Performance Optimization of ECTrans on AMD GPUs

Paul Mullowney
Data Center GPU Center of Excellence AMD
20th ECMWF Workshop in HPC in Meteorology
October 11, 2023

together we advance_

Agenda

- AMD GPUs
- ECTrans
- AMD profiling tools
- Optimizing ECTrans on AMD GPUs
- Prospects on upcoming architectures
- Summary

CSC
Kungl. Tekniska Högskolan
NORTHERN DATA
GENCI
Washington University in St. Louis School of Medicine
ILLINOIS NCSA
NIH
ARL
FIAS Frankfurt Institute for Advanced Studies
eni
Microsoft Azure
CERN
kt
OAK RIDGE National Laboratory
Lawrence Livermore National Laboratory
CGG
PAWSEY supercomputing centre

AMD INSTINCT MI250X

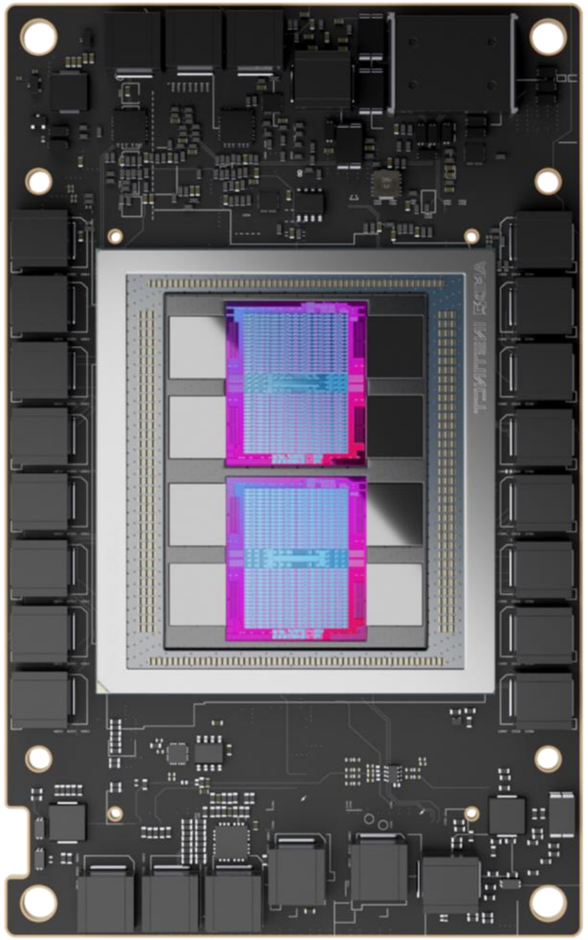
WORLD'S FIRST EXASCALE GPU

OCP ACCELERATOR MODULE (OAM)

220 COMPUTE UNITS

880 MATRIX CORES

128GB HBM2E 3.2TB/s BANDWIDTH



UP TO 8 EXTERNAL I/O LINKS

5 GPU-TO-GPU INFINITY FABRIC LINKS

2 COHERENT CPU-TO-GPU LINKS

1 LINK CONFIGURED AS PCI EXPRESS

AMD – ECMWF Collaboration

- Started in early 2023 : Objectives
  - Help improve the performance of the existing OpenMP® (and OpenACC) offloading implementation in critical codes like ECTrans/CloudSC
  - Build enough knowledge of the code bases to effectively help the porting to next generation devices
  - The vast majority of the porting work was done by ECMWF team

- Why OpenMP®
  - Internal AMD Fortran compiler development currently focuses on OpenMP®
    - Performance Portable and Productivity
    - AMD has not prioritized OpenACC
  - HPE Cray has been supporting both OpenACC and OpenMP® in their Fortran compiler
    - Same backend for OpenMP® /OpenACC accelerated code
  - Managed memory single address space (XNack)
    - Elimination of data movement pragmas

AMD Profilers

ROC-profiler (rocprof)

Hardware Counters

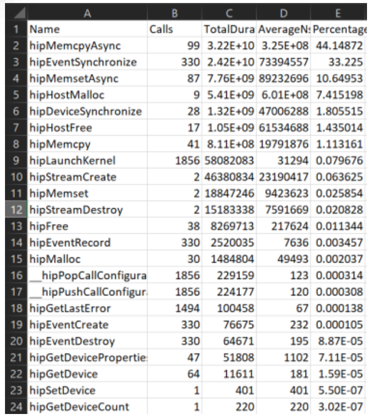
- Raw collection of GPU counters and traces
- Counter collection with user input files
- Counter results printed to a CSV

Traces and timelines

- Trace collection support for
- CPU copy
- HIP API
- HSA API
- GPU Kernels

Visualization

- Traces visualized with Perfetto



	A	B	C	D	E
1	Name	Calls	TotalDura	AverageN:	Percentage
2	hipMemcpyAsync	99	3.22E+10	3.25E+08	44.14872
3	hipEventSynchronize	330	2.42E+10	73394557	33.225
4	hipMemsetAsync	87	7.76E+09	89232696	10.64953
5	hipHostMalloc	9	5.41E+09	6.01E+08	7.415198
6	hipDeviceSynchronize	28	1.32E+09	47006288	1.805515
7	hipHostFree	17	1.05E+09	61534688	1.435014
8	hipMemcpy	41	8.11E+08	19791876	1.113161
9	hipLaunchKernel	1856	58082083	31294	0.079676
10	hipStreamCreate	2	46380834	23190417	0.063625
11	hipMemset	2	18847246	9423623	0.025854
12	hipStreamDestroy	2	15183338	7591669	0.020828
13	hipFree	38	8269713	217624	0.011344
14	hipEventRecord	330	2520035	7636	0.003457
15	hipMalloc	30	1484804	49493	0.002037
16	__hipPopCallConfigura	1856	229159	123	0.000314
17	__hipPushCallConfigur	1856	224177	120	0.000308
18	hipGetLastError	1494	100458	67	0.000138
19	hipEventCreate	330	76675	232	0.000105
20	hipEventDestroy	330	64671	195	8.87E-05
21	hipGetDevicePropertie	47	51808	1102	7.11E-05
22	hipGetDevice	64	11611	181	1.59E-05
23	hipSetDevice	1	401	401	5.50E-07
24	hipGetDeviceCount	1	220	220	3.02E-07

Omnitrace

Trace collection

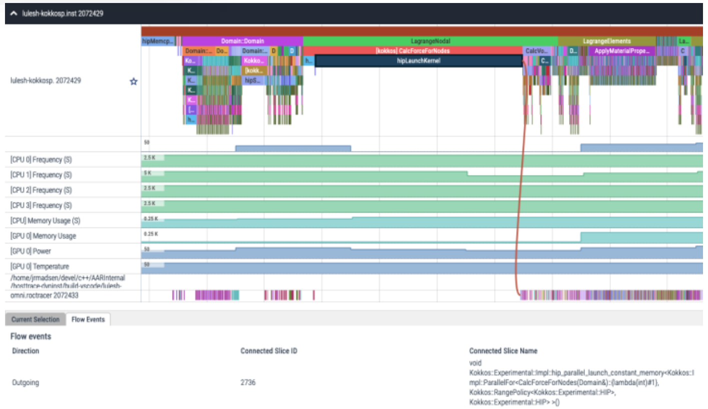
- Comprehensive trace collection
- CPU
- GPU

Supports

- CPU copy
- HIP API
- HSA API
- GPU Kernels
- OpenMP®
- MPI
- Kokkos
- p-threads
- multi-GPU

Visualization

- Traces visualized with Perfetto



Flow events

```
Direction          Connected Slice ID          Connected Slice Name
                                               void
Outgoing           2736                        Kokkos::Experimental::Impl::hip_parallel_launch_constant_memory<Kokkos::I
                                               mpl::ParallelFor<CalcForceForNodes(Domain&)::{lambda(int)#1},
                                               Kokkos::RangePolicy<Kokkos::Experimental::HIP>,
                                               Kokkos::Experimental::HIP> >()
```

Omniperf

Performance Analysis

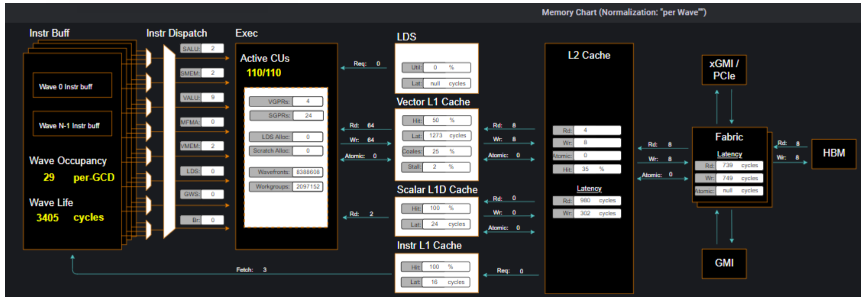
- Automated collection of hardware counters
- Analysis
- Visualization

Supports

- Speed of Light
- Memory chart
- Rooflines
- Kernel comparison

Visualization

- With Grafana or standalone GUI



AMD Profilers/References

AMD Develops several profiling tools to suit different needs AMD Lab Notes – Profilers

- ROC-profiler
  - Low level API for detailed kernel performance breakdowns.
  - Can be used with ROC-tracer/ROC-TX libraries to collect application timeline traces and User Annotated Code Regions
- Omniperf
  - High level interface to detailed kernel performance breakdowns using a wide range of hardware counters
  - Web-based GUI or command line interface (CLI)
  - Open-source project and not an official part of the ROCm stack. Users feedback, contributions, and issue submission are encouraged.
- Omnitrace
  - Comprehensive profiling tool for profiling/tracing tool for parallel application
  - Ideal tool for characterizing where optimization would have the greatest impact on the end-to-end execution of the application and/or viewing what else is happening on the system during a performance bottleneck

