2022 ECP ANNUAL MEETING

Argonne NATIONAL LABORATORY

ROOFLINE PERFORMANCE ANALYSIS W/ INTEL ADVISOR ON INTEL CPUS & GPUS

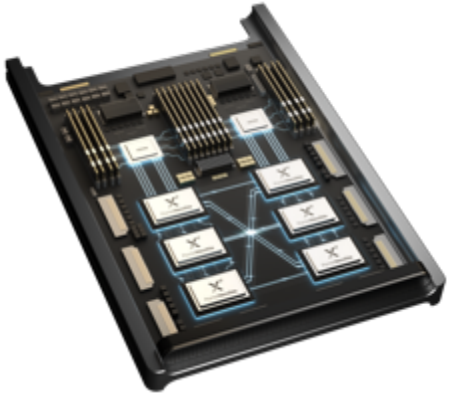
JAEHYUK KWACK
ALCF Perf. Engr. Group

5/2/2022

OVERVIEW OF AURORA TESTBED SYSTEMS

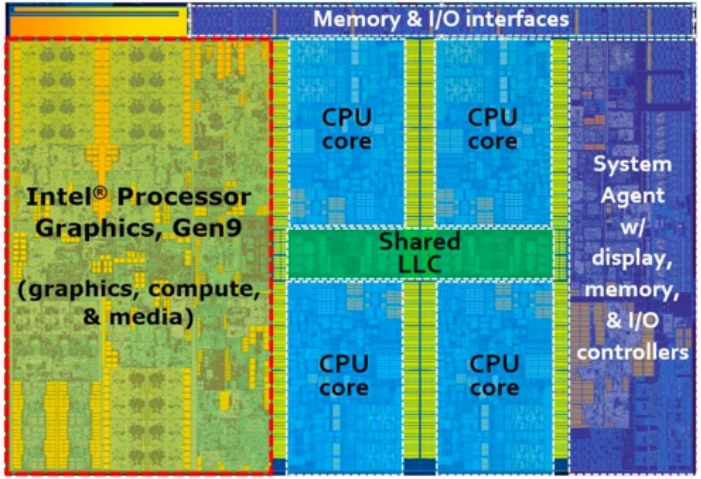
AURORA: A HIGH-LEVEL VIEW

- Intel-HPE machine arriving at Argonne
  - Sustained Performance **> 1 ExaFlops**
- Intel Xeon processor and Intel X^e GPUs
  - 2 Xeons (Sapphire Rapids)
  - 6 GPUs (Ponte Vecchio [PVC])
- Greater than 10 PB of total memory
- HPE/Cray XE platform and HPE Slingshot network
- Filesystem
  - Distributed Asynchronous Object Store (DAOS)
    - ≥ 230 PB of storage capacity
    - Bandwidth of > 25 TB/s
  - Lustre
    - 150 PB of storage capacity
    - Bandwidth of ~ 1TB/s

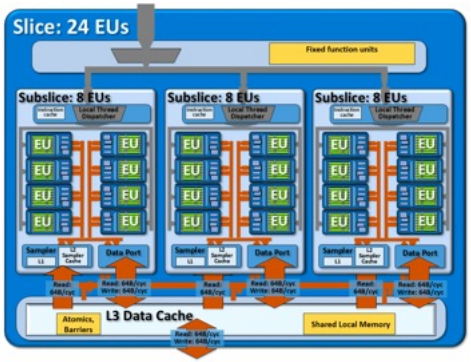


INTEL GEN9 ON TESTBED

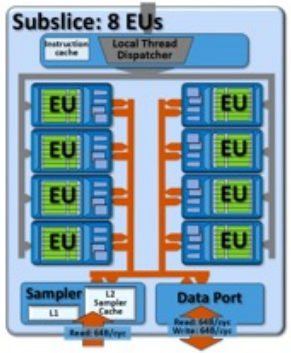
- Gen9: Double precision peak performance: 100-300 GF
  - Low by design due to power and space limits
  - Integrated GPU
- Hardware hierarchies
  - A GPU tile has multiple slices
  - A slice has multiple Sub-Slices
  - A sub-slice has multiple EUs



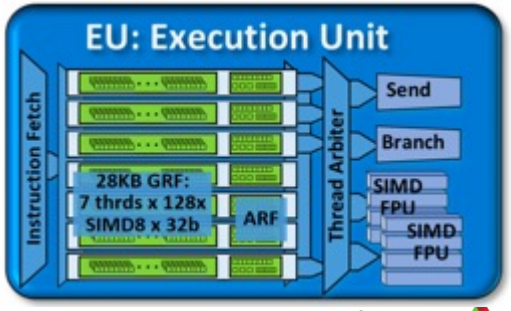
Slice



Sub-Slice



Execution Unit (EU)



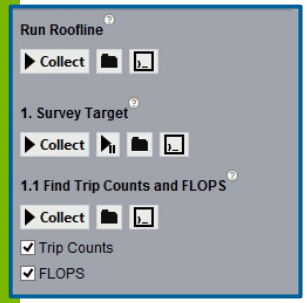
REMARKS BEFORE WE START

To help reduce any misunderstanding

- The Intel Gen9 GPU is a much lower performing device that is integrated into the same package as the CPU. While Intel has announced plans to introduce a new line of X^e brand high performance discrete GPUs, that hardware is not publicly available at this moment. The Gen9 GPU is therefore the most suitable Intel GPU for evaluation of HPC applications currently available.

ADVISOR ON INTEL CPUS AND GPUS

GETTING ROOFLINE DATA IN INTEL® ADVISOR: **TWO-PASS APPROACH**



Roofline : Axis X: **AI** = **#FLOP** / **#Bytes** Axis Y: **FLOP/S** = **#FLOP** (mask aware) / **#Seconds**	Overhead
Step 1: Survey (-collect survey) - Provide **#Seconds** - Root access not needed - User mode sampling, non-intrusive.	**1x**
Step 2: FLOPS (-collect tripcounts –flops) - Provide **#FLOP**, **#Bytes**, AVX-512 Mask - Root access not needed - Precise, instrumentation based, count number of instructions	**3-5x**

ORIGINAL, CACHE-AWARE (CARM) AND MEMORY-LEVEL ROOFLINE

CARM (cache-aware roofline)

- Single **AI** based on aggregated traffic:

  CPU core (GPU EUs) <-> memory sub-system
- Ceilings for compute, cache/memory levels
- AI independent of problem size

Unique features: algorithmic focus and simplicity

Original Roofline

- AI based on external memory :

  DDR (GPU GTI)
- Ceilings for DDR and compute
- AI dependent of problem size

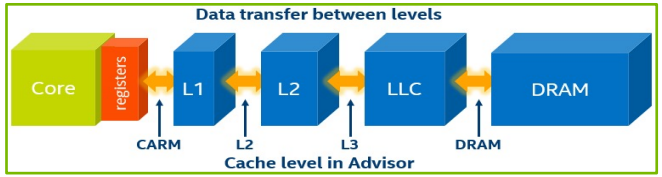
Unique features: DDR bound focus and simplicity

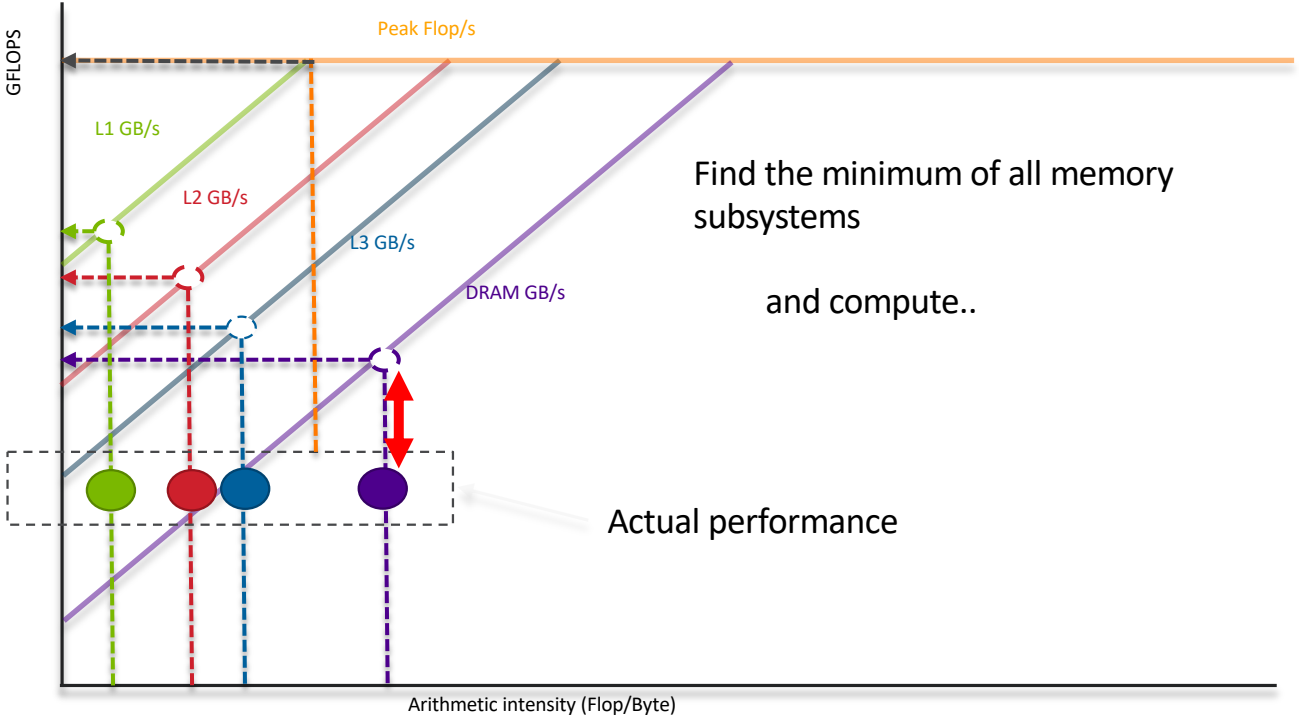
Memory Level Roofline - **MLR** (see also “Hierarchical Roofline” by LBL)

- **AI** for all memory sub-system levels, combines (1), CARM, (2)Original and (3) Lx-only perspectives
- Harder to interpret for multiple kernels at a time

Unique features: unambiguous bottleneck detection

HOW TO INTERPRET MLR ON CPU ?





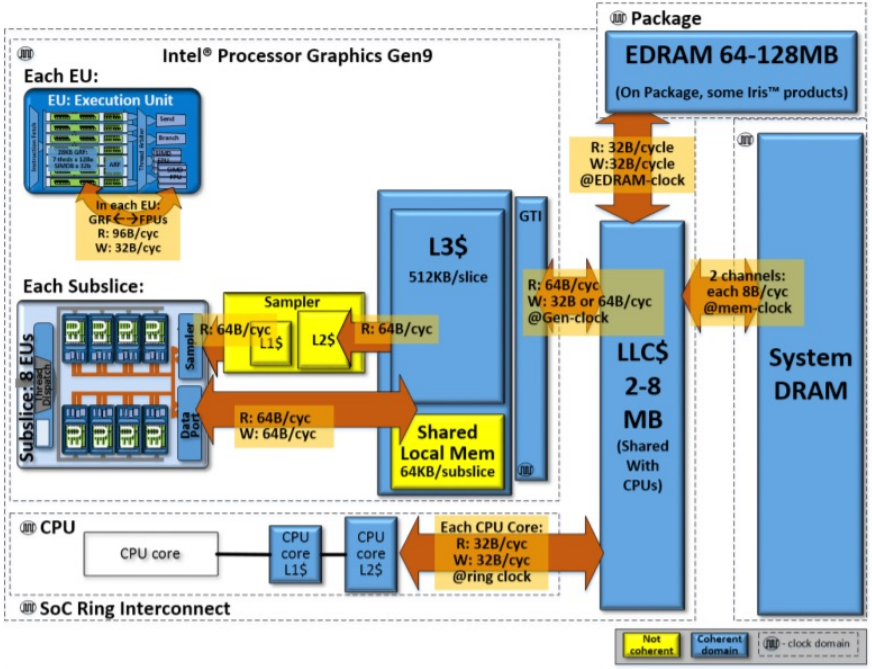
Find the minimum of all memory subsystems

and compute..

