts*(1+jStride+kStride);
      const double * __restrict__ beta_i= vectors[VECTOR_BETA_I] + ghosts*(1+jStride+kStride);
      const double * __restrict__ beta_j= vectors[VECTOR_BETA_J] + ghosts*(1+jStride+kStride);
      const double * __restrict__ beta_k= vectors[VECTOR_BETA_K] + ghosts*(1+jStride+kStride);
      const double * __restrict__ Dinv  = vectors[VECTOR_DINV  ] + ghosts*(1+jStride+kStride);
      const double * __restrict__ x_n;
      double * __restrict__ x_np1;
      if((s&1)==0){x_n      = vectors[         x_id] + ghosts*(1+jStride+kStride);
                   x_np1    = vectors[VECTOR_TEMP  ] + ghosts*(1+jStride+kStride);}
      else{        x_n      = vectors[VECTOR_TEMP  ] + ghosts*(1+jStride+kStride);
                   x_np1    = vectors[         x_id] + ghosts*(1+jStride+kStride);}

      for(k=klo;k<khi;k++){
      for(j=jlo;j<jhi;j++){
      // out-of-place must copy old value...
         for(i=ilo;i<ihi;i++){
            int ijk = i + j*jStride + k*kStride;
            x_np1[ijk] = x_n[ijk];
         }
         for(i=ilo+((ilo^j^k^color000)&1);i<ihi;i+=2){ // stride-2 GSRB
            int ijk = i + j*jStride + k*kStride;
            double Ax     = apply_op_ijk(x_n);
            x_np1[ijk] = x_n[ijk] + Dinv[ijk]*(rhs[ijk]-Ax);
         }
      }}

   } // boxes

   level->timers.smooth += (double)(getTime()-_timeStart);

} // s-loop
```

Loop for GSRB smooth (i.e. s from 0 to 6)

Loop for blocks (i.e. block from 0 to num_my_blocks)

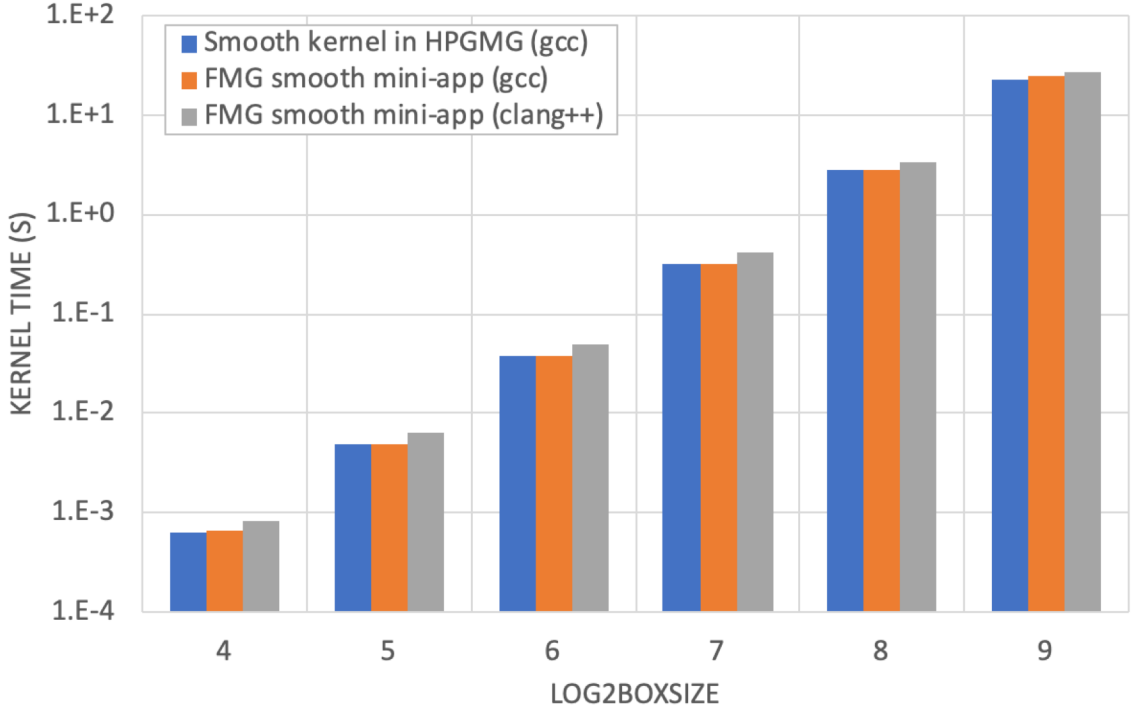
Loop for cells in the block (i.e. k,j,i from lo to hi)