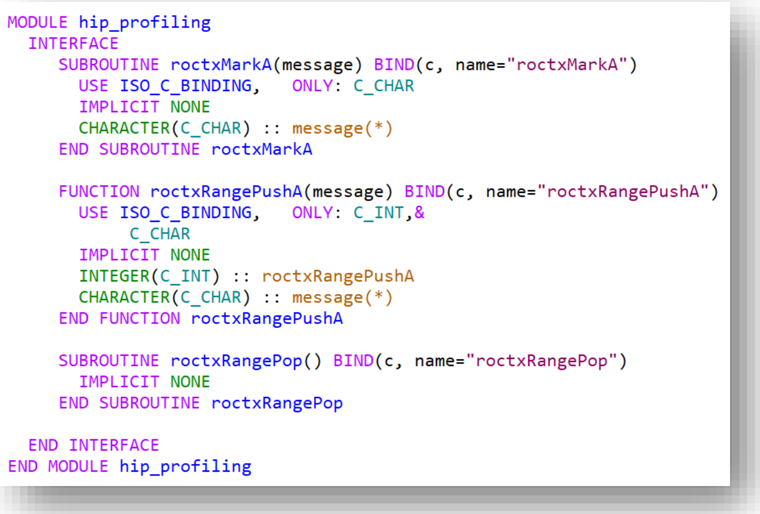
A Deeper Dive into ECTrans with ROC Profiler

- ECTrans with ROC-TX tracing
  - Raw implementation that is currently NOT portable
  - Refactor underway to fix portability and reduce invasiveness of the change
    - Intercept GSTATS calls to include device specific tracing

- ROC profiler

```
rocprof --roctx-trace --hip-trace --stats
        -o output.csv EXE ARGS

EXE : ectrans-benchmark-gpu-sp
ARGS : -n 10 --vordiv --truncation 159 --nlev 137 –norms
--roctx-trace : trace roctx API calls
--hip-trace : trace the HIP API calls
--stats : lists top kernels in output.stats.csv
```

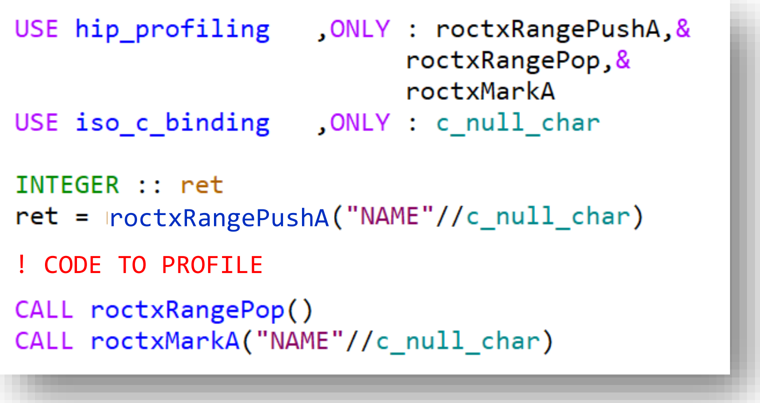


```fortran
MODULE hip_profiling
  INTERFACE
     SUBROUTINE roctxMarkA(message) BIND(c, name="roctxMarkA")
       USE ISO_C_BINDING,   ONLY: C_CHAR
       IMPLICIT NONE
       CHARACTER(C_CHAR) :: message(*)
     END SUBROUTINE roctxMarkA

     FUNCTION roctxRangePushA(message) BIND(c, name="roctxRangePushA")
       USE ISO_C_BINDING,   ONLY: C_INT,&
            C_CHAR
       IMPLICIT NONE
       INTEGER(C_INT) :: roctxRangePushA
       CHARACTER(C_CHAR) :: message(*)
     END FUNCTION roctxRangePushA

     SUBROUTINE roctxRangePop() BIND(c, name="roctxRangePop")
       IMPLICIT NONE
     END SUBROUTINE roctxRangePop

  END INTERFACE
END MODULE hip_profiling
```

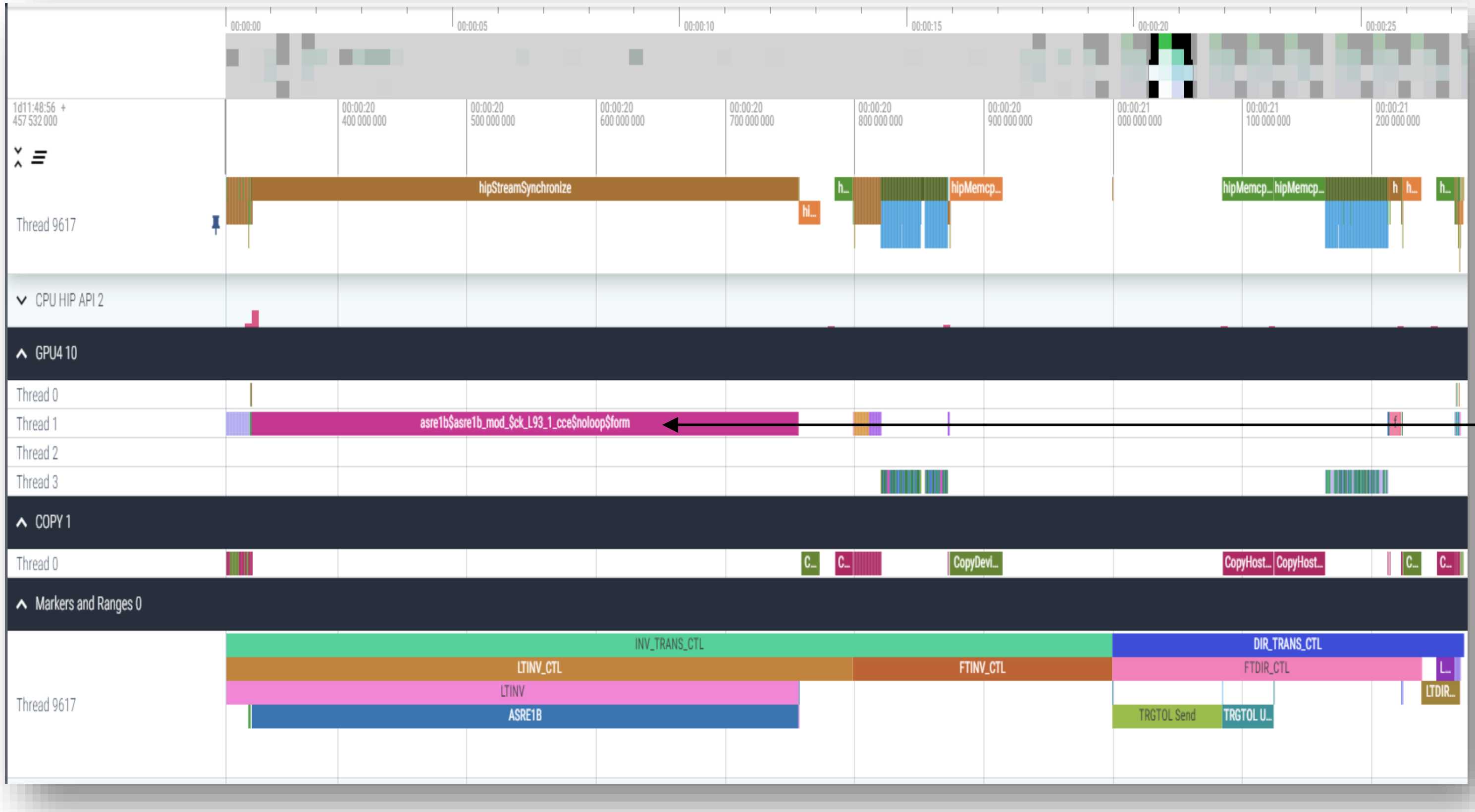


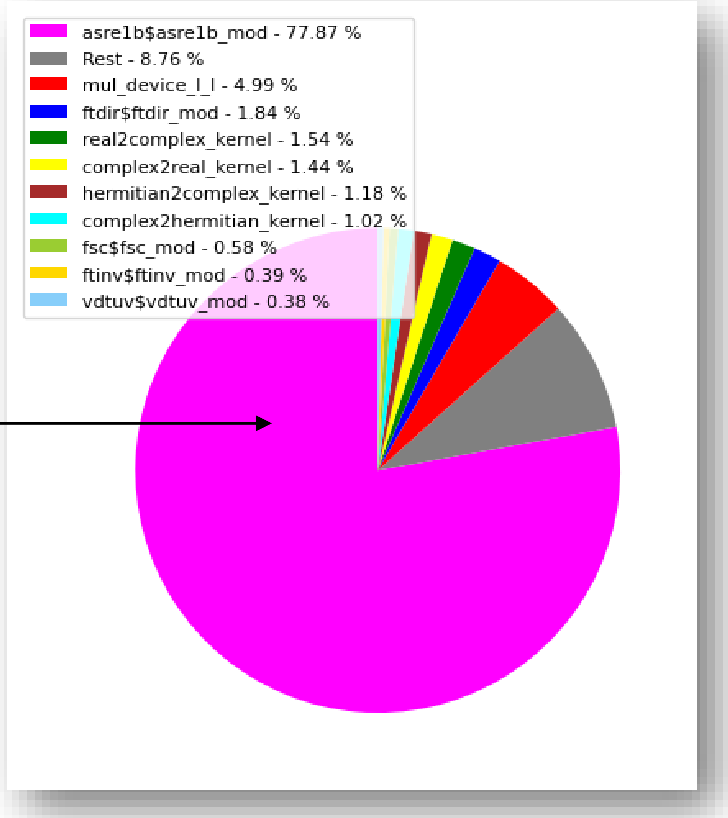
```fortran
USE hip_profiling    ,ONLY : roctxRangePushA,&
                             roctxRangePop,&
                             roctxMarkA
USE iso_c_binding    ,ONLY : c_null_char

INTEGER :: ret
ret = roctxRangePushA("NAME"//c_null_char)
! CODE TO PROFILE
CALL roctxRangePop()
CALL roctxMarkA("NAME"//c_null_char)
```