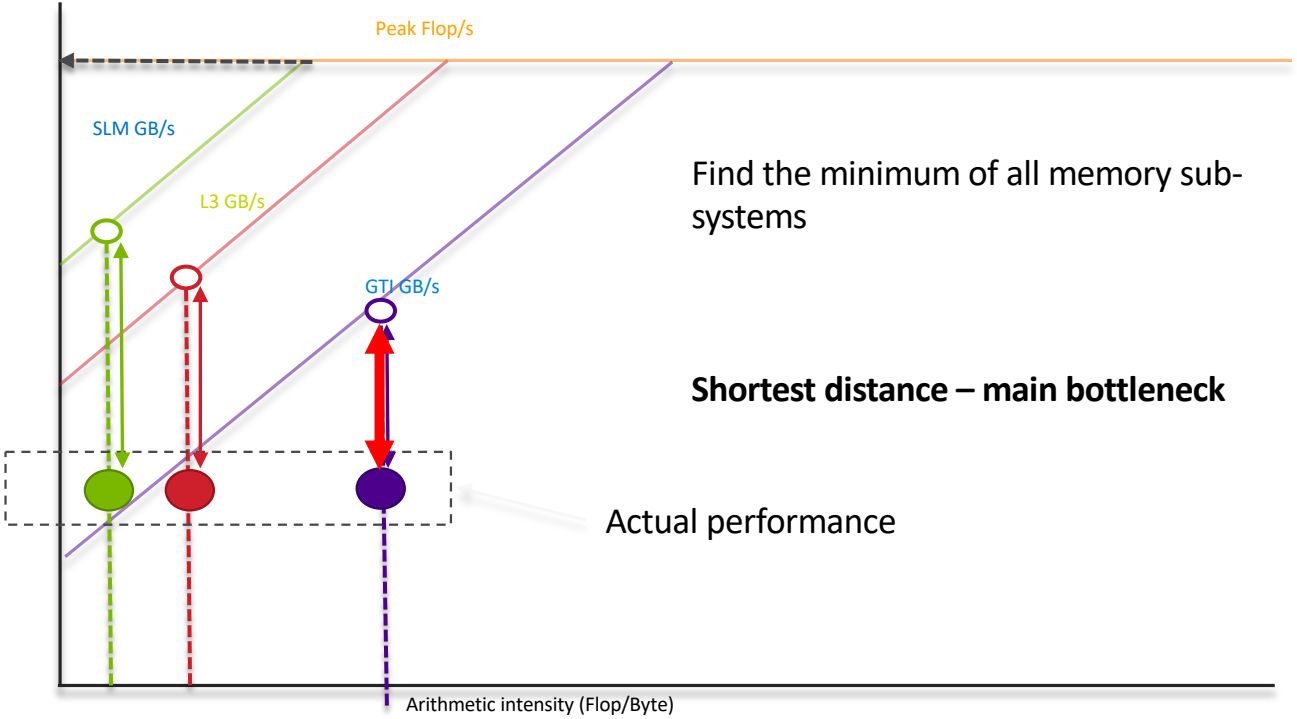
Actual performance

ADVISOR GPU ROOFLINE FEATURES

- Advisor provides an effective way for GPU roofline analysis on Intel GPUs.
- Memory Levels
  - **CARM**: Memory traffic generated by all execution units (EUs). Includes traffic between EUs and corresponding GPU cache or direct traffic to main memory. For each retired instruction with memory arguments, the size of each memory operand in bytes is added to this metric.
  - **L3**: Data transferred directly between execution units and L3 cache.
  - **SLM**: Memory access to/from Shared Local Memory (SLM), a dedicated structure within the L3 cache.
  - **GTI**: Represents GTI traffic/GPU memory read bandwidth, the accesses between the GPU, chip uncore (LLC), and main memory. Use this to get a sense of external memory traffic.
  - **L3 + SLM**: Summary traffic to/from L3 and Shared Local Memory.



HOW TO INTERPRET MLR ON GPU ?



Find the minimum of all memory sub-systems

Shortest distance – main bottleneck

Actual performance

HOW TO GENERATE CARM CPU ROOFLINE PROFILE?

As simple as: `$ advisor -collect roofline -- <your-executable-with-parameters>`

More details / How-To

```
$ source advisor-vars.sh
```

1st method. Not compatible with MPI applications :

```
$ advisor -collect roofline --project-dir
./your_project -- <your-executable-with-
parameters>
```

2nd method (compatible with MPI, more flexible):

```
$ advisor -collect survey --project-dir ./your_project --
<your-executable-with-parameters>

$ advisor -collect tripcounts --flop --project-dir
./your_project -- <your-executable-with-parameters>
```

(optional) copy data to your UI desktop system

```
$ advisor-gui ./your_project

$ advisor -report roofline --project-dir ./your_project > roofline.html
```

HOW TO GENERATE MLR+CARM CPU ROOFLINE PROFILE?

As simple as: `$ advisor -collect roofline -enable-cache-simulation -- <your-executable-with-parameters>`

More details / How-To

```
$ source advisor-vars.sh
```

1st method. Not compatible with MPI applications :

```
$ advisor -collect roofline -enable-cache-simulation --project-dir ./your_project --
<your-executable-with-parameters>
```

2nd method (compatible with MPI, more flexible):

```
$ advisor -collect survey --project-dir ./your_project --
<your-executable-with-parameters>

$ advisor -collect tripcounts -flop -enable-cache-simulation
--project-dir ./your_project -- <your-executable-with-
parameters>
```

(optional) copy data to your UI desktop system

```
$ advisor-gui ./your_project

$ advisor -report roofline --project-dir ./your_project > roofline.html
```

HOW TO GENERATE GPU (MLR & CARM) ROOFLINE PROFILE?

As simple as: $ advisor -collect roofline **--profile-gpu** -- <your-executable-with-parameters>

More details / How-To

```
$ source advisor-vars.sh
```

1st method. Not compatible with MPI applications :

```
$ advisor -collect roofline --profile-gpu --project-dir ./your_project -- <your-executable-with-parameters>
```

2nd method (compatible with MPI, more flexible):

```
$ advisor -collect survey --profile-gpu --project-dir ./your_project -- <your-executable-with-parameters>

$ advisor -collect tripcounts -flop --profile-gpu --project-dir ./your_project -- <your-executable-with-parameters>
```

(optional) copy data to your UI desktop system

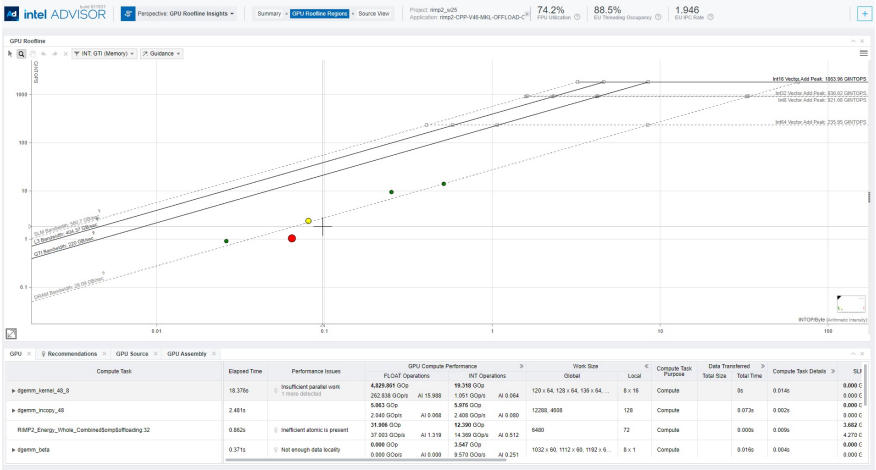
```
$ advisor-gui ./your_project

$ advisor -report roofline --gpu --project-dir ./your_project > roofline.html
```

--data-type=int | float(default) |mixed

GPU ROOFLINE: *EXTENDED* HTML GUI

See HTML report in project-dir/e000|rank.*/report folder by default

```
source advisor_install_dir/advisor-vars.sh

advisor
 --report all
 --project-dir ./your_project
 --report-output ./roofline.html
```



- View the result in web browser without having Intel® Advisor installed

EXTENDED HTML GUI

For any system with web browsers (e.g., Mac(M1), phones & tablets)

intel ADVISOR build 611353 | Perspective: GPU Roofline Insights | Summary · GPU Roofline Regions · Source View
Project: SC21_Advisor_rimp2_w25
Application: rimp2-CPP-V50-MKL-OMP-OFFLOAD

Program Metrics

33.63s
Program Elapsed Time

GPU

1.34 Compute (GINTOPS)

76.0%
FPU Utilization

OP/S and Bandwidth

GPU

ROOFLINE

GFLOPS

L3 Bandwidth: 403.74 GB/sec

This application is bounded by the Compute (GFLOPS)

Top Hotspots

GPU

Compute Task	Elapsed Time
dgemm_kernel_48_8	18.40s
dgemm_incopy_48	2.50s
dgemm_beta	0.49s
dgemm_oncopy_8	0.26s
binary_test	<0.01s

intel ADVISOR build 611353 | Perspective: GPU Roofline Insights | Summary · GPU Roofline Regions · Source View
Project: SC21_Advisor_rimp2_w25
Application: rimp2-CPP-V46-MKL-OFFLOAD-OPT

73.2% FPU Utilization | 88.4% EU Threading Occupancy | 1.943 EU IPC Rate

GPU Roofline

FLOAT; L3; SLM; GTI (Memory) | Guidance

GFLOPS

SLM Bandwidth: 580.64 GB/sec
L3 Bandwidth: 403.74 GB/sec
GTI Bandwidth: 220.06 GB/sec
DRAM Bandwidth: 27.69 GB/sec

L3 163.886

Self Elapsed Time: 1.021 s

GPU

Compute Task

- dgemm_oncopy_8
- dgemm_kernel_48_8
- RIMP2_Energy_Whole_Combinedompoffloading:32

intel ADVISOR build 611353 | Perspective: Offload Modeling | Summary · Accelerated Regions · Source View
Project: SC21_Advisor_rimp2_w25
Application: rimp2-CPP-V46-MKL-OFFLOAD-OPT

Top Metrics

1.0x
Speed-up for Accelerated Code

142
Number of Offloads

Program Metrics

Time on Baseline GPU	22.67s
Time on Target	23.72s

Estimated Time on GPU	23.70s
Data Transfer Tax	0s
Kernel Launch Tax	0.02s

Speed-up for Accelerated Code	1.0x
Number of Offloads	142
Target Platform	Gen9 GT4e
Baseline Platform	Intel (R) Iris Pro Graphics P580

Offload Bounded By

Compute	81%
L3 Cache BW	0%
LLC BW	0%
Memory BW	19%
SLM BW	0%
GTI BW	0%
Atomics	0%
Latencies	0%
Data Transfer	0%
Launch Tax	<1%
Trip Count	<1%
Non-Modeled	<1%

Top Offloaded

Kernel	Execution Time	Speed-Up	Bounded By	Data Transfer
dgemm_beta	Baseline 0.2ms, Target < 0.1ms	37.243x	Launch Tax, Compute	0B
dgemm_beta	Baseline 0.2ms, Target < 0.1ms	29.787x	Launch Tax, Compute	0B
dgemm_beta	Baseline 0.3ms, Target 0.2ms	1.149x	DRAM BW	0B
dgemm_beta	Baseline 0.3ms, Target 0.4ms	0.634x	DRAM BW	0B
dgemm_beta	Baseline 0.4ms, Target 0.5ms	0.709x	DRAM BW	0B