```
#define apply_op_ijk(x)                                                                                                                                \
(                                                                                                                                                  \
 -b*h2inv*(                                                                                                                                        \
    STENCIL_TWELFTH*(                                                                                                                              \
      + beta_i[ijk        ]*( 15.0*(x[ijk-1      ]-x[ijk]) - (x[ijk-2        ]-x[ijk+1      ]) )                                                   \
      + beta_i[ijk+1      ]*( 15.0*(x[ijk+1      ]-x[ijk]) - (x[ijk+2        ]-x[ijk-1      ]) )                                                   \
      + beta_j[ijk        ]*( 15.0*(x[ijk-jStride]-x[ijk]) - (x[ijk-2*jStride]-x[ijk+jStride]) )                                                   \
      + beta_j[ijk+jStride]*( 15.0*(x[ijk+jStride]-x[ijk]) - (x[ijk+2*jStride]-x[ijk-jStride]) )                                                   \
      + beta_k[ijk        ]*( 15.0*(x[ijk-kStride]-x[ijk]) - (x[ijk-2*kStride]-x[ijk+kStride]) )                                                   \
      + beta_k[ijk+kStride]*( 15.0*(x[ijk+kStride]-x[ijk]) - (x[ijk+2*kStride]-x[ijk-kStride]) )                                                   \
    )                                                                                                                                              \
    + 0.25*STENCIL_TWELFTH*(                                                                                                                       \
      + (beta_i[ijk        +jStride]-beta_i[ijk        -jStride]) * (x[ijk-1      +jStride]-x[ijk+jStride]-x[ijk-1      -jStride]+x[ijk-jStride])  \
      + (beta_i[ijk        +kStride]-beta_i[ijk        -kStride]) * (x[ijk-1      +kStride]-x[ijk+kStride]-x[ijk-1      -kStride]+x[ijk-kStride])  \
      + (beta_j[ijk        +1      ]-beta_j[ijk        -1      ]) * (x[ijk-jStride+1      ]-x[ijk+1      ]-x[ijk-jStride-1      ]+x[ijk-1      ])  \
      + (beta_j[ijk        +kStride]-beta_j[ijk        -kStride]) * (x[ijk-jStride+kStride]-x[ijk+kStride]-x[ijk-jStride-kStride]+x[ijk-kStride])  \
      + (beta_k[ijk        +1      ]-beta_k[ijk        -1      ]) * (x[ijk-kStride+1      ]-x[ijk+1      ]-x[ijk-kStride-1      ]+x[ijk-1      ])  \
      + (beta_k[ijk        +jStride]-beta_k[ijk        -jStride]) * (x[ijk-kStride+jStride]-x[ijk+jStride]-x[ijk-kStride-jStride]+x[ijk-jStride])  \
                                                                                                                                                   \
      + (beta_i[ijk+1      +jStride]-beta_i[ijk+1      -jStride]) * (x[ijk+1      +jStride]-x[ijk+jStride]-x[ijk+1      -jStride]+x[ijk-jStride])  \
      + (beta_i[ijk+1      +kStride]-beta_i[ijk+1      -kStride]) * (x[ijk+1      +kStride]-x[ijk+kStride]-x[ijk+1      -kStride]+x[ijk-kStride])  \
      + (beta_j[ijk+jStride+1      ]-beta_j[ijk+jStride-1      ]) * (x[ijk+jStride+1      ]-x[ijk+1      ]-x[ijk+jStride-1      ]+x[ijk-1      ])  \
      + (beta_j[ijk+jStride+kStride]-beta_j[ijk+jStride-kStride]) * (x[ijk+jStride+kStride]-x[ijk+kStride]-x[ijk+jStride-kStride]+x[ijk-kStride])  \
      + (beta_k[ijk+kStride+1      ]-beta_k[ijk+kStride-1      ]) * (x[ijk+kStride+1      ]-x[ijk+1      ]-x[ijk+kStride-1      ]+x[ijk-1      ])  \
      + (beta_k[ijk+kStride+jStride]-beta_k[ijk+kStride-jStride]) * (x[ijk+kStride+jStride]-x[ijk+jStride]-x[ijk+kStride-jStride]+x[ijk-jStride])  \
    )                                                                                                                                              \
  )                                                                                                                                                \
)
```

BASELINE PERFORMANCE

Smooth kernel in HPGMG vs. FMG smooth mini-app

- Test configuration
  - Processor
    - Intel Skylake 8180M
  - Compilers
    - gcc/8.2.0
    - clang++/12.0.0: Intel public SYCL compiler



SYCL/DPC++ IMPLEMENTATION

Creating a queue & setting up buffers

Initialization
(e.g., grids for levels, memory allocations, a series of levels for smooth in a FMG cycle, and so on)

Scope for SYCL execution

Create a queue and set up buffers

Loop for iterations (to measure sustained performance)

Loop for a series of levels for smooth in a FMG cycle (e.g., L2-L2-L1-L2-L2-L1-L0-L1-L2-L2-L1-L0 for 4 levels)

Loop for GSRB smooth (i.e. RBRBRB)

Submit the queue and add accessors to the queue

Finalization
(e.g., reporting smooth kernel time per level, memory deallocation, and so on)

Creating a queue

```
// SYCL
queue myQueue;
auto dev = myQueue.get_device();
auto ctxt = myQueue.get_context();
```

Setting up buffers

```
//
// Set up buffers for SYCL
//
buffer<double, 1> fp_base_all_buf(fp_base_all, range<1>(fp_base_all_size));
buffer<double, 1> b_buf(&b, range<1>(1));
buffer<double, 1> h2inv_levels_buf(h2inv_levels, range<1>(num_levels));
buffer<int, 1> jStride_levels_buf(jStride_levels, range<1>(num_levels));
buffer<int, 1> kStride_levels_buf(kStride_levels, range<1>(num_levels));
buffer<int, 1> lowi_levels_buf(lowi_levels, range<1>(num_levels));
buffer<int, 1> lowj_levels_buf(lowj_levels, range<1>(num_levels));
buffer<int, 1> lowk_levels_buf(lowk_levels, range<1>(num_levels));
buffer<int, 1> ilo_base_buf(ilo_base, range<1>(num_levels_blocks));
buffer<int, 1> jlo_base_buf(jlo_base, range<1>(num_levels_blocks));
buffer<int, 1> klo_base_buf(klo_base, range<1>(num_levels_blocks));
buffer<int, 1> ihi_base_buf(ihi_base, range<1>(num_levels_blocks));
buffer<int, 1> jhi_base_buf(jhi_base, range<1>(num_levels_blocks));
buffer<int, 1> khi_base_buf(khi_base, range<1>(num_levels_blocks));
buffer<int, 1> levels_ijk_lohi_base_id0_buf(levels_ijk_lohi_base_id0,\
                                                range<1>(num_levels));
buffer<uint64_t, 1> levels_vectors_id0_buf(levels_vectors_id0, \
                                                range<1>(num_levels));
buffer<int, 1> levels_box_volume_buf(levels_box_volume, range<1>(num_levels));
```

Submitting the queue & adding accessors to the queue

Initialization
(e.g., grids for levels, memory allocations, a series of levels for smooth in a FMG cycle, and so on)

Scope for SYCL execution

Create a queue and set up buffers

Loop for iterations (to measure sustained performance)

Loop for a series of levels for smooth in a FMG cycle (e.g., L2-L2-L1-L2-L2-L1-L0-L1-L2-L2-L1-L0 for 4 levels)

Loop for GSRB smooth (i.e. RBRBRB)

Submit the queue and add accessors to the queue

Finalization
(e.g., reporting smooth kernel time per level, memory deallocation, and so on)