Zeroth Order Optimizations

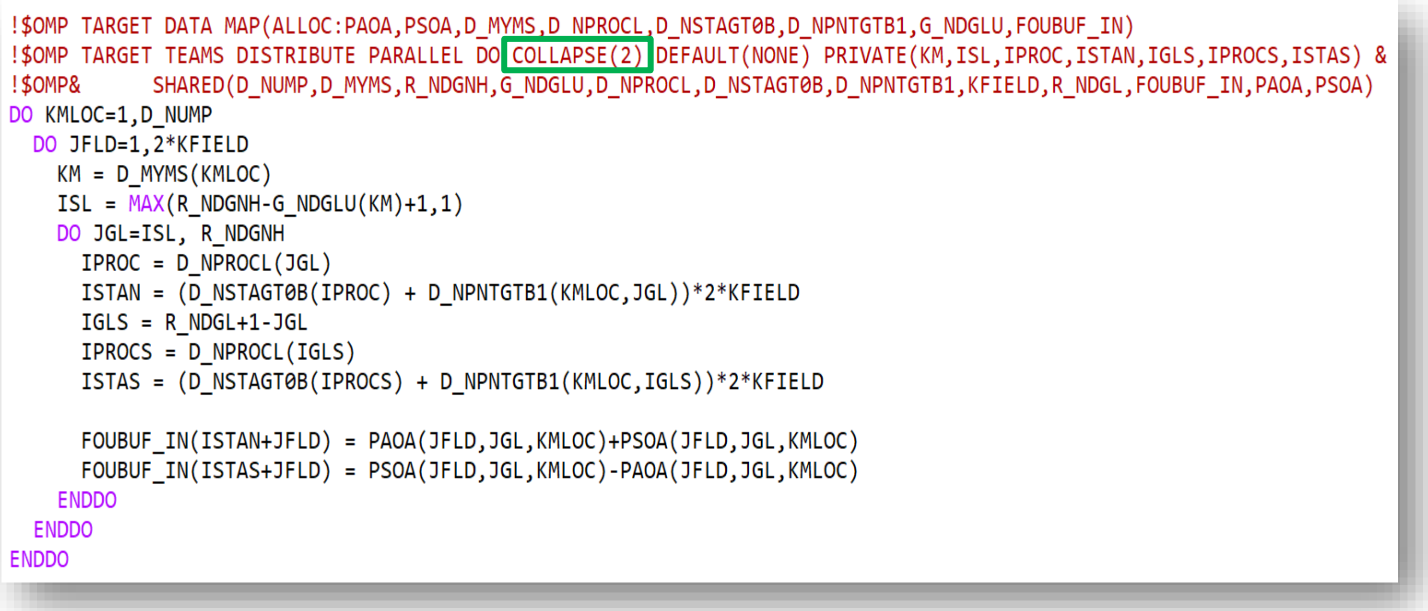
- Can visualize the traced timeline *output.json* in Perfetto
- Prior to commit 9afc482, one GPU kernel was dominant in the OpenMP® implementation





Zeroth Order Optimizations

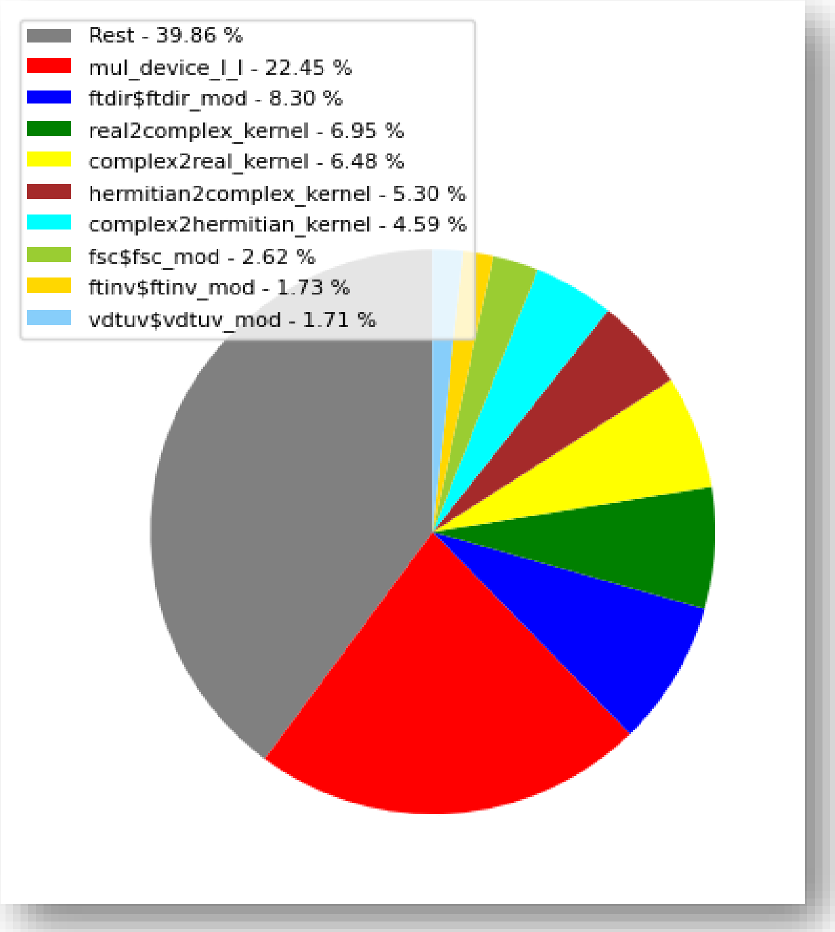
Simple fix to the OMP kernel launch fixes the performance problem!



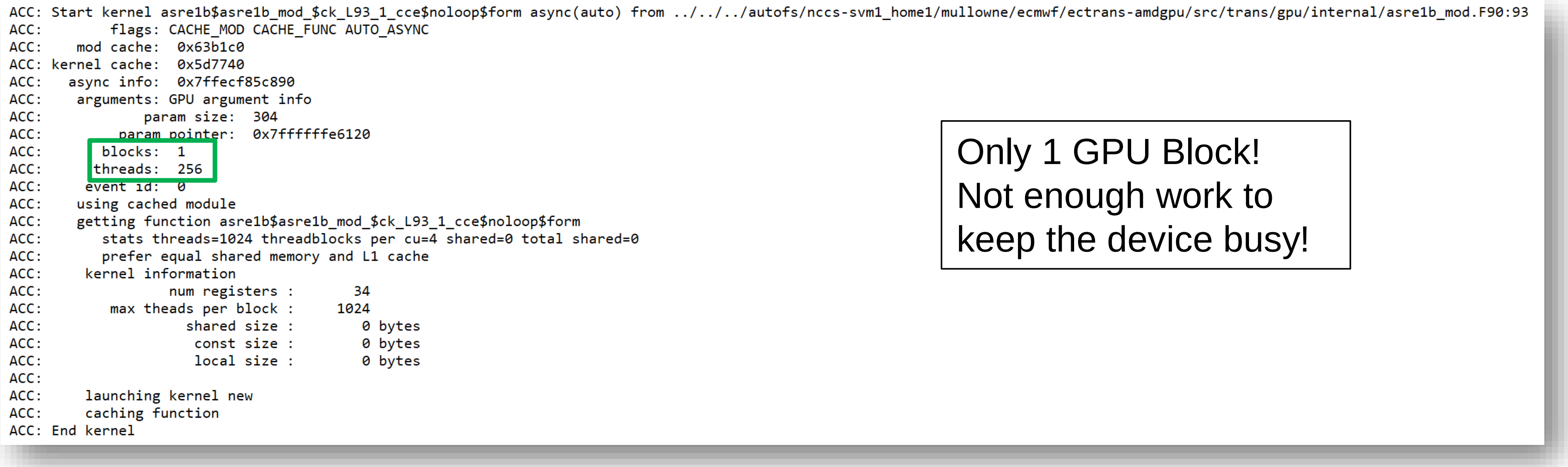
```
!$OMP TARGET DATA MAP(ALLOC:PAOA,PSOA,D_MYMS,D_NPROCL,D_NSTAGT0B,D_NPNTGTB1,G_NDGLU,FOUBUF_IN)
!$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO COLLAPSE(2) DEFAULT(NONE) PRIVATE(KM,ISL,IPROC,ISTAN,IGLS,IPROCS,ISTAS) &
!$OMP&      SHARED(D_NUMP,D_MYMS,R_NDGNH,G_NDGLU,D_NPROCL,D_NSTAGT0B,D_NPNTGTB1,KFIELD,R_NDGL,FOUBUF_IN,PAOA,PSOA)
DO KMLOC=1,D_NUMP
  DO JFLD=1,2*KFIELD
    KM = D_MYMS(KMLOC)
    ISL = MAX(R_NDGNH-G_NDGLU(KM)+1,1)
    DO JGL=ISL, R_NDGNH
      IPROC = D_NPROCL(JGL)
      ISTAN = (D_NSTAGT0B(IPROC) + D_NPNTGTB1(KMLOC,JGL))*2*KFIELD
      IGLS = R_NDGL+1-JGL
      IPROCS = D_NPROCL(IGLS)
      ISTAS = (D_NSTAGT0B(IPROCS) + D_NPNTGTB1(KMLOC,IGLS))*2*KFIELD

      FOUBUF_IN(ISTAN+JFLD) = PAOA(JFLD,JGL,KMLOC)+PSOA(JFLD,JGL,KMLOC)
      FOUBUF_IN(ISTAS+JFLD) = PSOA(JFLD,JGL,KMLOC)-PAOA(JFLD,JGL,KMLOC)
    ENDDO
  ENDDO
ENDDO
```