Top Non-Offloaded

No Data Available

ROOFLINE ON MULTI-GPU SYSTEMS

Add --target-gpu option in command line

```
advisor
 --collect roofline
 --profile-gpu
 --project-dir  ./your_project
 --target-gpu 0:77:0.0
 -- <your-executable-with-
parameters>
```

To get target GPU value

- Run advisor –help collect | grep –target-gpu

```
                                    OS) option with an Intel compiler.
                                    Tip: Disabling can minimize overhead.
--target-gpu=0:0:2.0 | 0:3:0.0 (0:3:0.0)
                                    The target GPU adapter that will be used to
                                    collect GPU profiling data.
--target-pid=<unsigned integer>
                                    Attach collection to a running process specified
```

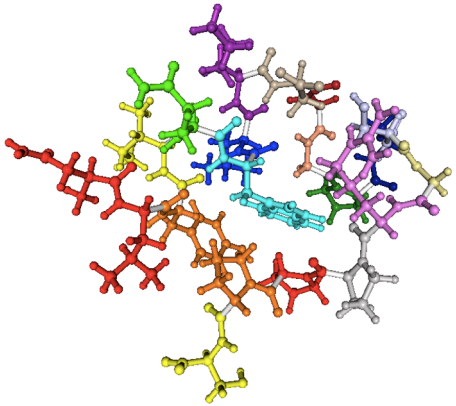
ROOFLINE DEMO CASE

- GAMESS RI-MP2 MINI-APP

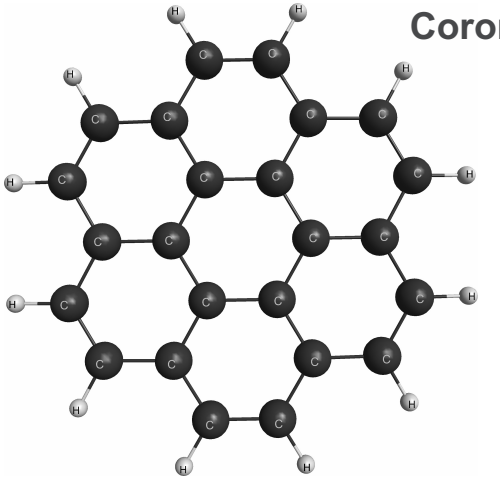
Collaborated w/ Colleen Bertoni (ANL)

INTRO OF GAMESS

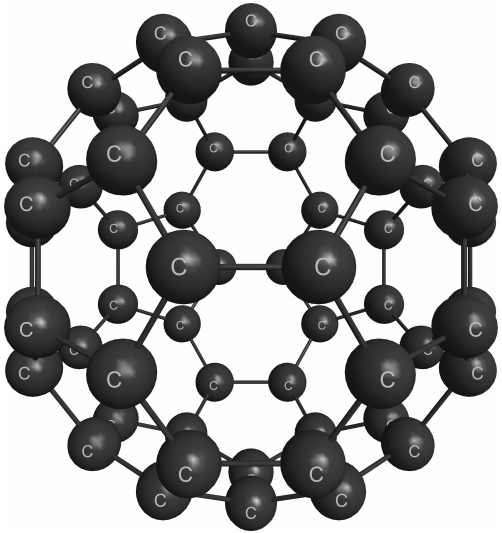
- GAMESS is the General Atomic and Molecular Electronic Structure System
  - General-purpose electronic structure code (many methods and capabilities)
  - ~1 million lines of Fortran
  - Optional C/C++ GPU-accelerated libraries/applications in GAMESS
- Scientific problem of interest
  - FMO/RI-MP2 calculations towards accurate simulations of catalysis reactions inside a mesopourous silica nanoparticle



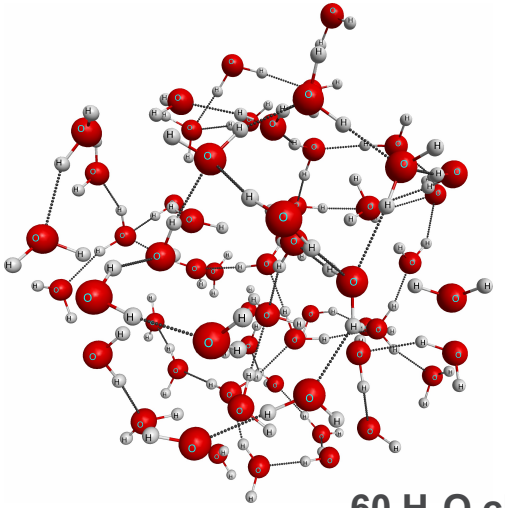
RI-MP2 MINI-APP EXAMPLES



Coronene



Fullerene



60 H_2O clusters

INTRO OF RI-MP2 MINI-APP

An ECP Proxy Application

- Computes RI-MP2 perturbative correction to the Hartree-Fock energy

$$E^{(2)} = \sum_{i \leq j}^{occ} (2 - \delta_{ij}) \sum_{ab}^{vir} \frac{(ia|jb)[2(ja|jb) - (ib|ja)]}{\epsilon_i + \epsilon_j - (\epsilon_a + \epsilon_b)}$$

INTRO OF RI-MP2 MINI-APP

An ECP Proxy Application

- Computes RI-MP2 perturbative correction to the Hartree-Fock energy

$$E^{(2)} = \sum_{i \leq j}^{occ} (2 - \delta_{ij}) \sum_{ab}^{vir} \frac{(ia|jb)[2(ja|jb) - (ib|ja)]}{\epsilon_i + \epsilon_j - (\epsilon_a + \epsilon_b)}$$

- Simplifies expression with the RI approximation letting it be written in terms of matrix multiplication

$$(ia|jb) = \sum_{n}^{aux} B^i_{an} B^j_{bn}$$

- Combine i,j, loops over occ orbitals, and then reduce over a,b, virtual orbitals to compute the final correlation energy

```
for(int JACT=0;JACT<NACT;JACT++){
  for(int IACT=0; IACT<=JACT;IACT++){

    // A kernel to compute QVV via Matrix Multiplication
    QVV[0:NVIR][0:NVIR] = B32[IACT][0:NVIR][0:NAUXBASD] *
                                  B32[JACT][0:NAUXBASD][0:NVIR];

    // Accumulating E2 using QVV[][], eij[][], and eab[][]
    for(int IB=0; IB<NVIR; IB++) {
      for(int IA=0; IA<NVIR; IA++) {
          Tijab = QVV[IB*NVIR][IA] /
                          ( eij[JACT][IACT] – eab[IB][IA]);
          Qt = QVV[IB][IA] + QVV[IB][IA];
          E2_t += Tijab * (Q_t – QVV[IA][IB]);
    }}
    FAC = (IACT==JACT) ? (1.0E0) : (2.0E0);
    E2 += E2_t *FAC;
}}
```

RI-MP2 CODE VERSIONS

- We constructed four RI-MP2 variants to explore with w25.rand.
- These variants add progressive/incremental levels of optimization as follow:
  - **V0-CPU**: an initial version,
  - **V1-CPU**: QVV computations use MKL DGEMM,
  - **V3-GPU**: offloaded V1-CPU to GPU using OpenMP Target offloading model
  - **V5-GPU:** QVV computation uses less MKL DGEMM calls by restructuring an outer loop; For E2, IACT loop is distributed subslices w/ “target teams distribute”, and then IB loop uses EUs w/ “parallel for” at each subslice, with optimal values for num_teams, and threads_limit

V0-CPU

```
double *QVV;
double E2_local=0.0E0;
QVV = new double[NVIR*NVIR];

for(int JACT=0;JACT<NACT;JACT++){
for(int IACT=0;IACT<=JACT;IACT++){

    // Compute QVV
    std::fill_n(QVV,NVIR*NVIR,0.0);
    for (int j = 0; j < NVIR; ++j) {
    for (int i = 0; i < NVIR; ++i) {
    for (int l = 0; l < NAUXBASD; ++l) {
       QVV(j,i) += B32(IACT,i,l)*B32(JACT,j,l);
    }}}

    // Accumulate E2
    double E2_t=0.0;
    for(int IB=0; IB<NVIR; IB++){
    for(int IA=0; IA<NVIR; IA++){
       double Tijab = QVV(IB,IA) / ( eij(JACT,IACT) - eab(IB,IA) );
       double Q_t = 2*QVV(IB,IA);
       E2_t += Tijab * (Q_t - QVV(IA,IB) );
    }}   // loop for IA and IB
    double FAC = (IACT==JACT) ? (1.0E0) : (2.0E0);
    E2_local += FAC*E2_t;
}}   // loop for IACT and JACT

*E2 = *E2 + E2_local;
delete[] QVV;
```

- **QVV: Computing QVV**
- **E2:E2 accumulation**

V0-CPU

- QVV: Computing QVV
- E2:E2 accumulation

```
double *QVV;
double E2_local=0.0E0;
QVV = new double[NVIR*NVIR];

for(int JACT=0;JACT<NACT;JACT++){
for(int IACT=0;IACT<=JACT;IACT++){
```

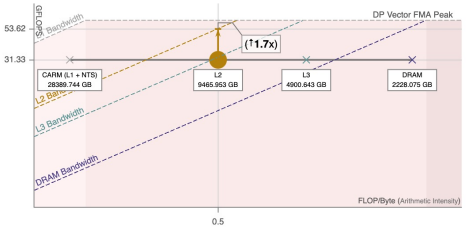
QVV

```
// Compute QVV
std::fill_n(QVV,NVIR*NVIR,0.0);
for (int j = 0; j < NVIR; ++j) {
for (int i = 0; i < NVIR; ++i) {
for (int l = 0; l < NAUXBASD; ++l) {
   QVV(j,i) += B32(IACT,i,l)*B32(JACT,j,l);
}}}
```

E2

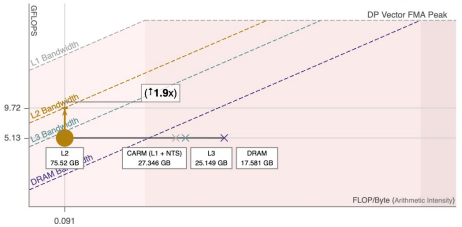
```
// Accumulate E2
double E2_t=0.0;
for(int IB=0; IB<NVIR; IB++){
for(int IA=0; IA<NVIR; IA++){
   double Tijab = QVV(IB,IA) / ( eij(JACT,IACT) - eab(IB,IA) );
   double Q_t = 2*QVV(IB,IA);
   E2_t += Tijab * (Q_t - QVV(IA,IB) );
}}   // loop for IA and IB
double FAC = (IACT==JACT) ? (1.0E0) : (2.0E0);
E2_local += FAC*E2_t;
}}  // loop for IACT and JACT

*E2 = *E2 + E2_local;
delete[] QVV;
```

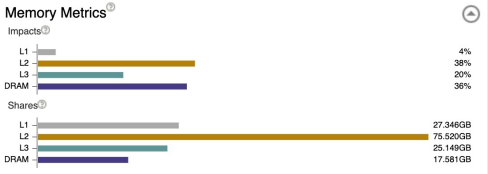
- QVV: L2 cache bound
  - 31.33 GF/s
  - L2 AI= 0.5



Data traffic (Bytes) ratios:
L2 : L3: DRAM = 4.2 : 2.2 : 1

- E2: L2 cache bound
  - 5.13 GF/s
  - L2 AI= 0.091





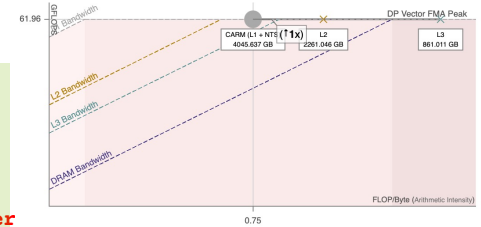
Data traffic (Bytes) ratios:
L2 : L3: DRAM = 4.3 : 1.4 : 1