```
int block,s;
for(s=0;s<2*NUM_SMOOTHS;s++){ // there are two sweeps per GSRB smooth
  TIC_smooth = getTime();

myQueue.submit([&](handler& h) {  // start of the submit
```

Submitting queue & adding accessors

```
   auto fp_base_all_d = fp_base_all_buf.get_access<access::mode::read_write>(h);
   auto b_d = b_buf.get_access<access::mode::read>(h);
   auto h2inv_d = h2inv_levels_buf.get_access<access::mode::read>(h);
   auto jStride_d = jStride_levels_buf.get_access<access::mode::read>(h);
   auto kStride_d = kStride_levels_buf.get_access<access::mode::read>(h);
   auto lowi_d = lowi_levels_buf.get_access<access::mode::read>(h);
   auto lowj_d = lowj_levels_buf.get_access<access::mode::read>(h);
   auto lowk_d = lowk_levels_buf.get_access<access::mode::read>(h);
   auto ilo_base_d = ilo_base_buf.get_access<access::mode::read>(h);
   auto jlo_base_d = jlo_base_buf.get_access<access::mode::read>(h);
   auto klo_base_d = klo_base_buf.get_access<access::mode::read>(h);
   auto ihi_base_d = ihi_base_buf.get_access<access::mode::read>(h);
   auto jhi_base_d = jhi_base_buf.get_access<access::mode::read>(h);
   auto khi_base_d = khi_base_buf.get_access<access::mode::read>(h);
   auto levels_ijk_lohi_base_id0_d = \
            levels_ijk_lohi_base_id0_buf.get_access<access::mode::read>(h);
   auto levels_vectors_id0_d = levels_vectors_id0_buf.get_access<access::mode::read>(h);
   auto levels_box_volume_d = levels_box_volume_buf.get_access<access::mode::read>(h);

   h.parallel_for<class smooth_a_block>(range<3>(1,1,level->num_my_blocks), // global range
       id<3>(ilevel,s,0),   // offset
       [=](item<3> item) {
     int ilevel = item[0];
     int s = item[1];
     int block = item[2];
     int numVectors = VECTORS_RESERVED;
     int box_volume = levels_box_volume_d[ilevel];
     double b = b_d[0];

     const int ilo = ilo_base_d[ levels_ijk_lohi_base_id0_d[ilevel]+block ];
     const int jlo = jlo_base_d[ levels_ijk_lohi_base_id0_d[ilevel]+block ];
     const int klo = klo_base_d[ levels_ijk_lohi_base_id0_d[ilevel]+block ];
     const int ihi = ihi_base_d[ levels_ijk_lohi_base_id0_d[ilevel]+block ];
     const int jhi = jhi_base_d[ levels_ijk_lohi_base_id0_d[ilevel]+block ];
     const int khi = khi_base_d[ levels_ijk_lohi_base_id0_d[ilevel]+block ];

     int i,j,k;
     const double h2inv = h2inv_d[ilevel];
     const int jStride = jStride_d[ilevel];
     const int kStride = kStride_d[ilevel];
     // is element 000 red or black on *THIS* sweep
     const int color000 = (lowi_d[ilevel]^lowj_d[ilevel]^lowk_d[ilevel]^s)&1;

     double * vectors[VECTORS_RESERVED];
     for(int v=0;v<numVectors;v++){vectors[v] = &fp_base_all_d[0] +\
          levels_vectors_id0_d[ilevel] + (uint64_t)box_volume*v;} // setup vector pointers
```

parallel_for in the queue

```
// continue to the next page
```

Submitting the queue & adding accessors to the queue

Initialization
(e.g., grids for levels, memory allocations, a series of levels for smooth in a FMG cycle, and so on)

Scope for SYCL execution

Create a queue and set up buffers

Loop for iterations (to measure sustained performance)

Loop for a series of levels for smooth in a FMG cycle (e.g., L2-L2-L1-L2-L2-L1-L0-L1-L2-L2-L1-L0 for 4 levels)

Loop for GSRB smooth (i.e. RBRBRB)

Submit the queue and add accessors to the queue

Finalization
(e.g., reporting smooth kernel time per level, memory deallocation, and so on)