ASRE1B Opts PR



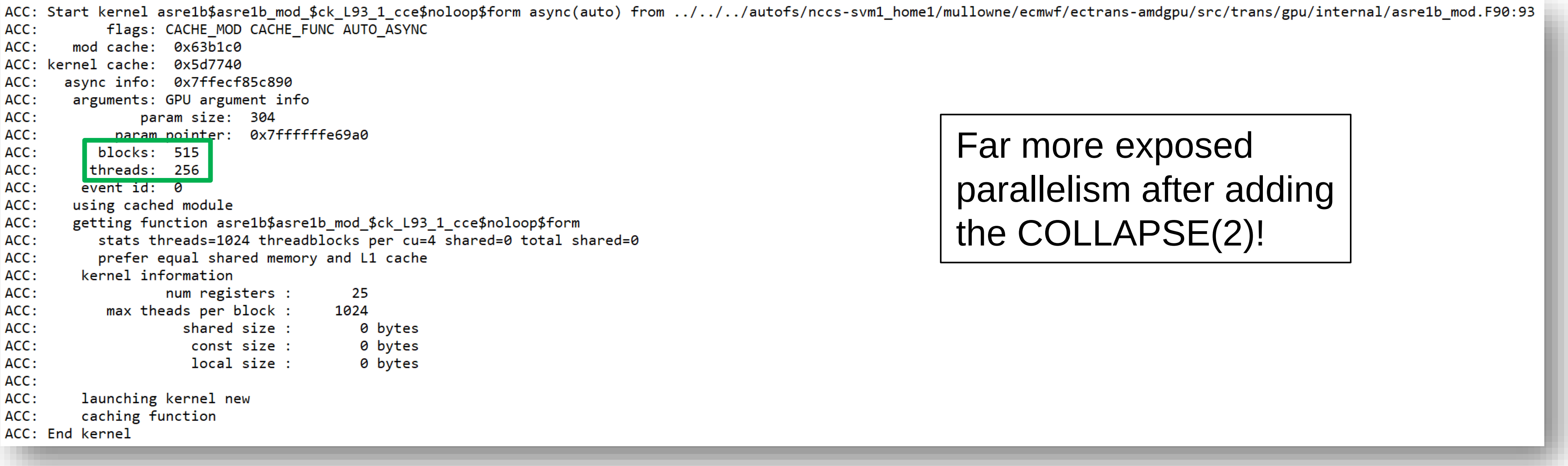
CRAY_ACC_DEBUG=3 : Pre Fix



```
ACC: Start kernel asre1b$asre1b_mod_$ck_L93_1_cce$noloop$form async(auto) from ../../../autofs/nccs-svm1_home1/mullowne/ecmwf/ectrans-amdgpu/src/trans/gpu/internal/asre1b_mod.F90:93
ACC:        flags: CACHE_MOD CACHE_FUNC AUTO_ASYNC
ACC:    mod cache:  0x63b1c0
ACC: kernel cache:  0x5d7740
ACC:   async info:  0x7ffecf85c890
ACC:    arguments: GPU argument info
ACC:            param size:  304
ACC:         param pointer:  0x7ffffffe6120
ACC:       blocks:  1
ACC:      threads:  256
ACC:     event id:  0
ACC:    using cached module
ACC:    getting function asre1b$asre1b_mod_$ck_L93_1_cce$noloop$form
ACC:       stats threads=1024 threadblocks per cu=4 shared=0 total shared=0
ACC:       prefer equal shared memory and L1 cache
ACC:     kernel information
ACC:               num registers :        34
ACC:        max theads per block :      1024
ACC:                 shared size :        0 bytes
ACC:                  const size :        0 bytes
ACC:                  local size :        0 bytes
ACC:
ACC:     launching kernel new
ACC:     caching function
ACC: End kernel
```

Only 1 GPU Block!
Not enough work to
keep the device busy!

CRAY_ACC_DEBUG=3 : Post Fix



```
ACC: Start kernel asre1b$asre1b_mod_$ck_L93_1_cce$noloop$form async(auto) from ../../../autofs/nccs-svm1_home1/mullowne/ecmwf/ectrans-amdgpu/src/trans/gpu/internal/asre1b_mod.F90:93
ACC:        flags: CACHE_MOD CACHE_FUNC AUTO_ASYNC
ACC:    mod cache:  0x63b1c0
ACC: kernel cache:  0x5d7740
ACC:   async info:  0x7ffecf85c890
ACC:    arguments: GPU argument info
ACC:            param size:  304
ACC:         param pointer:  0x7ffffffe69a0
ACC:       blocks:  515
ACC:      threads:  256
ACC:     event id:  0
ACC:    using cached module
ACC:    getting function asre1b$asre1b_mod_$ck_L93_1_cce$noloop$form
ACC:       stats threads=1024 threadblocks per cu=4 shared=0 total shared=0
ACC:       prefer equal shared memory and L1 cache
ACC:     kernel information
ACC:               num registers :        25
ACC:        max theads per block :      1024
ACC:                 shared size :        0 bytes
ACC:                  const size :        0 bytes
ACC:                  local size :        0 bytes
ACC:
ACC:     launching kernel new
ACC:     caching function
ACC: End kernel
```

Far more exposed
parallelism after adding
the COLLAPSE(2)!

Omniperf Analysis

- Client-side Installation
- Profile

```
omniperf profile –n PROFILE_NAME EXE ARGS
PROFILE_NAME : directory for storing the name of the profile.
                              ./workloads/PROFILE_NAME/mi200/
EXE : ectrans-benchmark-gpu-sp
ARGS : -n 10 --vordiv --truncation 159 --nlev 137 –norms
```

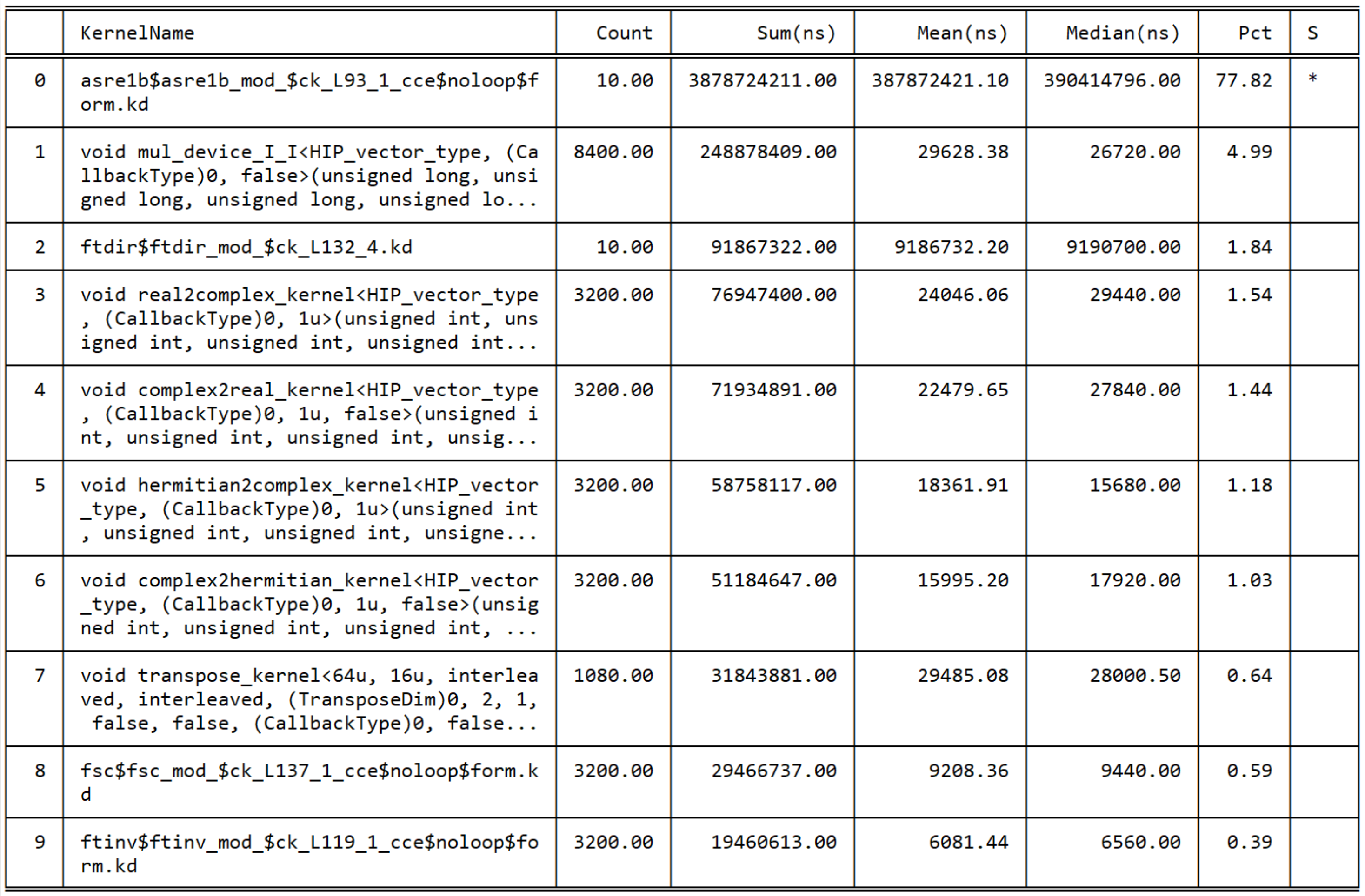
Will take a while to run because detailed profiles are done for ALL performance counters

- Analysis

```
omniperf analyze -p workloads/PROFILE_NAME/mi200/  -b 1 (2, 3, …) –k 0 (1, 2, …)
-b (a.k.a –metric)
  1 : lists system info
  2 : Speed-of-light measurements
-k specifies a kernel id. 0 being the most expensive followed by 1, 2, …
```

Omniperf CLI : Top Kernels (Pre ASRE1B Fix)