V1-CPU

- QVV: Computing QVV using MKL DGEMM
- E2:E2 accumulation

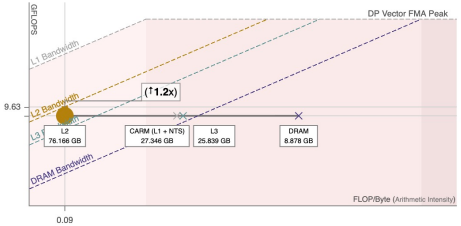
```
double *QVV;
double E2_local=0.0E0;
QVV = new double[NVIR*NVIR];

for(int JACT=0;JACT<NACT;JACT++){
for(int IACT=0;IACT<=JACT;IACT++){

    // Compute QVV
QVV int n=NVIR;
    int k=NAUXBASD;
    double one = 1.0;
    double zero = 0.0;
dgemm("T","N",&n,&n,&k,&one,&B32(IACT,0,0),&k,&B32(JACT,0,0),&k,&zero,QVV,&n);

    // Accumulate E2
E2  double E2_t=0.0;
    for(int IB=0; IB<NVIR; IB++){
    for(int IA=0; IA<NVIR; IA++){
       double Tijab = QVV(IB,IA) / ( eij(JACT,IACT) - eab(IB,IA) );
       double Q_t = 2*QVV(IB,IA);
       E2_t += Tijab * (Q_t - QVV(IA,IB) );
    }}   // loop for IA and IB
    double FAC = (IACT==JACT) ? (1.0E0) : (2.0E0);
    E2_local += FAC*E2_t;
}}   // loop for IACT and JACT

*E2 = *E2 + E2_local;
delete[] QVV;
```

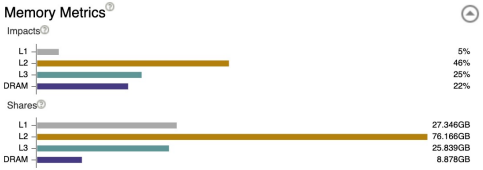
- QVV: COMPUTE DP FMA bound
  - 61.96 GF/s
  - L2 AI= 1.34



Data traffic (Bytes) ratios:
L1 : L2: L3 = 4.7 : 2.6 : 1
NO DRAM at all! Fitting into the cache completely!

- E2: L2 cache bound
  - 7.95 GF/s
  - L2 AI= 0.090

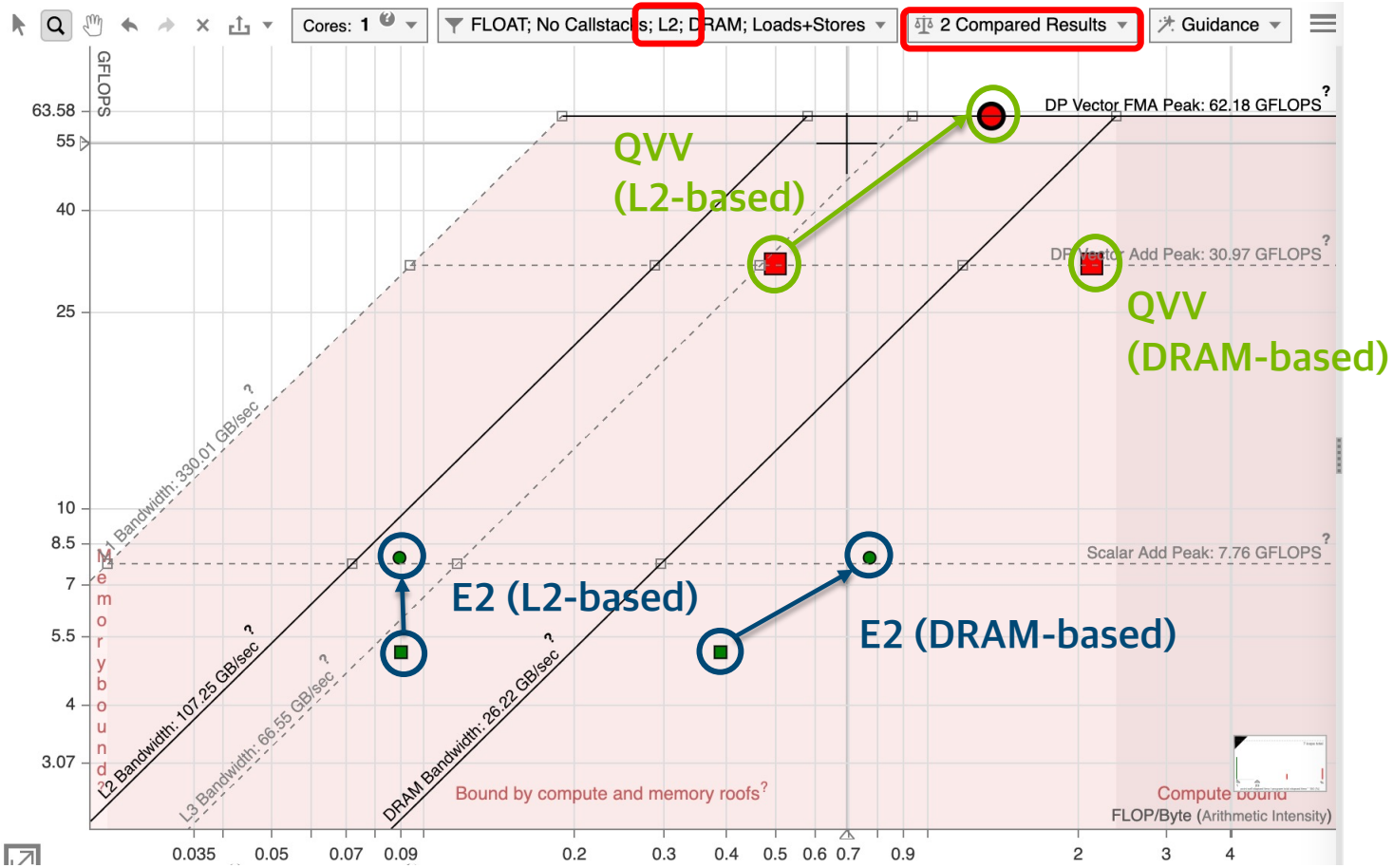




Data traffic (Bytes) ratios:
L2 : L3: DRAM = 8.6 : 2.9 : 1

V0-CPU to V1-CPU

- **QVV computation is compute-bound (by MKL) w/ much lower memory pressure; it consequently increases performance of E2.**



V3-GPU: Initial CPU-like version

```
double *QVV;
double E2_local=0.0E0;
int dnum=0;
QVV = new double[NVIR*NVIR];
double *B32I, *B32J;

#pragma omp target enter data map(alloc:QVV[0:NVIR*NVIR])
device(dnum)
#pragma omp target enter data
map(to:eij[0:NACT*NACT],eab[0:NVIR*NVIR],B32[0:B32size])
device(dnum)
for(int JACT=0;JACT<NACT;JACT++){
for(int IACT=0;IACT<=JACT;IACT++){

    // Compute QVV
    int n=NVIR;
    int k=NAUXBASD;
    double one = 1.0;
    double zero = 0.0;
    B32I = &B32(IACT,0,0);
    B32J = &B32(JACT,0,0);
    #pragma omp target variant dispatch
use_device_ptr(B32I,B32J,B32,QVV) device(dnum)
    dgemm("T","N",&n,&n,&k,&one,B32I,&k,B32J,&k,&zero,QVV,&n);

    // Accumulate E2
    double E2_t=0.0;
    #pragma omp target teams distribute parallel for
reduction(+:E2_t) map(tofrom:E2_t) collapse(2) device(dnum)
    {
        for(int IB=0; IB<NVIR; IB++){
        for(int IA=0; IA<NVIR; IA++){
           double Tijab = QVV(IB,IA) / (eij(JACT,IACT)-eab(IB,IA));
           double Q_t = 2*QVV(IB,IA);
           E2_t += Tijab * (Q_t - QVV(IA,IB) );
        }}   // loop for IA and IB
    }   // omp target teams distribute parallel for
    double FAC = (IACT==JACT) ? (1.0E0) : (2.0E0);
    E2_local +=  FAC*E2_t;
}}   // loop for IACT and JACT

*E2 = *E2 + E2_local;
delete[] QVV;

#pragma omp target exit data map(release:QVV[0:NVIR*NVIR])
device(dnum)
#pragma omp target exit data
map(release:eij[0:NACT*NACT],eab[0:NVIR*NVIR],B32[0:B32size])
device(dnum)
```

- Computing QVV using MKL DGEMM w/ GPU offloading
- E2 accumulation w/ GPU offloading

V3-GPU: initial CPU-like version

- QVV: DP FMA bound
  - 127.25 GF/s
  - L3 AI= 1.14

SUMMARY

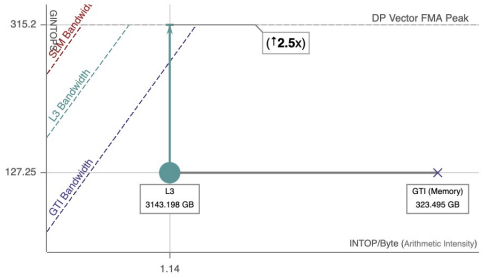
Elapsed Time	28.11s	GINTOPS	0.50
		GFLOPS	127.25
Global	80 x 64	Local	8 x 16

INSTRUCTION MIX DETAILS

Compute	236.22	88%
Memory	11.80	4%
Other	19.46	7%

ROOFLINE GUIDANCE

This kernel is bounded by the DP Vector FMA Peak



MEMORY METRICS

Impacts

L3	84%
GTI (Memory)	15%

Shares

L3	3143.198GB
GTI (Memory)	323.495GB

PERFORMANCE CHARACTERISTICS

Active:	63.1%
Stalled:	11.0%
Idle:	26.0%

SIMD Width:	8
EU Threading Occupancy:	62.4%
2 FPUs Active:	56.1%

- E2: L3 GPU cache bound
  - 1.41 GF/s
  - L3 AI= 0.071

SUMMARY

Elapsed Time	23.06s	GINTOPS	0.84
		GFLOPS	1.41
Global	6480		

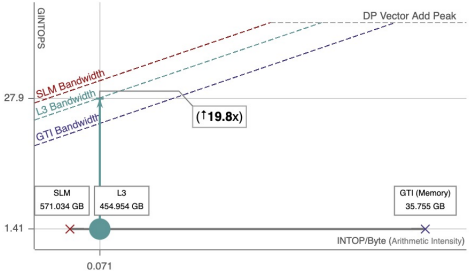
INSTRUCTION MIX DETAILS

Atomic	19.14	10%
Compute	19.40	10%
Memory	0.65	<1%
Other	159.36	80%

Instruction	Count
MOVE	90.04
OTHER	19.61
CONTROL FLOW	49.71

ROOFLINE GUIDANCE

This kernel is bounded by the L3 Bandwidth
Improve cache line utilization by optimizing memory access pattern.
Current cache line utilization is **6%**. It means that the number of bytes used by an execution unit is less than the number of bytes transferred to the L3 cache.