```
// continue from the previous page

    const double * __restrict__ rhs   = vectors[VECTOR_R     ]+ghosts*(1+jStride+kStride);
    const double * __restrict__ beta_i= vectors[VECTOR_BETA_I]+ghosts*(1+jStride+kStride);
    const double * __restrict__ beta_j= vectors[VECTOR_BETA_J]+ghosts*(1+jStride+kStride);
    const double * __restrict__ beta_k= vectors[VECTOR_BETA_K]+ghosts*(1+jStride+kStride);
    const double * __restrict__ Dinv  = vectors[VECTOR_DINV  ]+ghosts*(1+jStride+kStride);
    const double * __restrict__ x_n;
          double * __restrict__ x_np1;
                   if((s&1)==0){x_n   = vectors[VECTOR_U     ]+ghosts*(1+jStride+kStride);
                                x_np1 = vectors[VECTOR_TEMP  ]+ghosts*(1+jStride+kStride);}
                         else{x_n   = vectors[VECTOR_TEMP  ]+ghosts*(1+jStride+kStride);
                                x_np1 = vectors[VECTOR_U     ]+ghosts*(1+jStride+kStride);}
    for(k=klo;k<khi;k++){
    for(j=jlo;j<jhi;j++){
      // out-of-place must copy old value...
      for(i=ilo;i<ihi;i++){
        int ijk = i + j*jStride + k*kStride;
        x_np1[ijk] = x_n[ijk];
      }
      for(i=ilo+((ilo^j^k^color000)&1);i<ihi;i+=2){ // stride-2 GSRB
        int ijk = i + j*jStride + k*kStride;
        double Ax     = apply_op_ijk(x_n);
        x_np1[ijk] = x_n[ijk] + Dinv[ijk]*(rhs[ijk]-Ax);
      }
    }}

  });    // End of smooth_a_block kernel
```

End of parallel_for in the queue

```
});  // end of the submit
```

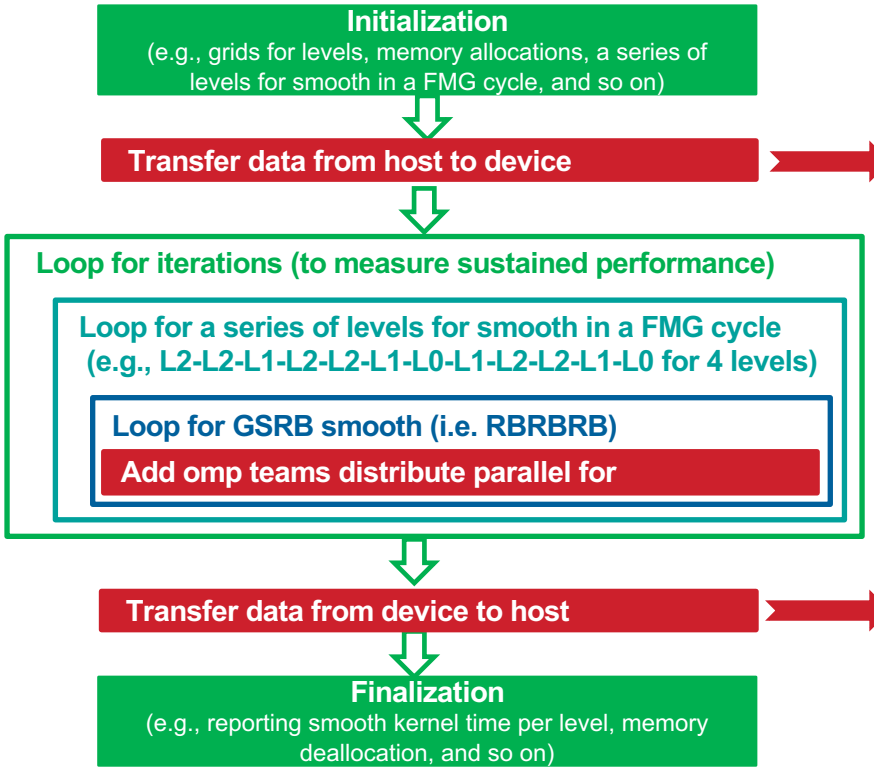
End of the queue submission

```
TOC_smooth = getTime();
timer = TOC_smooth - TIC_smooth;
level->timers.smooth += (double)(timer.count());

} // s-loop
```

OPENMP TARGET OFFLOADING IMPLEMENTATION

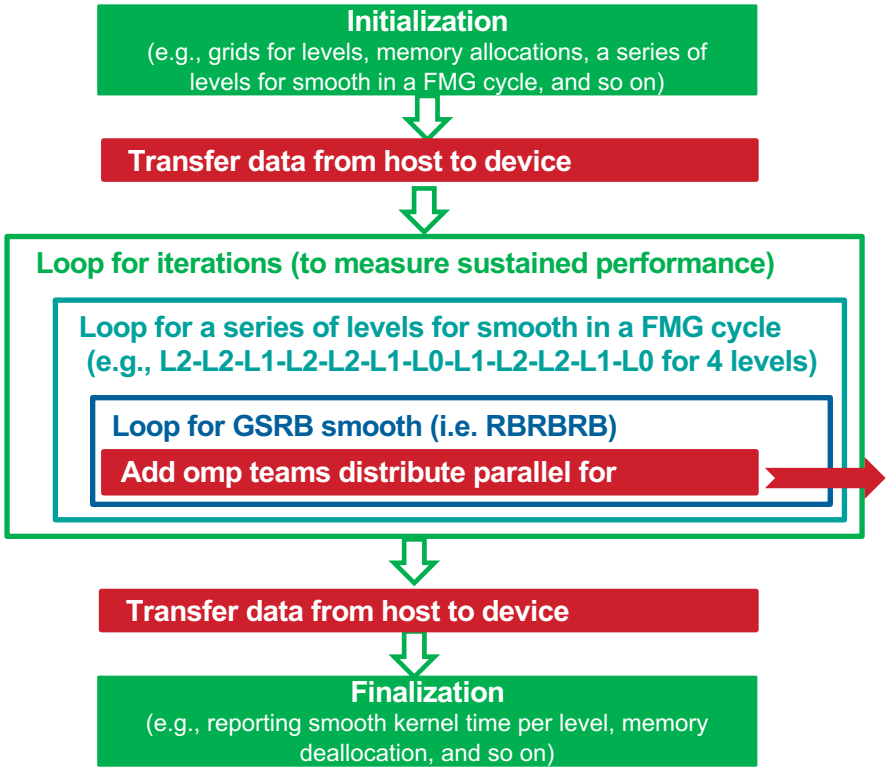
omp target enter/exit data map