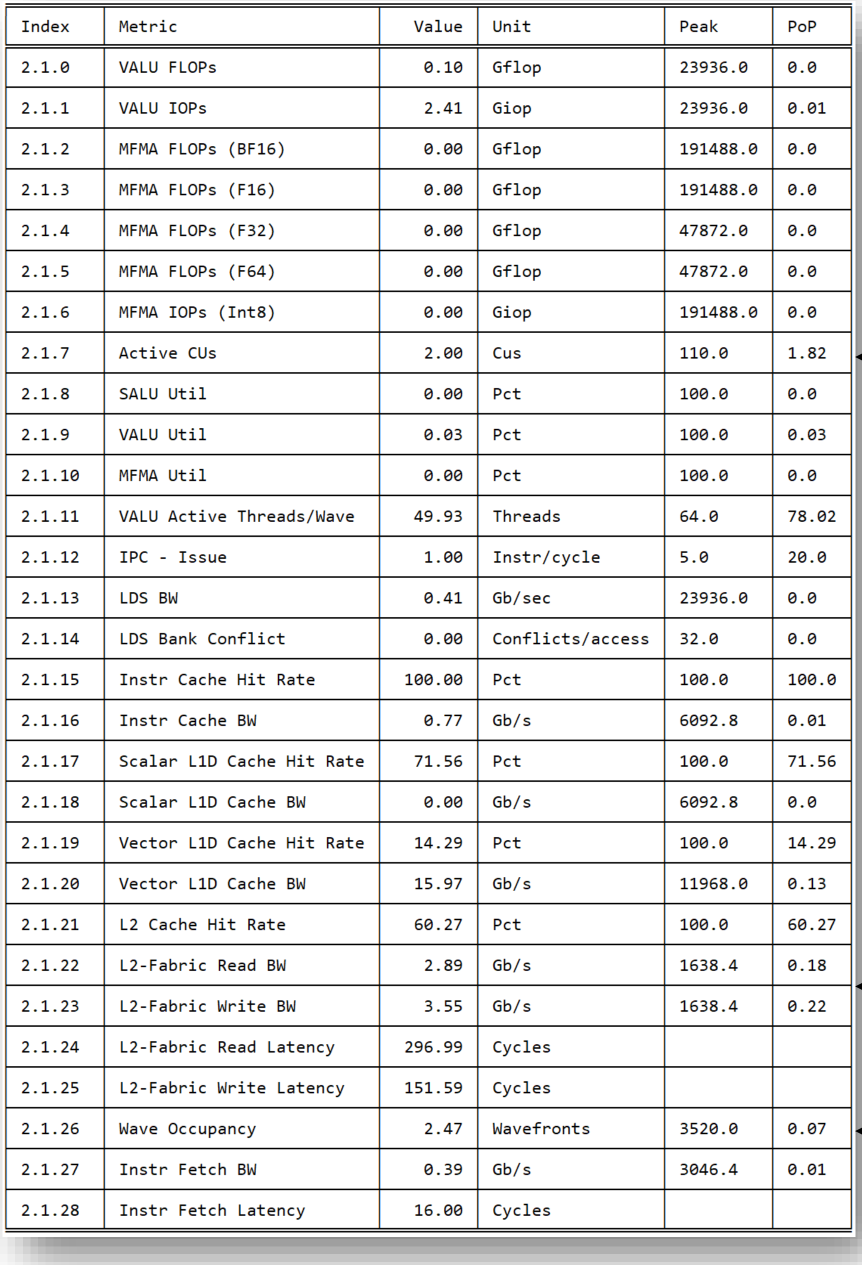
```
omniperf analyze -p workloads/original/mi200/  -b 2 -k 0
```



	KernelName	Count	Sum(ns)	Mean(ns)	Median(ns)	Pct	S
0	asre1b$asre1b_mod_$ck_L93_1_cce$noloop$form.kd	10.00	3878724211.00	387872421.10	390414796.00	77.82	*
1	void mul_device_I_I<HIP_vector_type, (CallbackType)0, false>(unsigned long, unsigned long, unsigned long, unsigned lo...	8400.00	248878409.00	29628.38	26720.00	4.99	
2	ftdir$ftdir_mod_$ck_L132_4.kd	10.00	91867322.00	9186732.20	9190700.00	1.84	
3	void real2complex_kernel<HIP_vector_type, (CallbackType)0, 1u>(unsigned int, unsigned int, unsigned int, unsigned int...	3200.00	76947400.00	24046.06	29440.00	1.54	
4	void complex2real_kernel<HIP_vector_type, (CallbackType)0, 1u, false>(unsigned int, unsigned int, unsigned int, unsig...	3200.00	71934891.00	22479.65	27840.00	1.44	
5	void hermitian2complex_kernel<HIP_vector_type, (CallbackType)0, 1u>(unsigned int, unsigned int, unsigned int, unsigne...	3200.00	58758117.00	18361.91	15680.00	1.18	
6	void complex2hermitian_kernel<HIP_vector_type, (CallbackType)0, 1u, false>(unsigned int, unsigned int, unsigned int, ...	3200.00	51184647.00	15995.20	17920.00	1.03	
7	void transpose_kernel<64u, 16u, interleaved, interleaved, (TransposeDim)0, 2, 1, false, false, (CallbackType)0, false...	1080.00	31843881.00	29485.08	28000.50	0.64	
8	fsc$fsc_mod_$ck_L137_1_cce$noloop$form.kd	3200.00	29466737.00	9208.36	9440.00	0.59	
9	ftinv$ftinv_mod_$ck_L119_1_cce$noloop$form.kd	3200.00	19460613.00	6081.44	6560.00	0.39	

Omniperf CLI : Speed-of-Light (Pre ASRE1B Fix)

```
omniperf analyze -p workloads/original/mi200/  -b 2 -k 0
```



Index	Metric	Value	Unit	Peak	PoP
2.1.0	VALU FLOPs	0.10	Gflop	23936.0	0.0
2.1.1	VALU IOPs	2.41	Giop	23936.0	0.01
2.1.2	MFMA FLOPs (BF16)	0.00	Gflop	191488.0	0.0
2.1.3	MFMA FLOPs (F16)	0.00	Gflop	191488.0	0.0
2.1.4	MFMA FLOPs (F32)	0.00	Gflop	47872.0	0.0
2.1.5	MFMA FLOPs (F64)	0.00	Gflop	47872.0	0.0
2.1.6	MFMA IOPs (Int8)	0.00	Giop	191488.0	0.0
2.1.7	Active CUs	2.00	Cus	110.0	1.82
2.1.8	SALU Util	0.00	Pct	100.0	0.0
2.1.9	VALU Util	0.03	Pct	100.0	0.03
2.1.10	MFMA Util	0.00	Pct	100.0	0.0
2.1.11	VALU Active Threads/Wave	49.93	Threads	64.0	78.02
2.1.12	IPC - Issue	1.00	Instr/cycle	5.0	20.0
2.1.13	LDS BW	0.41	Gb/sec	23936.0	0.0
2.1.14	LDS Bank Conflict	0.00	Conflicts/access	32.0	0.0
2.1.15	Instr Cache Hit Rate	100.00	Pct	100.0	100.0
2.1.16	Instr Cache BW	0.77	Gb/s	6092.8	0.01
2.1.17	Scalar L1D Cache Hit Rate	71.56	Pct	100.0	71.56
2.1.18	Scalar L1D Cache BW	0.00	Gb/s	6092.8	0.0
2.1.19	Vector L1D Cache Hit Rate	14.29	Pct	100.0	14.29
2.1.20	Vector L1D Cache BW	15.97	Gb/s	11968.0	0.13
2.1.21	L2 Cache Hit Rate	60.27	Pct	100.0	60.27
2.1.22	L2-Fabric Read BW	2.89	Gb/s	1638.4	0.18
2.1.23	L2-Fabric Write BW	3.55	Gb/s	1638.4	0.22
2.1.24	L2-Fabric Read Latency	296.99	Cycles		
2.1.25	L2-Fabric Write Latency	151.59	Cycles		
2.1.26	Wave Occupancy	2.47	Wavefronts	3520.0	0.07
2.1.27	Instr Fetch BW	0.39	Gb/s	3046.4	0.01
2.1.28	Instr Fetch Latency	16.00	Cycles		

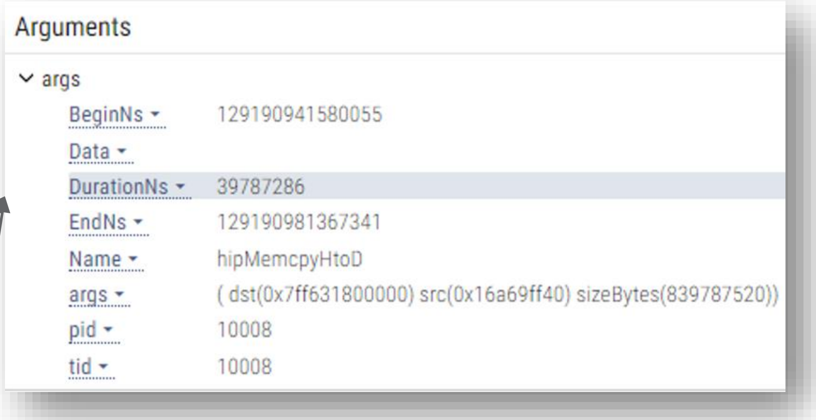
Terrible usage of GPU Compute Units (CUs)!

Terrible L2 Read/Write Bandwidth!

Terrible Wave Occupancy!

First Order Optimizations

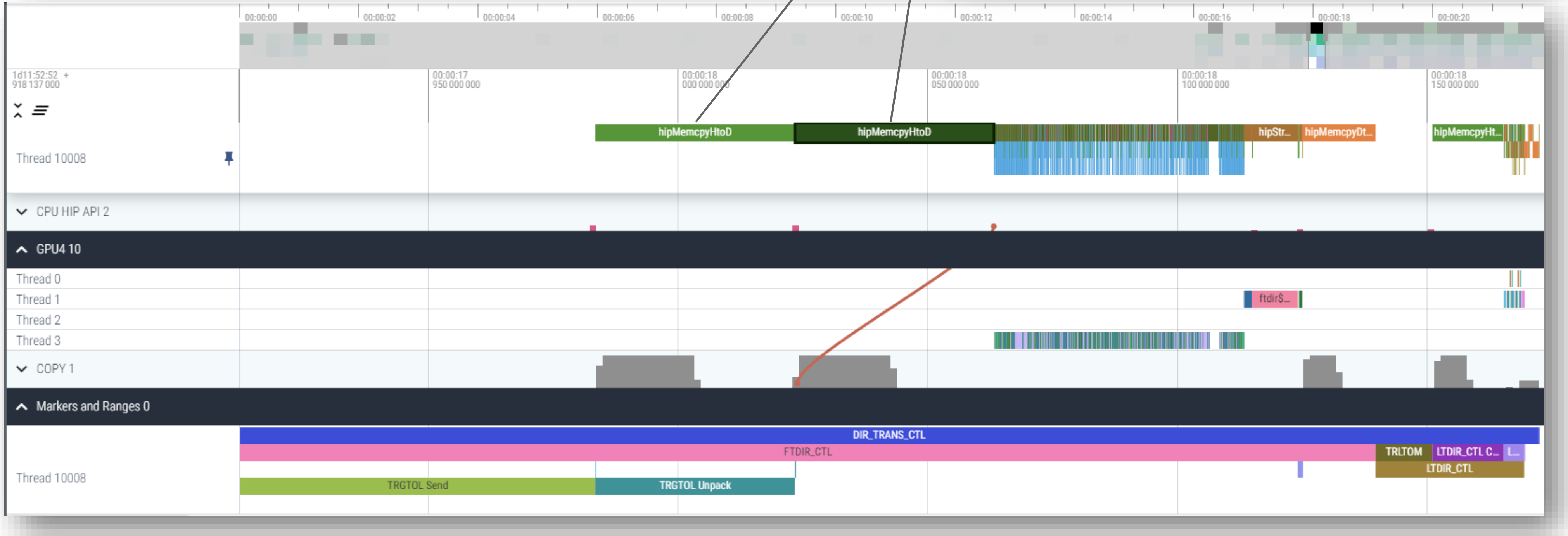
- Same hipMemcpy operation is happening twice!
- TRGTOL_mod.F90 :
  - data transpose computation
  - Only has a GPU aware MPI implementation
- Issue for OpenACC and OpenMP® implementations