MEMORY METRICS

Impacts

L3	50%
SLM	42%
GTI (Memory)	7%

Shares

L3	454.954GB
SLM	571.034GB
GTI (Memory)	35.755GB

PERFORMANCE CHARACTERISTICS

Active:	13.1%
Stalled:	78.0%
Idle:	8.9%

SIMD Width:	16
EU Threading Occupancy:	57.4%
2 FPUs Active:	0.5%

V5-GPU: restructured loops for better GPU performance

```
double *QVV;
double E2_local=0.0E0;
int dnum=0;
QVV = new double[NVIR*NACT*NVIR];
double *B32J;

#pragma omp target enter data map(alloc:QVV[0:NVIR*NACT*NVIR])
device(dnum)
#pragma omp target enter data
map(to:eij[0:NACT*NACT],eab[0:NVIR*NVIR],B32[0:B32size]) device(dnum)
for(int JACT=0;JACT<NACT;JACT++){

    // Compute QVV
    int m=NVIR*(JACT+1);
    int n=NVIR;
    int k=NAUXBASD;
    double one = 1.0;
    double zero = 0.0;
    B32J = &B32(JACT,0,0);
    #pragma omp target variant dispatch use_device_ptr(B32,B32J,QVV)
device(dnum)
    dgemm("T","N",&m,&n,&k,&one,B32,&k,B32J,&k,&zero,QVV,&m);

    // Accumulate E2
    #pragma omp target teams distribute reduction(+:E2_local)
num_teams(90) thread_limit(72) device(dnum)
    for(int IACT=0; IACT<=JACT; IACT++){
       double E2_t=0.0;
       #pragma omp parallel for reduction(+:E2_t)
       for(int IB=0; IB<NVIR; IB++){
```

```
       for(int IA=0; IA<NVIR; IA++){
          double Tijab = QVV(IB,IACT,IA) / ( eij(JACT,IACT) -
eab(IB,IA) );
          double Q_t = 2*QVV(IB,IACT,IA);
          E2_t += Tijab * (Q_t - QVV(IA,IACT,IB) );
       }}   // loop for IA and IB
       double FAC = (IACT==JACT) ? (1.0E0) : (2.0E0);
       E2_local +=  FAC*E2_t;
    }   // loop for IACT
}   // loop for JACT

*E2 = *E2 + E2_local;
delete[] QVV;

#pragma omp target exit data map(release:QVV[0:NVIR*NACT*NVIR])
device(dnum)
#pragma omp target exit data
map(release:eij[0:NACT*NACT],eab[0:NVIR*NVIR],B32[0:B32size])
device(dnum)
```

- Computing QVV with less MKL DGEMM offloading calls on GPU by restructuring an outer loop
- E2 accumulation w/ hierarchial GPU offloading (IACT-loop to Subslices (SS), and IB-loop to Execution Units (EUs)

V5-GPU: restructured loops for better GPU performance

- QVV: DP FMA bound
  - 262.84 GF/s
  - L3 AI= 1.66

SUMMARY

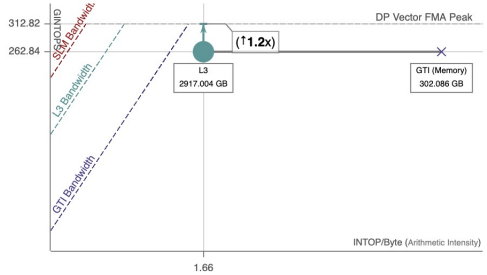
Elapsed Time	18.38s	GINTOPS	1.05
		GFLOPS	262.84
Global	120 x 64, 128 x 64, 136 x 6...	Local	8 x 16

INSTRUCTION MIX DETAILS

Compute	319.21	88%
Memory	16.01	4%
Other	26.38	7%

ROOFLINE GUIDANCE

This kernel is bounded by the DP Vector FMA Peak



MEMORY METRICS

Impacts

L3	84%
GTI (Memory)	15%

Shares

L3	2917.004GB
GTI (Memory)	302.086GB

PERFORMANCE CHARACTERISTICS

Active:	91.8%
Stalled:	3.2%
Idle:	5.0%

SIMD Width:	8
EU Threading Occupancy:	91.7%
2 FPUs Active:	89.0%

- E2: L3 GPU cache bound
  - 37.0 GF/s
  - L3 AI= .019

SUMMARY

Elapsed Time	0.86s	GINTOPS	14.37
		GFLOPS	37.00
Global	6480		

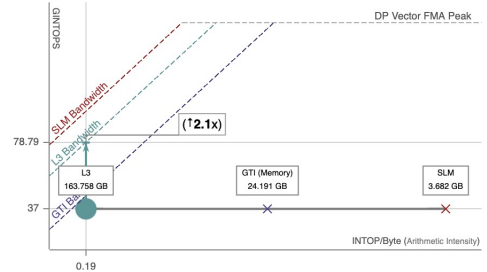
INSTRUCTION MIX DETAILS

Atomic	0.07	<1%
Compute	7.26	64%
Memory	0.43	4%
Other	3.50	31%

Instruction	Count
MOVE	2.11
SYNC	<0.01
CONTROL FLOW	1.10
OTHER	0.29

ROOFLINE GUIDANCE

This kernel is bounded by the L3 Bandwidth
Improve cache line utilization by optimizing memory access pattern.
Current cache line utilization is **16%**. It means that the number of bytes used by an execution unit is less than the number of bytes transferred to the L3 cache.



MEMORY METRICS

Impacts

L3	77%
SLM	1%
GTI (Memory)	21%

Shares

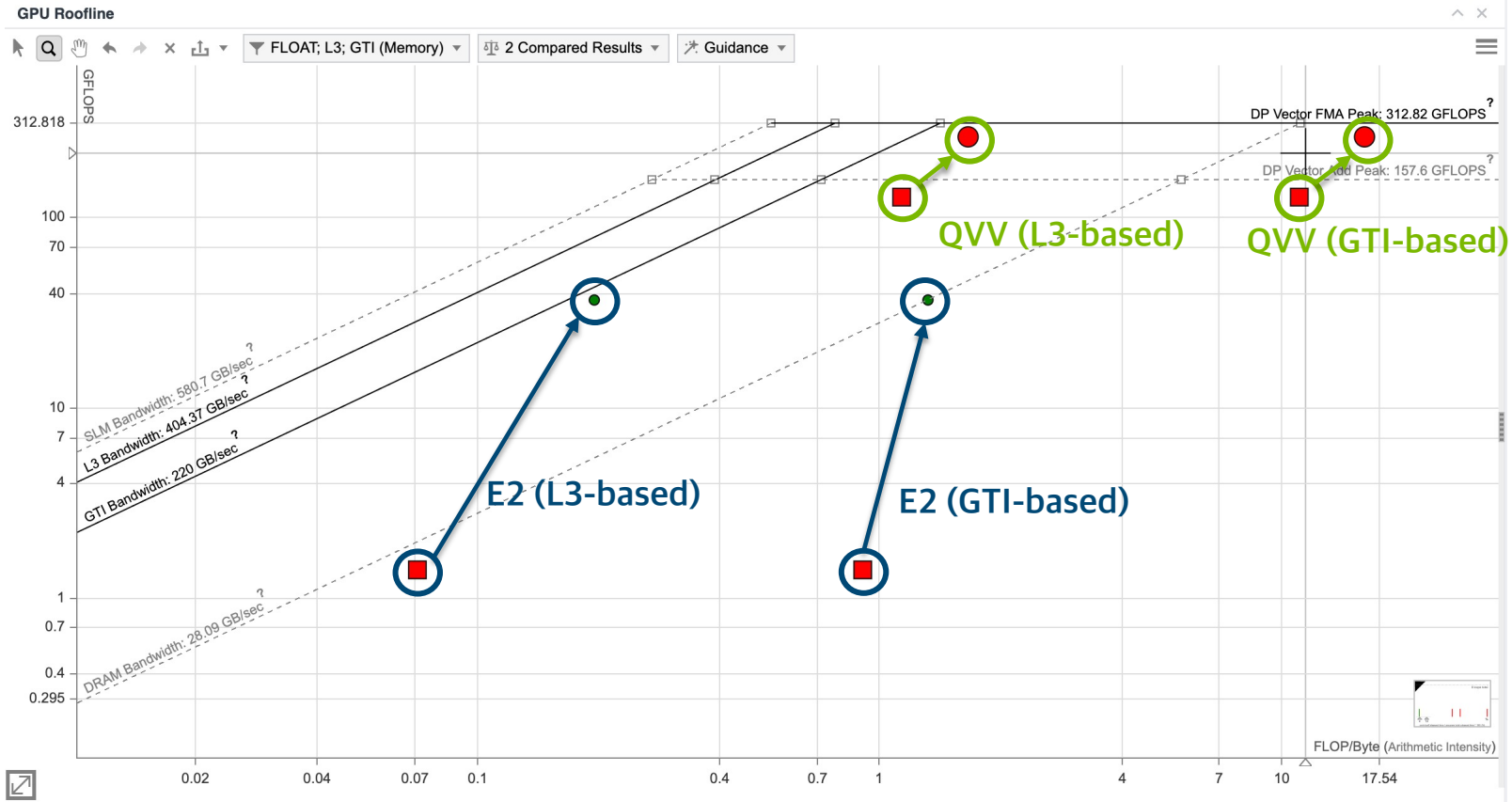
L3	163.758GB
SLM	3.682GB
GTI (Memory)	24.191GB

PERFORMANCE CHARACTERISTICS

Active:	47.2%
Stalled:	46.4%
Idle:	6.4%

SIMD Width:	8
EU Threading Occupancy:	75.4%
2 FPUs Active:	12.6%

Comparison from V3-GPU to V5-GPU

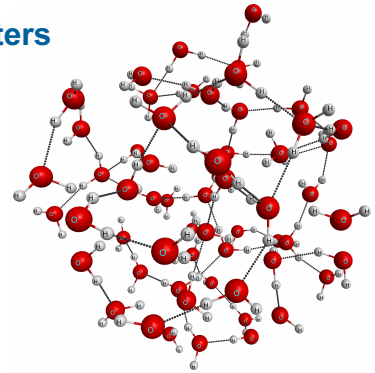


PERFORMANCE RESULTS

Tested Input: w25.rand (random data with structure of 25 H_2O clusters)
Employed Compute Node: Intel(R) Xeon(R) CPU E3-1585 v5

- CPU: Intel Xeon Skylake @ 3.5GHz (4 cores)
- GPU: Intel Gen9 GT4e @ 1.15GHz (72 EUs)

H_2O clusters



Selected versions	QVV		E2		Overall	
	Target	Time(s)	Target	Time(s)	Time(s)	Speedup over V00
V0-CPU	CPU	156.68	CPU	1.34	158.02	1.0x
V1-CPU	CPU	86.41	CPU	0.92	87.33	1.8x
V3-GPU	GPU	32.38	GPU	23.06	55.44	2.9x
V5-GPU	GPU	21.49	GPU	0.86	22.35	7.1x

QUICK OVERVIEW OF ARGONNE ROOFLINE USE-CASES

ARGONNE ROOFLINE USE-CASES

Selected Argonne Applications for Roofline Use-cases

Application	ANL PoC	Programming models	Field of Science
NekBench	Kris Rowe	OCCA with OpenCL backend	Computational Fluid Dynamics
XSBench/RSBench	John Tramm	OpenMP Target	Monte Carlo neutron transport
AMR-Wind	JaeHyuk Kwack	SYCL/DPC++	Wind farm simulation (CFD/FSI)

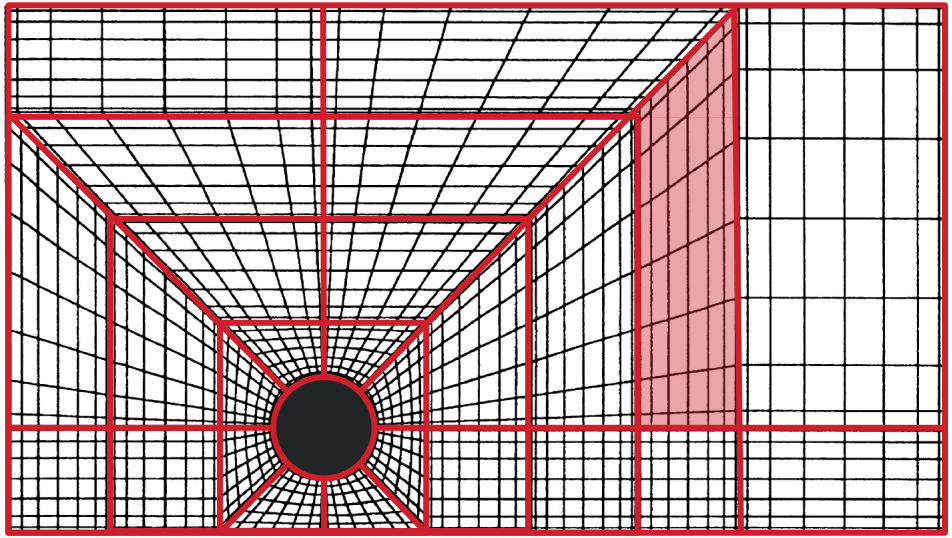
NEKBENCH ON INTEL GEN9 GPU

BY KRIS ROWE (ANL)