```
#pragma omp target enter data map(to: fp_base_all[0:fp_base_all_size])
#pragma omp target enter data map(to: h2inv_levels[0:num_levels])
#pragma omp target enter data map(to: jStride_levels[0:num_levels])
#pragma omp target enter data map(to: kStride_levels[0:num_levels])
#pragma omp target enter data map(to: lowi_levels[0:num_levels])
#pragma omp target enter data map(to: lowj_levels[0:num_levels])
#pragma omp target enter data map(to: lowk_levels[0:num_levels])
#pragma omp target enter data map(to: ilo_base[0:num_levels_blocks])
#pragma omp target enter data map(to: jlo_base[0:num_levels_blocks])
#pragma omp target enter data map(to: klo_base[0:num_levels_blocks])
#pragma omp target enter data map(to: ihi_base[0:num_levels_blocks])
#pragma omp target enter data map(to: jhi_base[0:num_levels_blocks])
#pragma omp target enter data map(to: khi_base[0:num_levels_blocks])
#pragma omp target enter data map(to: levels_ijk_lohi_base_id0[0:num_levels])
#pragma omp target enter data map(to: levels_vectors_id0[0:num_levels])
#pragma omp target enter data map(to: levels_box_volume[0:num_levels])
```

```
#pragma omp target exit data map(release: fp_base_all[0:fp_base_all_size])
#pragma omp target exit data map(release: h2inv_levels[0:num_levels])
#pragma omp target exit data map(release: jStride_levels[0:num_levels])
#pragma omp target exit data map(release: kStride_levels[0:num_levels])
#pragma omp target exit data map(release: lowi_levels[0:num_levels])
#pragma omp target exit data map(release: lowj_levels[0:num_levels])
#pragma omp target exit data map(release: lowk_levels[0:num_levels])
#pragma omp target exit data map(release: ilo_base[0:num_levels_blocks])
#pragma omp target exit data map(release: jlo_base[0:num_levels_blocks])
#pragma omp target exit data map(release: klo_base[0:num_levels_blocks])
#pragma omp target exit data map(release: ihi_base[0:num_levels_blocks])
#pragma omp target exit data map(release: jhi_base[0:num_levels_blocks])
#pragma omp target exit data map(release: khi_base[0:num_levels_blocks])
#pragma omp target exit data map(release: levels_ijk_lohi_base_id0[0:num_levels])
#pragma omp target exit data map(release: levels_vectors_id0[0:num_levels])
#pragma omp target exit data map(release: levels_box_volume[0:num_levels])
```

omp target teams distribute parallel for




```
int block,s;
for(s=0;s<2*NUM_SMOOTHS;s++){ // there are two sweeps per GSRB smooth
   double _timeStart = getTime();

   // loop over all block/tiles this process owns...
   #pragma omp target teams distribute parallel for
   for(block=0;block<level->num_my_blocks;block++){
      const int ilo = ilo_levels[ilevel][block];
      const int jlo = jlo_levels[ilevel][block];
      const int klo = klo_levels[ilevel][block];
      const int ihi = ihi_levels[ilevel][block];
      const int jhi = jhi_levels[ilevel][block];
      const int khi = khi_levels[ilevel][block];

      int i,j,k;
      const double h2inv = h2inv_levels[ilevel];
      const int jStride = jStride_levels[ilevel];
      const int kStride = kStride_levels[ilevel];
      // is element 000 red or black on *THIS* sweep
      const int color000 = (lowi_levels[ilevel]^lowj_levels[ilevel]^lowk_levels[ilevel]^s)&1;

      const double * __restrict__ rhs    = vectors[       rhs_id] + ghosts*(1+jStride+kStride);
      const double * __restrict__ beta_i= vectors[VECTOR_BETA_I] + ghosts*(1+jStride+kStride);
      const double * __restrict__ beta_j= vectors[VECTOR_BETA_J] + ghosts*(1+jStride+kStride);
      const double * __restrict__ beta_k= vectors[VECTOR_BETA_K] + ghosts*(1+jStride+kStride);
      const double * __restrict__ Dinv   = vectors[VECTOR_DINV  ] + ghosts*(1+jStride+kStride);
      const double * __restrict__ x_n;
      double * __restrict__ x_np1;
      if((s&1)==0){x_n      = vectors[         x_id] + ghosts*(1+jStride+kStride);
                   x_np1    = vectors[VECTOR_TEMP  ] + ghosts*(1+jStride+kStride);}
      else{        x_n      = vectors[VECTOR_TEMP  ] + ghosts*(1+jStride+kStride);
                   x_np1    = vectors[         x_id] + ghosts*(1+jStride+kStride);}

      for(k=klo;k<khi;k++){
      for(j=jlo;j<jhi;j++){
      // out-of-place must copy old value...
         for(i=ilo;i<ihi;i++){
            int ijk = i + j*jStride + k*kStride;
            x_np1[ijk] = x_n[ijk];
         }
         for(i=ilo+((ilo^j^k^color000)&1);i<ihi;i+=2){ // stride-2 GSRB
            int ijk = i + j*jStride + k*kStride;
            double Ax     = apply_op_ijk(x_n);
            x_np1[ijk] = x_n[ijk] + Dinv[ijk]*(rhs[ijk]-Ax);
         }
      }}