Arguments

args

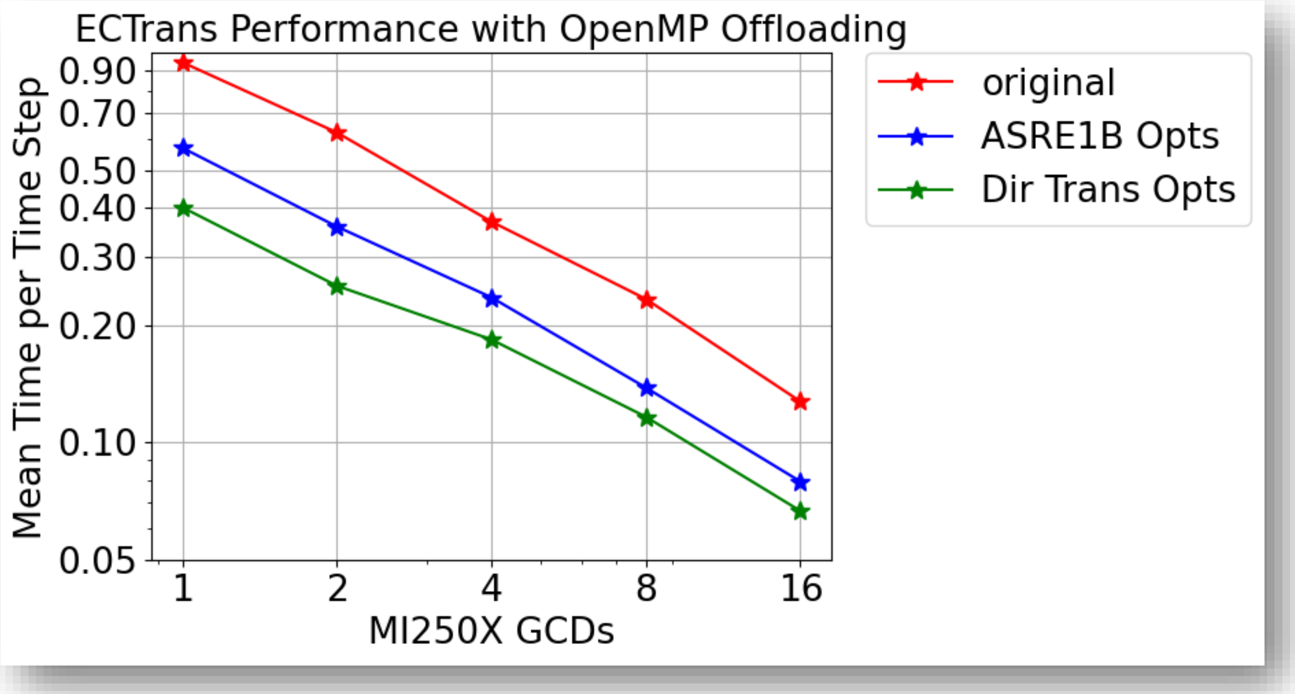
BeginNs	129190941580055
Data	
DurationNs	39787286
EndNs	129190981367341
Name	hipMemcpyHtoD
args	(dst(0x7ff631800000) src(0x16a69ff40) sizeBytes(839787520))
pid	10008
tid	10008

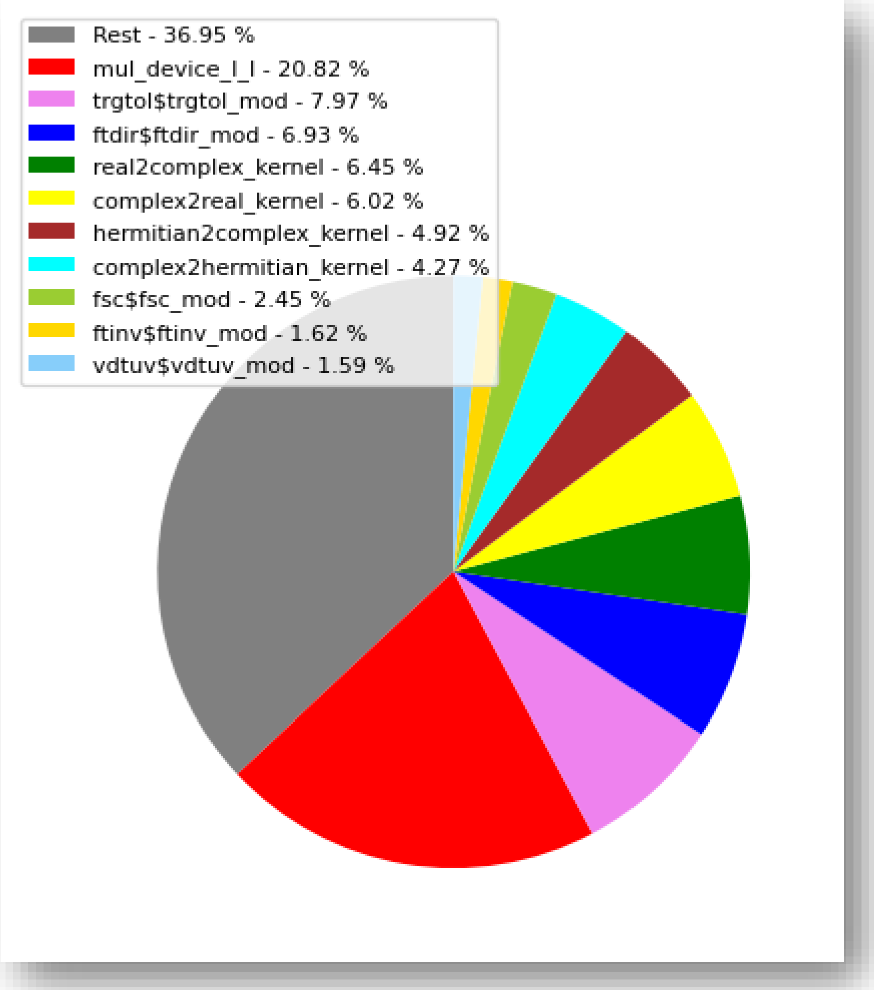


First Order Optimizations

Add a GPU implementation of TRGTOL_MOD.F90

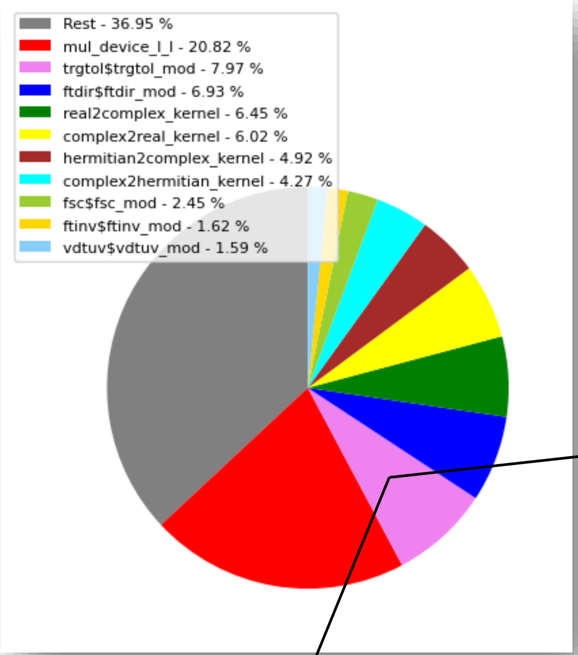
- Implement compute expensive loops on device
- Eliminate hipMemcpys
- OpenACC and OpenMP® achieve nearly identical performance