NEKRS

Helmholtz Operator

- NekRS is a port of Nek5000 to GPUs
- Uses the OCCA portability library
- Several iterative solves per time step
  - 5-20+ iterations per solve
- Each iteration requires evaluation of the Helmholtz Operator
  - Local stencil is dense within each element
  - Halo exchange between elements



NEKBENCH::AXHELM

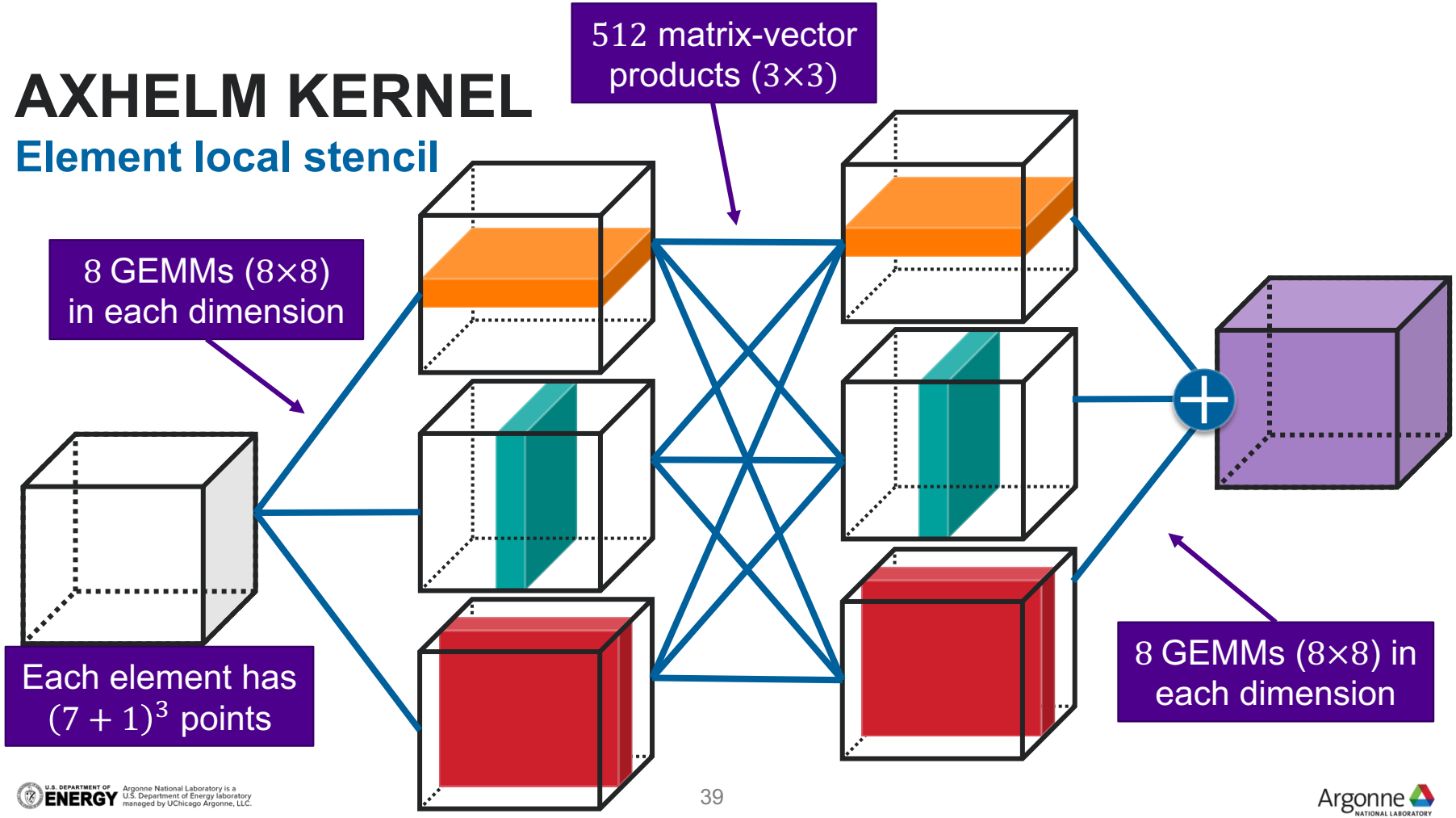
Local Helmholtz Operator Kernel Benchmark

- NekBench contains kernel and mini-app benchmarks relevant to NekRS
- The axhelm kernel evaluates the local Helmholtz operator stencil—i.e.
  - without MPI communication
  - without mesh topology/connectivity
- Benchmark repeatedly evaluates the kernel for a prescribed number of repetitions
- OCCA OpenCL backend was used
- Five different kernel implementations were tested

Benchmark Parameters		
P	7	Polynomial Order
E	1600	Number of elements
R	1000	Number of trials

AXHELM KERNEL

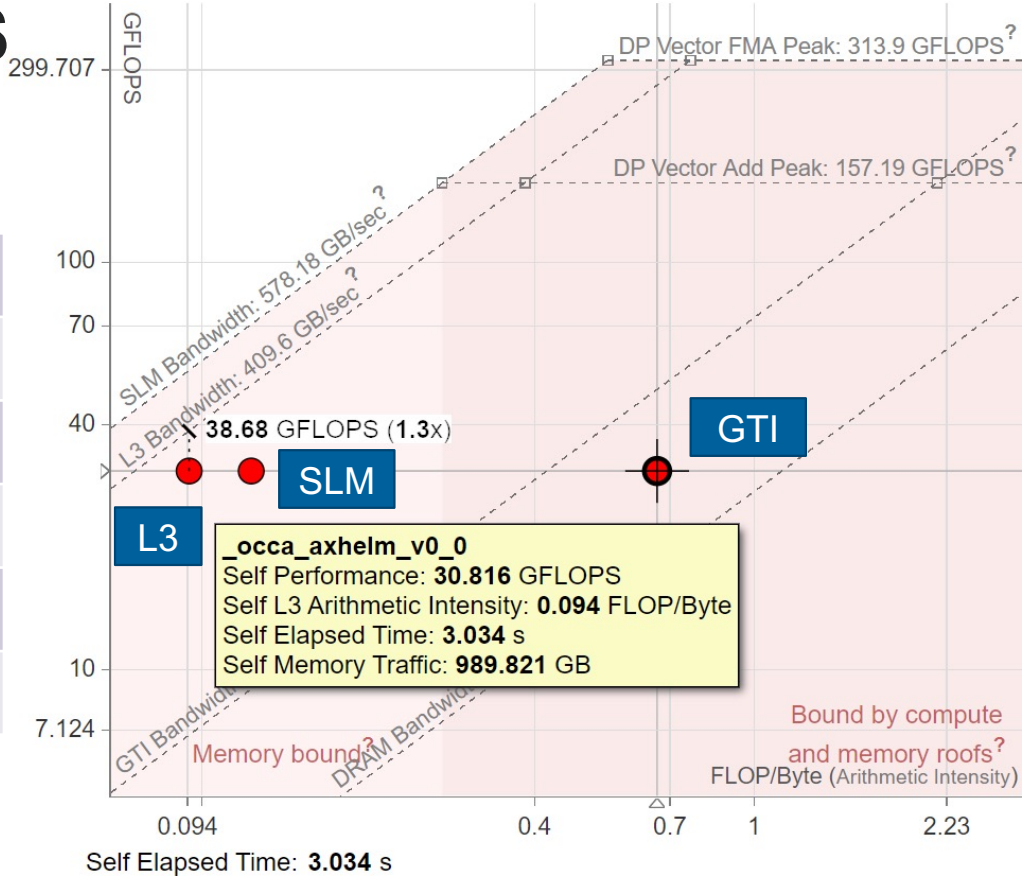
Element local stencil



ROOFLINE ANALYSIS

Kernel Version 0

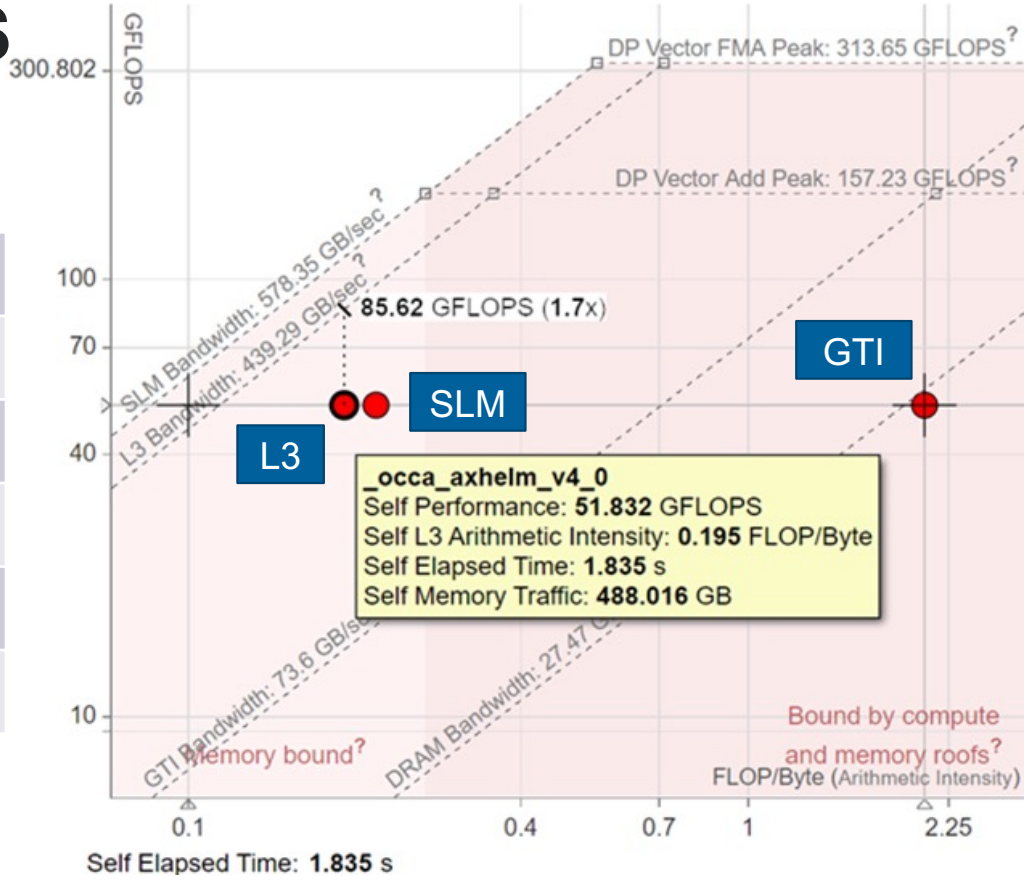
Work Group	8×8
# Work Groups	1600×1
Strategy	2D slices
Inputs	global → SLM
Work Arrays	SLM
Outputs	registers → global



ROOFLINE ANALYSIS

Kernel Version 4

Work Group	8×8×4
# Work Groups	1600×1×1
Strategy	half-cube, symmetry
Inputs	global → SLM
Work Arrays	SLM
Outputs	registers → global