   } // boxes

   level->timers.smooth += (double)(getTime()-_timeStart);

} // s-loop
```

PERFORMANCE ON INTEL GEN9 GPU & NVIDIA V100 GPU

TEST BED SYSTEMS IN ARGONNE JLSE [2]

Intel Gen9 integrated GPU vs. NVIDIA V100 discrete GPU

*** Theoretical peak**
**** Measured peak**

JLSE Nodes	GPU_V100	IRIS	V100/Gen9
GPU	NVIDIA V100	Intel Gen9 GT4e	
GPU DP peak	7.8 TFLOPS*	0.33 TFLOPS*	24x
GPU BW peak	900 GB/s (HBM)*	34.1 GB/s (DRAM)* 73.6 GB/s (EDRAM via GTI)**	26x (HBM/DRAM) 12x (HBM/EDRAM)
SYCL/DPC++ Compiler	Public LLVM SYCL-CUDA	Intel DPC++ Beta09	
OpenMP Target Compiler	LLVM/ clang-11.0.0, IBM/ xlc-16.1.1	Intel ICX Beta09	

Remark

Intel Gen9 integrated GPU is not intended for HPC workloads. It's just being used as a proxy development platform to try out the new programming model in advance of datacenter GPUs being available.

TILE CONFIGURATION VS. # OF BLOCKS

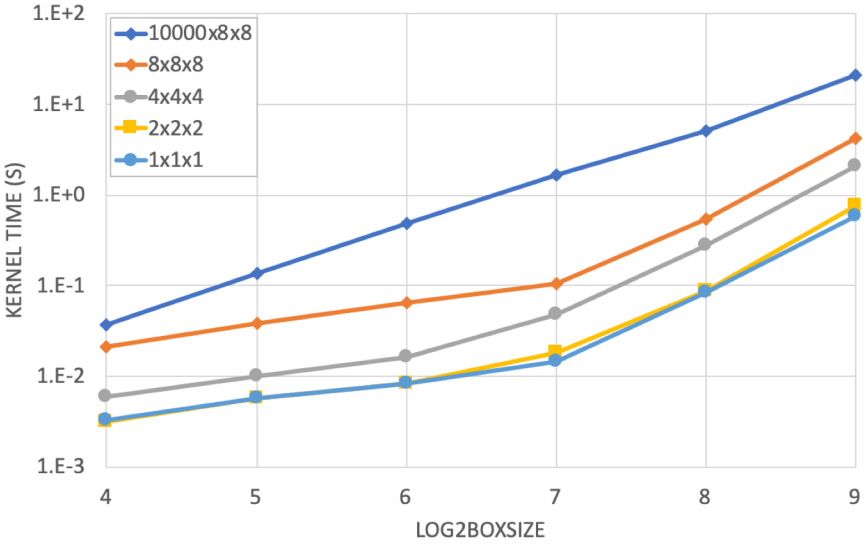
Level	Tile configurations				
	10000x8x8	8x8x8	4x4x4	2x2x2	1x1x1
1	1	1	1	1	8
2	1	1	1	8	64
3	1	1	8	64	512
4	4	8	64	512	4,096
5	16	64	512	4,096	32,768
6	64	512	4,096	32,768	262,144
7	256	4,096	32,768	262,144	2,097,152
8	1,024	32,768	262,144	2,097,152	16,777,216
9	4,096	262,144	2,097,152	16,777,216	134,217,728

More items for parallel_for

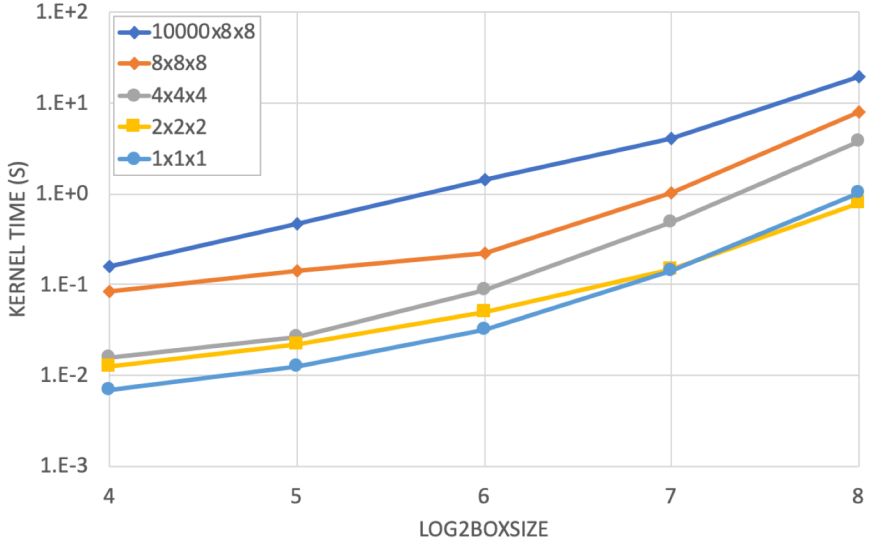
SYCL/DPC++ PERFORMANCE

with various tile configurations

*Lower is better



On NVIDIA V100 GPU



On Intel Gen9 GPU

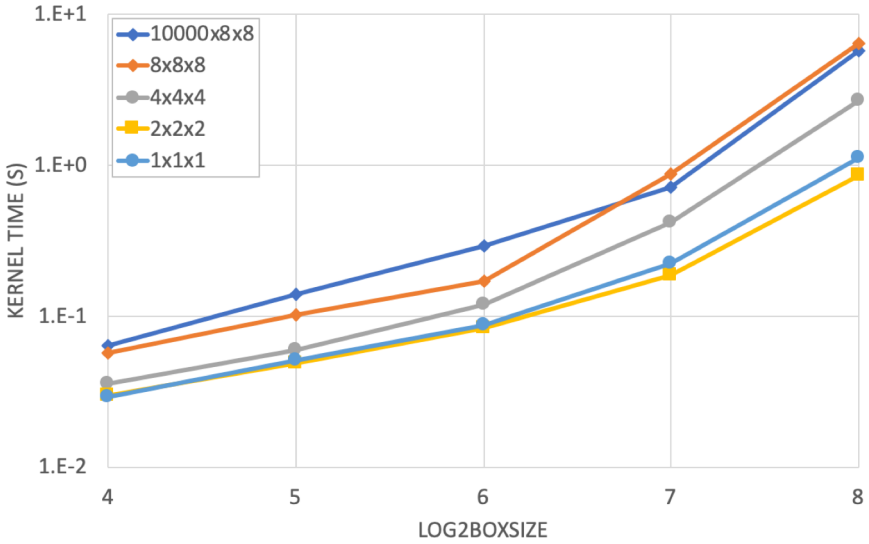
OPENMP OFFLOADING PERFORMANCE

with various tile configurations

Compiler	Runtime errors
LLVM/ clang-11.0.0	Libomptarget fatal error
IBM/ xlc-16.1.1	illegal memory access issue