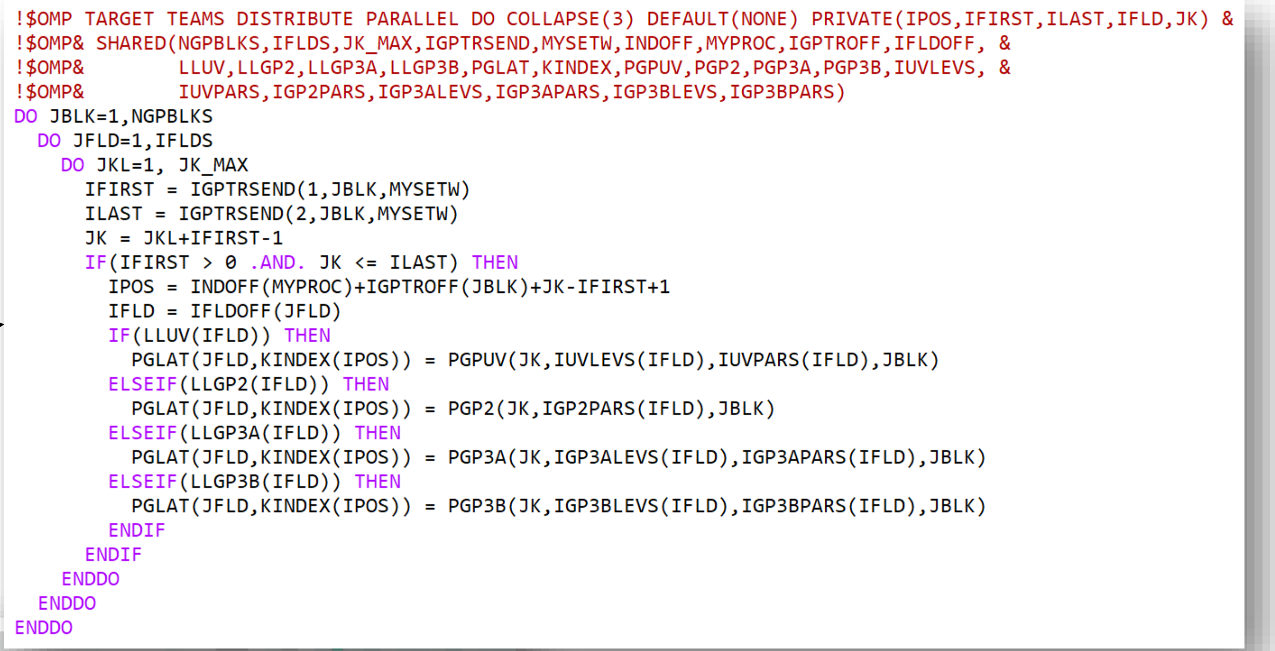




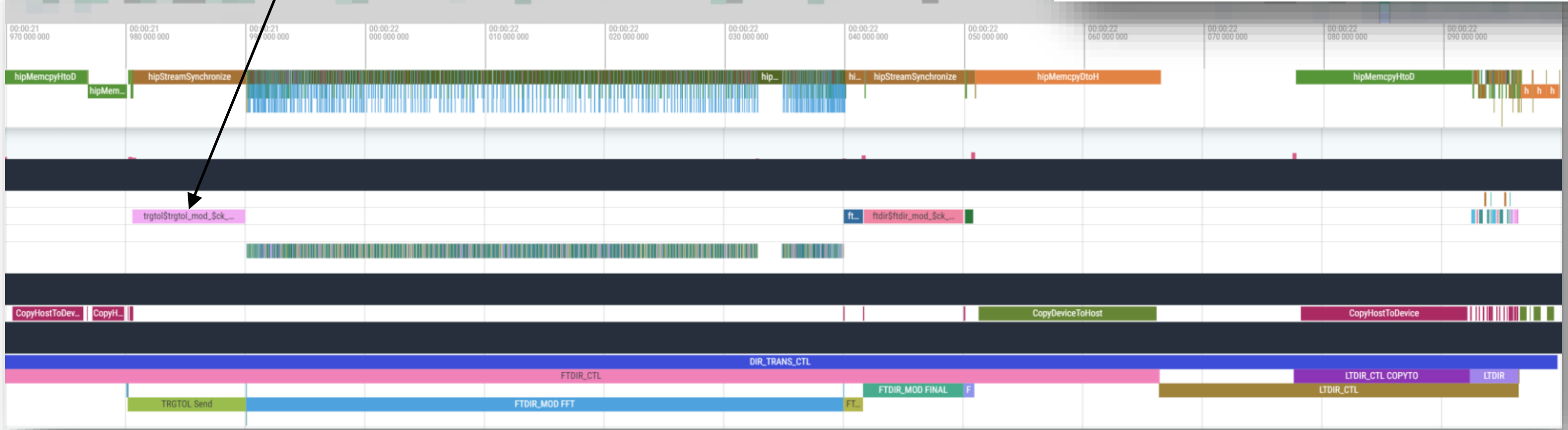
Second Order Optimizations

TRGTOL : “transpose” kernel with
- branching
- integer address calculations



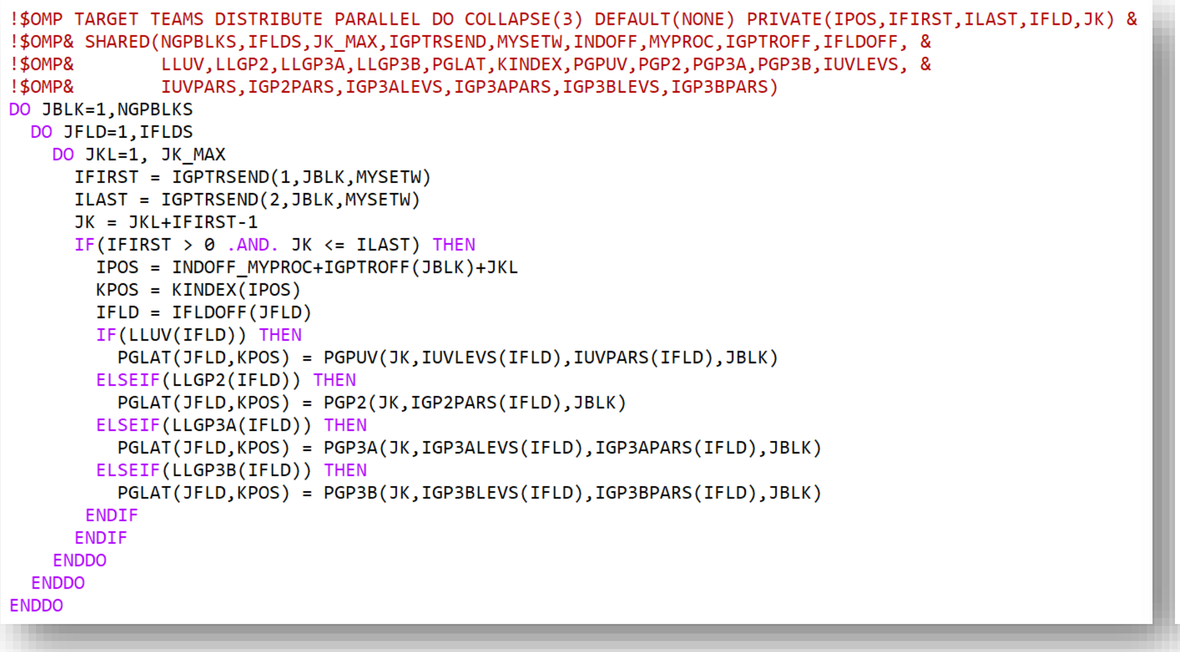


```
!$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO COLLAPSE(3) DEFAULT(NONE) PRIVATE(IPOS,IFIRST,ILAST,IFLD,JK) &
!$OMP& SHARED(NGPBLKS,IFLDS,JK_MAX,IGPTRSEND,MYSETW,INDOFF,MYPROC,IGPTROFF,IFLDOFF, &
!$OMP&        LLUV,LLGP2,LLGP3A,LLGP3B,PGLAT,KINDEX,PGPUV,PGP2,PGP3A,PGP3B,IUVLEVS, &
!$OMP&        IUVPARS,IGP2PARS,IGP3ALEVS,IGP3APARS,IGP3BLEVS,IGP3BPARS)
DO JBLK=1,NGPBLKS
  DO JFLD=1,IFLDS
    DO JKL=1, JK_MAX
      IFIRST = IGPTRSEND(1,JBLK,MYSETW)
      ILAST = IGPTRSEND(2,JBLK,MYSETW)
      JK = JKL+IFIRST-1
      IF(IFIRST > 0 .AND. JK <= ILAST) THEN
        IPOS = INDOFF(MYPROC)+IGPTROFF(JBLK)+JK-IFIRST+1
        IFLD = IFLDOFF(JFLD)
        IF(LLUV(IFLD)) THEN
          PGLAT(JFLD,KINDEX(IPOS)) = PGPUV(JK,IUVLEVS(IFLD),IUVPARS(IFLD),JBLK)
        ELSEIF(LLGP2(IFLD)) THEN
          PGLAT(JFLD,KINDEX(IPOS)) = PGP2(JK,IGP2PARS(IFLD),JBLK)
        ELSEIF(LLGP3A(IFLD)) THEN
          PGLAT(JFLD,KINDEX(IPOS)) = PGP3A(JK,IGP3ALEVS(IFLD),IGP3APARS(IFLD),JBLK)
        ELSEIF(LLGP3B(IFLD)) THEN
          PGLAT(JFLD,KINDEX(IPOS)) = PGP3B(JK,IGP3BLEVS(IFLD),IGP3BPARS(IFLD),JBLK)
        ENDIF
      ENDIF
    ENDDO
  ENDDO
ENDDO
```