NEKBENCH ROOFLINE ANALYSIS

Comparison of Results

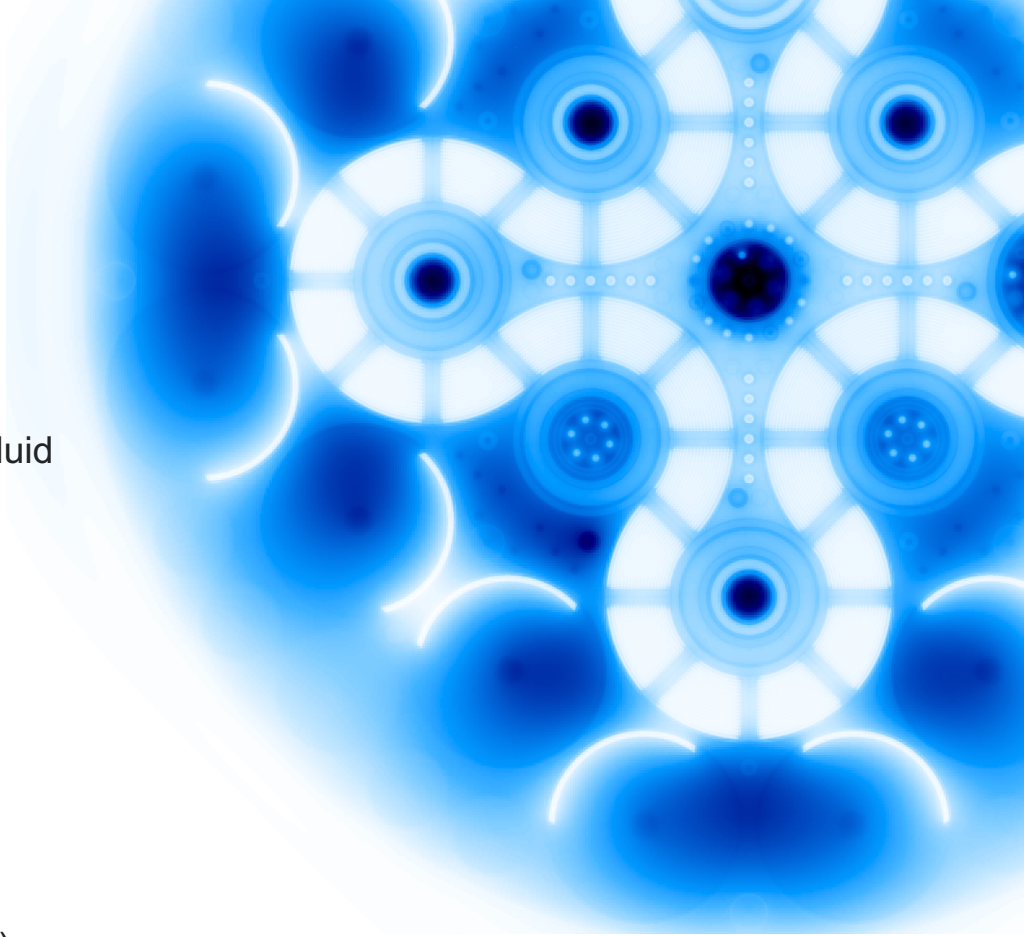
- Both kernel versions get mapped to SIMD width 16 on Gen9
- Number of operations (FP, INT, mixed) is nearly identical for both kernels
  - The AI for each type is 2-4x larger in kernel v4.
- Both kernels are memory bound for L3 and SLM
- Kernel v0 loads more data from global, SLM memory; calls more barriers
- Kernel v4 uses more SLM, registers—leads to lower occupancy

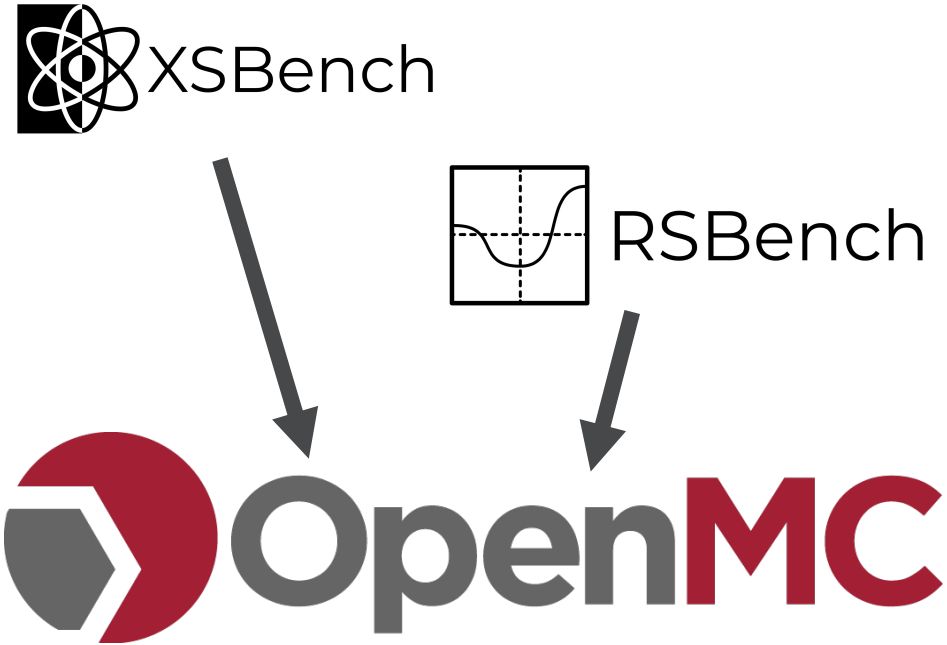
XSBENCH/RSBENCH ON INTEL GEN9 GPU

BY JOHN TRAMM (ANL)

OpenMC

- Full Application
- Science: Monte Carlo particle transport
- Exascale Challenge Problem:
  - Full core nuclear reactor simulation
  - Multiphysics coupling with computational fluid dynamics code (Nek5000)
- Code Info:
  - Open Source
  - C++
- Key Kernels:
  - Cross section lookups
  - Tallying
- State of the code:
  - MPI + OpenMP Threading
  - Undergoing port to GPU (via OMP Offload)



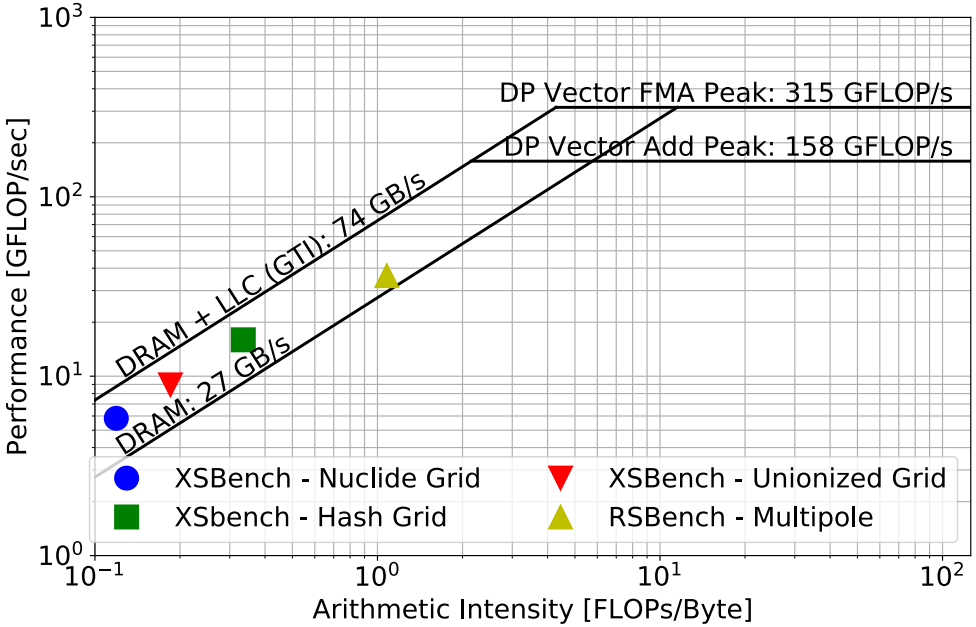


- XSBench and RSBench are **mini-apps** representing key kernels from the full application OpenMC
- Ported both mini-apps to:
  - OpenMP Threading
  - OpenMP Offload
  - OpenCL
  - SYCL
  - CUDA
- Mini-apps contain "baked in" default test problems for performance analysis
- No dependencies, making them very easy to compile/run

ROOFLINE ANALYSIS ON GEN9 WITH OPENMP

	Memory Usage [MB]	Runtime [s]	Intensity	GFLOP/s
RSBench – Multipole	26	61.0	1.08	36.4
XSBench – Nuclide Grid Search	139	15.8	0.12	5.8
XSBench – Logarithmic Hash Grid Search	152	5.8	0.34	16.0
XSBench – Unionized Grid Search	4,248	5.3	0.19	8.9

- The different algorithms represent different methods of accomplishing the same task
- They trade reductions in memory footprint for increases in floating point work
- These results show that for this task, it's more advantageous to minimize work rather than maximize efficiency



Run as:

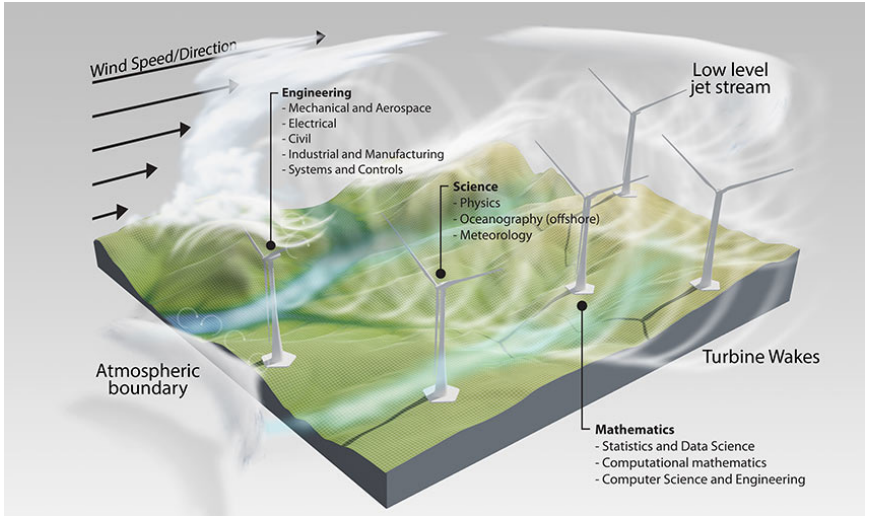
```
./rsbench -m event -l 17000000
./XSBench -m event -g 8500 -G <grid_type>
```

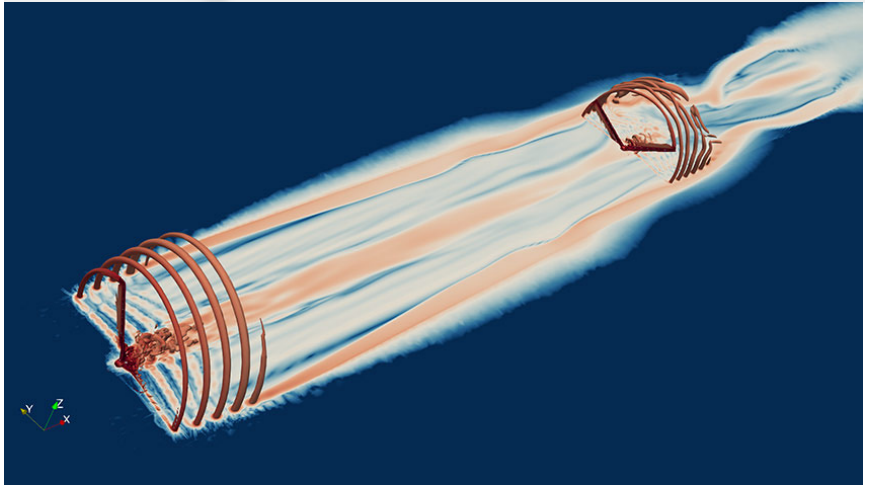
AMR-WIND ON INTEL GEN9 GPU

BY JAEHYUK KWACK

AMR-WIND

- One of Physics modules of ECP ExaWind project
- A massively parallel, block-structured adaptive-mesh CFD code
  - An incompressible Navier-Stokes solver
  - Including neutral ABL (Atmospheric Boundary Layer) physics
  - A background solver when coupled with a near-body solver (e.g., Nalu-Wind) with overset methodology to perform blade-resolved simulations of multiple wind turbines within a wind farm.
- Aurora programming model: DPC++
- Dependency: AMReX
- Key development teams
  - NREL (ExaWind project): Mike Sprague, Jon Rood, Paul Mullowney
  - LBL (AMReX project): Weiqun Zhang