Possibly using pointers inside of the target regions could be problematic with some compilers, so a reproducer (at the end of this talk) was tested with the above compilers.	

On NVIDIA V100 GPU

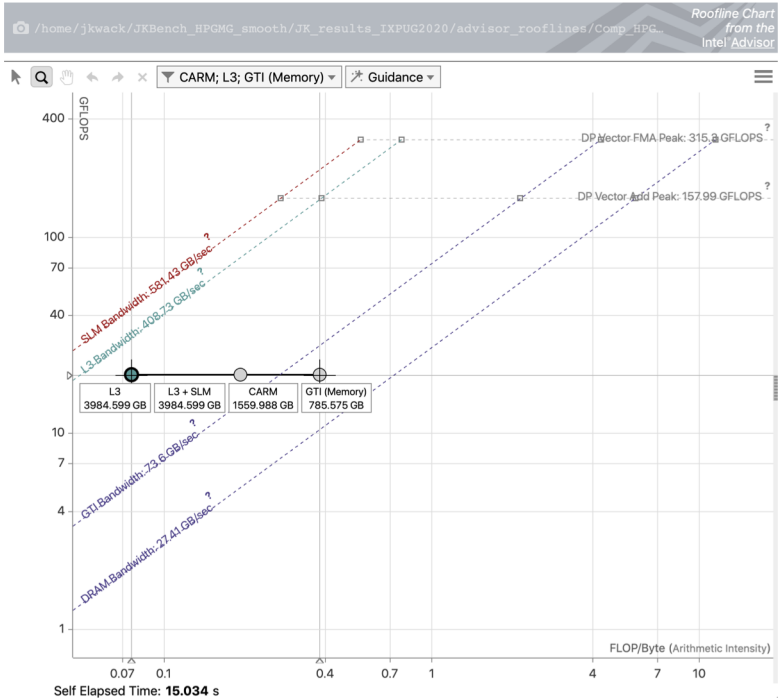
*Lower is better



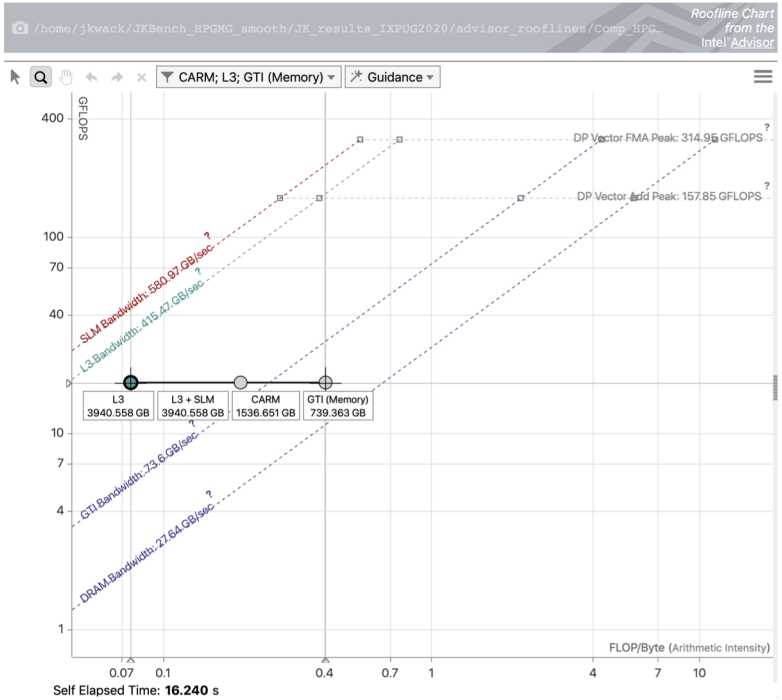
On Intel Gen9 GPU

ADVISOR ROOFLINE FEATURES ON INTEL GEN9

256^3 grid with 2x2x2 configuration



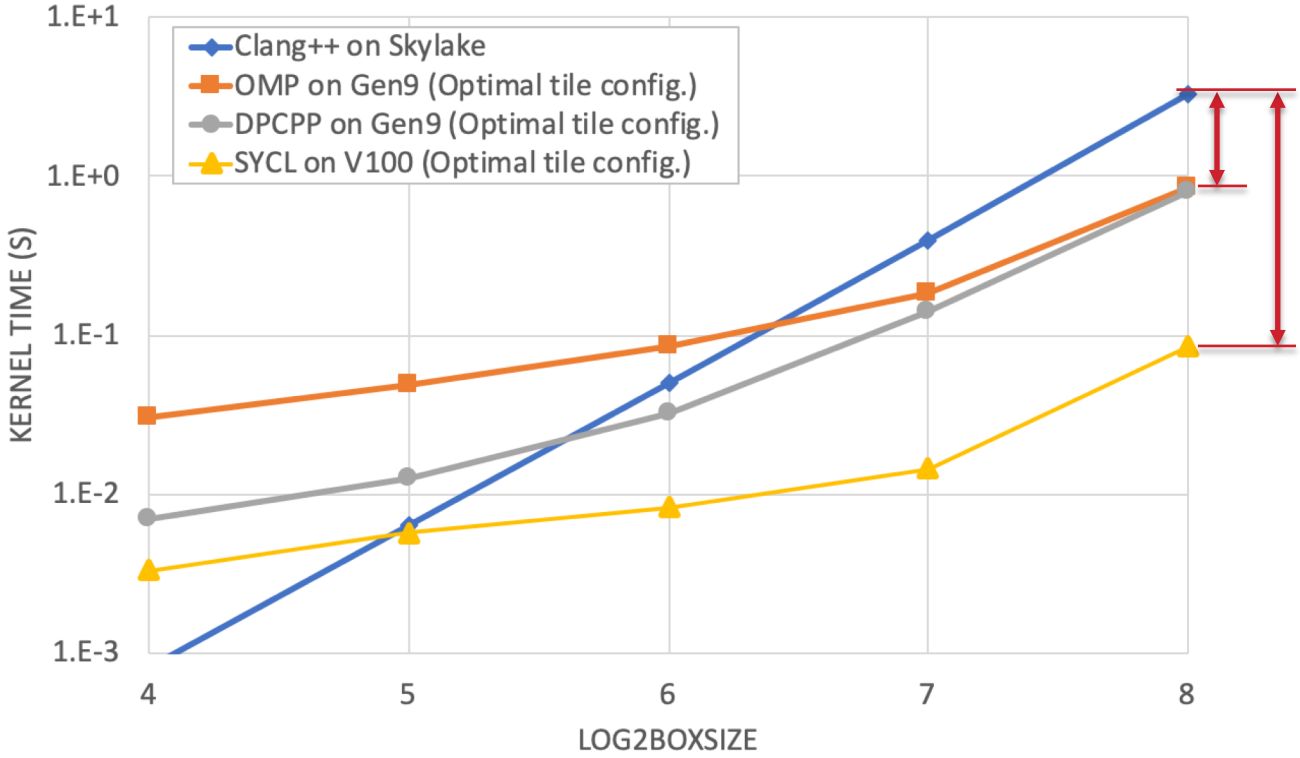
SYCL/DPC++ version



OpenMP target offloading version

PERFORMANCE COMPARISON

w/ best tile configurations for GPUs



Around 10x speedup on V100 over Gen9: reasonable speedup since the kernel is BW bound

Test code with a pointer in an OpenMP target region

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <omp.h>
#define n_base 100

int main(int argc, char *argv[]){

  double * base;
  unsigned long int n_base2=n_base*n_base;
  base = (double *)malloc(n_base2*sizeof(double));

  for (int i=0;i<n_base2;i++) {base[i] = 0.0;}            // Initialization

  #pragma omp target enter data map(to:base[0:n_base2])   // Target enter data

  // loop over all block/tiles this process owns...
  #pragma omp target teams distribute parallel for
  for(int i=0;i<n_base;i++){
    double * vectors[n_base];
    for(int j=0;j<n_base;j++) {vectors[j] = &base[0] + j*n_base;};
    double * __restrict__ base_tmp = vectors[i];
    for(int j=0;j<n_base;j++) {base_tmp[j] = (double)(i*n_base+j);}
  }

  // Target exit data
  #pragma omp target exit data map(from:base[0:n_base2])

  // Print out the results, deallocate base, and return
  for(int i=n_base2-10;i<n_base2;i++) {printf("base[%d]=%lf\n",i,base[i]);}
  free(base);  return(0);
}
```

Compiler	GPU	Max array size*
xlc	NVIDIA V100	1102^2
clang	NVIDIA V100	22116^2
icx	Intel Gen9	23170^2

*Larger is better

With the same code, OpenMP target offloading compilers have different capacities to handle an array with pointers in the device.

CONCLUDING REMARKS

CONCLUDING REMARKS

- FMG Smooth mini-app has been extracted from HPGMG-FV.
- FMG Smooth mini-app has been implemented with SYCL/DPC++
  - With a large number of blocks, the SYCL/DPC++ version shows meaningful speed-up on NVIDIA V100 and Intel Gen9 GPUs
- FMG Smooth mini-app has been implemented with OpenMP Target Offloading.
  - Successfully, built and tested it on Intel Gen9
  - On NVIDIA V100, LLVM clang and IBM xlc compilers returned runtime errors.
  - Via a test code, it turns out that compilers show different capacity to handle an array with pointers in a target region.

- Next step
  - SYCL implementation for full HPGMG-FV
  - OpenMP Target offloading tests with other compilers (e.g., PGI, Cray)
  - Kokkos implementation (Working-in-progress)

ACKNOWLEDGEMENT

- This work was supported by
  - the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357,
  - and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration).
- We also gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.
- We would like to thank Kent Moffat from Intel for reviewing data and providing feedback for this talk.

REFERENCES

1. "HPGMG webpage," https://hpgmg.org, 2020.
2. "JLSE webpage", https://www.jlse.anl.gov, 2020.

THANKS!