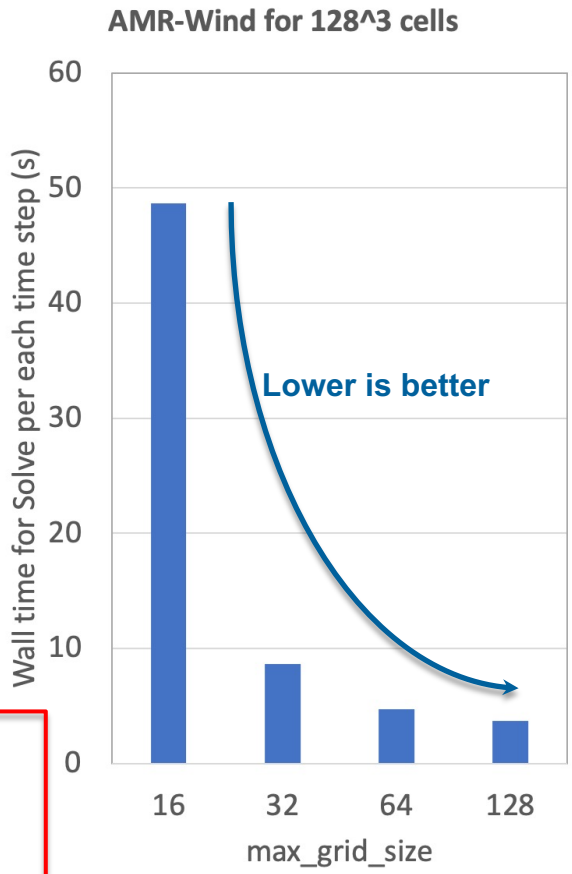
(Image credit: NREL)

AMR-WIND ON INTEL GPU

At the 5th time step

- AMR-Wind version :: 6af41101-DIRTY
- AMReX version :: 21.04
- Input: abl_godunov.i
  - amr.n_cell=128 128 128
  - amr.max_grid_size = {16, 32, 64, 128}
  - time.max_step = 5
- Wall time at 5th time step

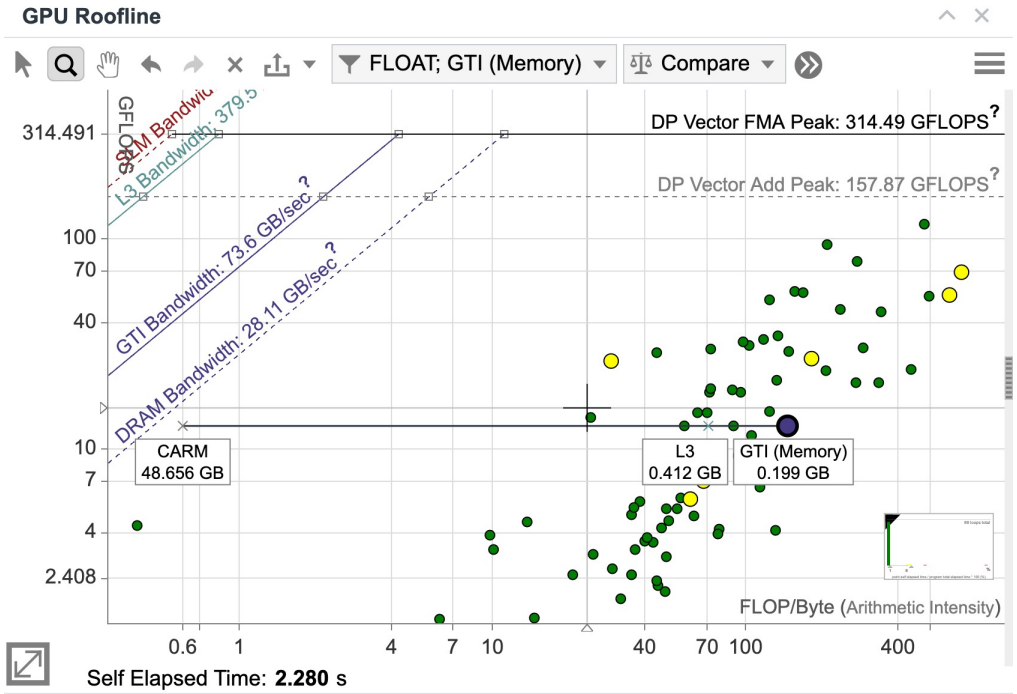
max_grid_size	16	32	64	128
WT_Pre	0.0745	0.00964	0.0057	0.00452
WT_Solve	48.65	8.667	4.724	3.707
WT_Post	3.76	2.26	1.78	1.54
WT_Total	52.49	10.94	6.514	5.248

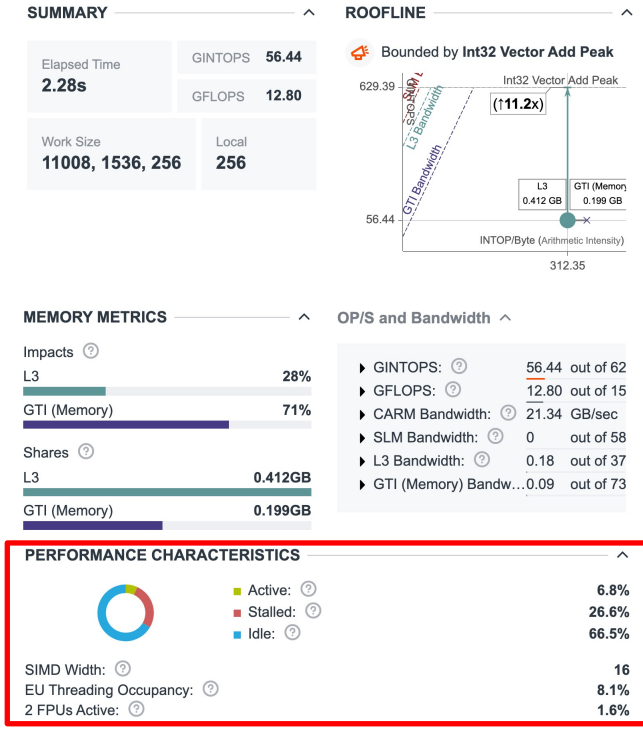


Repeated process

ADVISOR ROOFLINE RESULTS

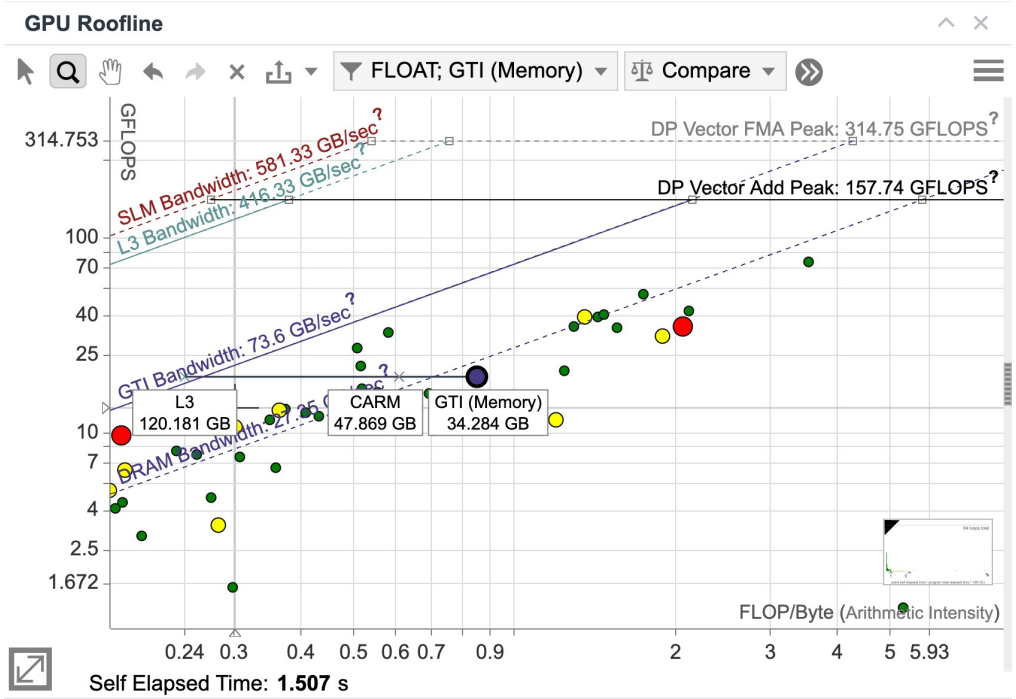
max_grid_size = 16 for 128^3 cells (e.g., MLPoisson::Fsmooth)

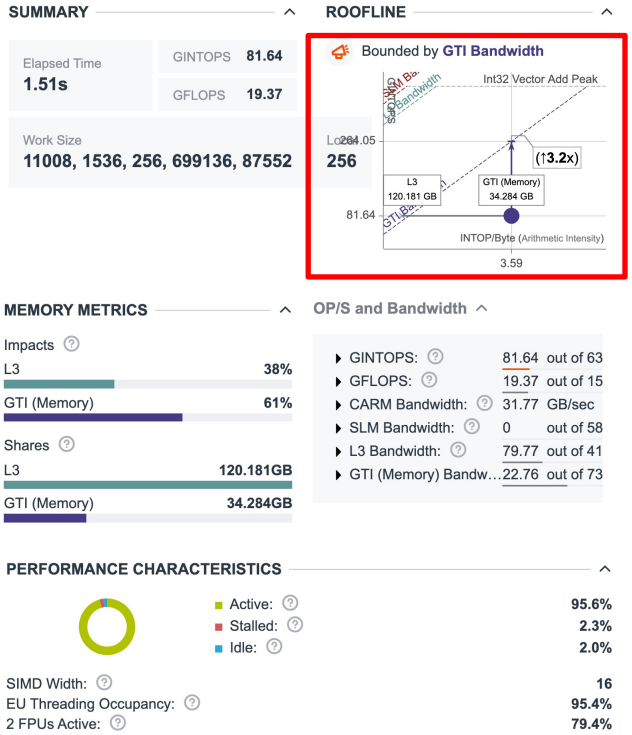




ADVISOR ROOFLINE RESULTS

max_grid_size = 128 for 128^3 cells (e.g., MLPoisson::Fsmooth)





CONCLUDING REMARKS

CONCLUDING REMARKS

- Argonne application developers and other US DOE collaborators have been actively porting their applications to Aurora testbed systems with Intel oneAPI toolkits for the coming Aurora Exa-scale supercomputer at Argonne National Laboratory.
- Intel Advisor successfully provides the detailed performance data via roofline analysis features on both Intel CPUs and GPUs.
- Call-to-actions
  - If you are working on ECP projects and interested in Aurora testbeds with Intel oneAPI toolkit, you can get the access via https://www.jlse.anl.gov/accessing-jlse-resources/.
  - If you are interested in Intel GPUs with Intel oneAPI toolkits for your own applications, you may try the Intel DevCloud system via the following link:
    - https://www.intel.com/content/www/us/en/developer/tools/devcloud/overview.html
  - Enjoy!

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 **Colleen Bertoni, Kris Row, and John Tramm** at Argonne National Laboratory for sharing Argonne use-cases, **Zakhar Matveev**, **Kirill Rogozhin, Yuila Efimenko,** and **Ekaterina Guseva** for **Advisor** and **Louise Huot** for **MKL** for providing technical supports for issues, reviewing data and providing feedback for this study.

THANKS!