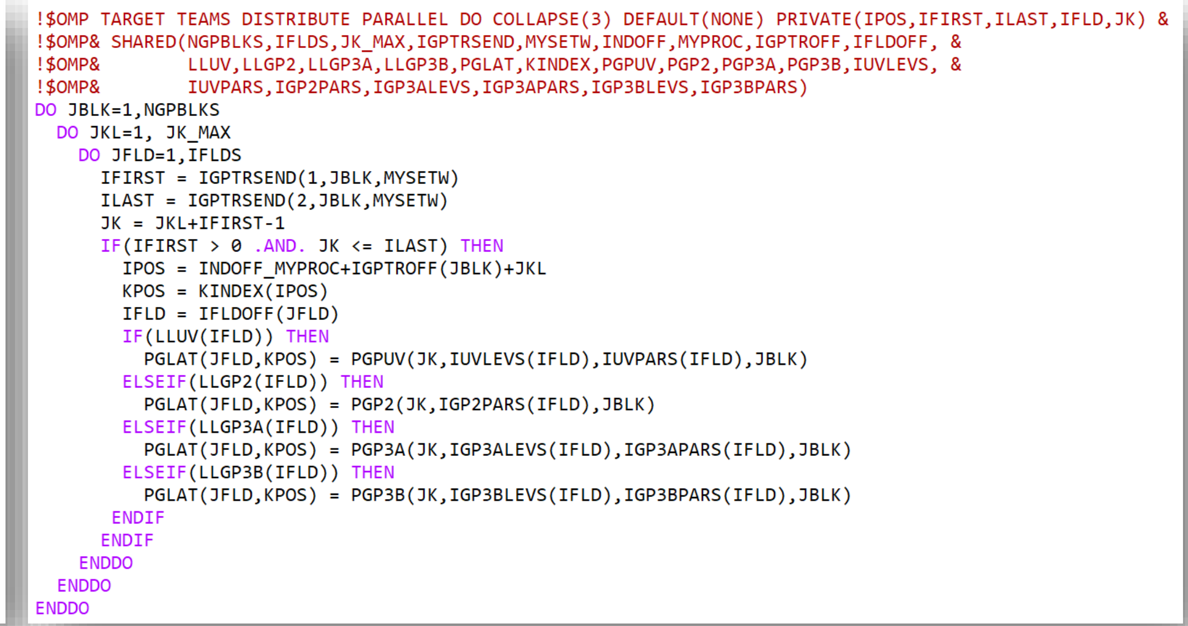


Second Order Optimizations

- Swap the DO Loop Ordering of the 2 innermost loops
  - This will impact the L2 Cache Read/Write performance from HBM
  - Could this have other impacts?



```
!$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO COLLAPSE(3) DEFAULT(NONE) PRIVATE(IPOS,IFIRST,ILAST,IFLD,JK) &
!$OMP& SHARED(NGPBLKS,IFLDS,JK_MAX,IGPTRSEND,MYSETW,INDOFF,MYPROC,IGPTROFF,IFLDOFF, &
!$OMP&        LLUV,LLGP2,LLGP3A,LLGP3B,PGLAT,KINDEX,PGPUV,PGP2,PGP3A,PGP3B,IUVLEVS, &
!$OMP&        IUVPARS,IGP2PARS,IGP3ALEVS,IGP3APARS,IGP3BLEVS,IGP3BPARS)
DO JBLK=1,NGPBLKS
  DO JFLD=1,IFLDS
    DO JKL=1, JK_MAX
      IFIRST = IGPTRSEND(1,JBLK,MYSETW)
      ILAST = IGPTRSEND(2,JBLK,MYSETW)
      JK = JKL+IFIRST-1
      IF(IFIRST > 0 .AND. JK <= ILAST) THEN
        IPOS = INDOFF_MYPROC+IGPTROFF(JBLK)+JKL
        KPOS = KINDEX(IPOS)
        IFLD = IFLDOFF(JFLD)
        IF(LLUV(IFLD)) THEN
          PGLAT(JFLD,KPOS) = PGPUV(JK,IUVLEVS(IFLD),IUVPARS(IFLD),JBLK)
        ELSEIF(LLGP2(IFLD)) THEN
          PGLAT(JFLD,KPOS) = PGP2(JK,IGP2PARS(IFLD),JBLK)
        ELSEIF(LLGP3A(IFLD)) THEN
          PGLAT(JFLD,KPOS) = PGP3A(JK,IGP3ALEVS(IFLD),IGP3APARS(IFLD),JBLK)
        ELSEIF(LLGP3B(IFLD)) THEN
          PGLAT(JFLD,KPOS) = PGP3B(JK,IGP3BLEVS(IFLD),IGP3BPARS(IFLD),JBLK)
       ENDIF
      ENDIF
    ENDDO
  ENDDO
ENDDO
```

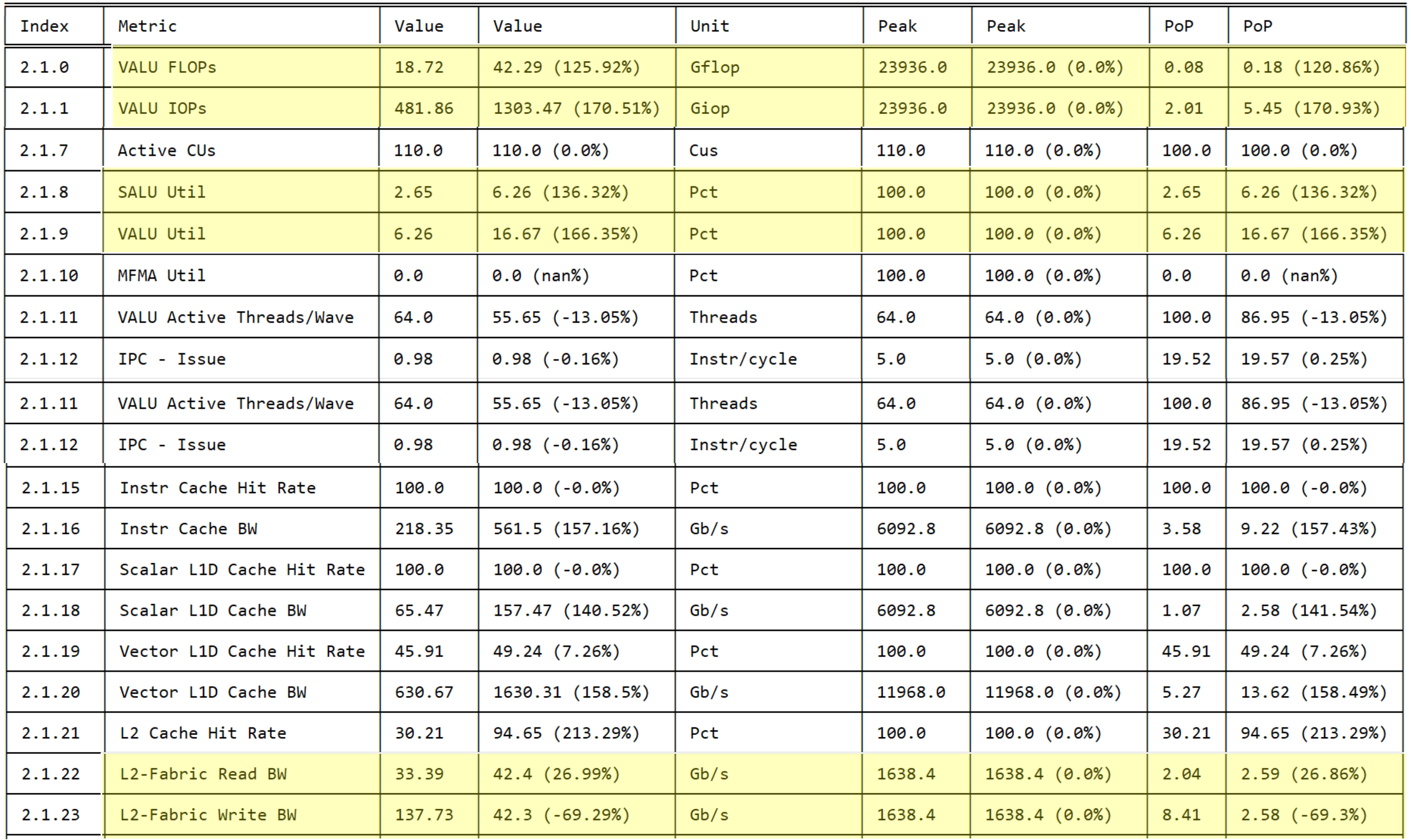


```
!$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO COLLAPSE(3) DEFAULT(NONE) PRIVATE(IPOS,IFIRST,ILAST,IFLD,JK) &
!$OMP& SHARED(NGPBLKS,IFLDS,JK_MAX,IGPTRSEND,MYSETW,INDOFF,MYPROC,IGPTROFF,IFLDOFF, &
!$OMP&        LLUV,LLGP2,LLGP3A,LLGP3B,PGLAT,KINDEX,PGPUV,PGP2,PGP3A,PGP3B,IUVLEVS, &
!$OMP&        IUVPARS,IGP2PARS,IGP3ALEVS,IGP3APARS,IGP3BLEVS,IGP3BPARS)
DO JBLK=1,NGPBLKS
  DO JKL=1, JK_MAX
    DO JFLD=1,IFLDS
      IFIRST = IGPTRSEND(1,JBLK,MYSETW)
      ILAST = IGPTRSEND(2,JBLK,MYSETW)
      JK = JKL+IFIRST-1
      IF(IFIRST > 0 .AND. JK <= ILAST) THEN
        IPOS = INDOFF_MYPROC+IGPTROFF(JBLK)+JKL
        KPOS = KINDEX(IPOS)
        IFLD = IFLDOFF(JFLD)
        IF(LLUV(IFLD)) THEN
          PGLAT(JFLD,KPOS) = PGPUV(JK,IUVLEVS(IFLD),IUVPARS(IFLD),JBLK)
        ELSEIF(LLGP2(IFLD)) THEN
          PGLAT(JFLD,KPOS) = PGP2(JK,IGP2PARS(IFLD),JBLK)
        ELSEIF(LLGP3A(IFLD)) THEN
          PGLAT(JFLD,KPOS) = PGP3A(JK,IGP3ALEVS(IFLD),IGP3APARS(IFLD),JBLK)
        ELSEIF(LLGP3B(IFLD)) THEN
          PGLAT(JFLD,KPOS) = PGP3B(JK,IGP3BLEVS(IFLD),IGP3BPARS(IFLD),JBLK)
       ENDIF
      ENDIF
    ENDDO
  ENDDO
ENDDO
```

	Original	DO Loop SWAP
Mean Kernel Execution Time	9.33 ms	4.24 ms

Omniperf CLI : Multiple run comparison

```
omniperf analyze -p workloads/run1/mi200/ -k 1 -p workloads/run2/mi200/ -k 6 -b 2
```



Index	Metric	Value	Value	Unit	Peak	Peak	PoP	PoP
2.1.0	VALU FLOPs	18.72	42.29 (125.92%)	Gflop	23936.0	23936.0 (0.0%)	0.08	0.18 (120.86%)
2.1.1	VALU IOPs	481.86	1303.47 (170.51%)	Giop	23936.0	23936.0 (0.0%)	2.01	5.45 (170.93%)
2.1.7	Active CUs	110.0	110.0 (0.0%)	Cus	110.0	110.0 (0.0%)	100.0	100.0 (0.0%)
2.1.8	SALU Util	2.65	6.26 (136.32%)	Pct	100.0	100.0 (0.0%)	2.65	6.26 (136.32%)
2.1.9	VALU Util	6.26	16.67 (166.35%)	Pct	100.0	100.0 (0.0%)	6.26	16.67 (166.35%)
2.1.10	MFMA Util	0.0	0.0 (nan%)	Pct	100.0	100.0 (0.0%)	0.0	0.0 (nan%)
2.1.11	VALU Active Threads/Wave	64.0	55.65 (-13.05%)	Threads	64.0	64.0 (0.0%)	100.0	86.95 (-13.05%)
2.1.12	IPC - Issue	0.98	0.98 (-0.16%)	Instr/cycle	5.0	5.0 (0.0%)	19.52	19.57 (0.25%)
2.1.11	VALU Active Threads/Wave	64.0	55.65 (-13.05%)	Threads	64.0	64.0 (0.0%)	100.0	86.95 (-13.05%)
2.1.12	IPC - Issue	0.98	0.98 (-0.16%)	Instr/cycle	5.0	5.0 (0.0%)	19.52	19.57 (0.25%)
2.1.15	Instr Cache Hit Rate	100.0	100.0 (-0.0%)	Pct	100.0	100.0 (0.0%)	100.0	100.0 (-0.0%)
2.1.16	Instr Cache BW	218.35	561.5 (157.16%)	Gb/s	6092.8	6092.8 (0.0%)	3.58	9.22 (157.43%)
2.1.17	Scalar L1D Cache Hit Rate	100.0	100.0 (-0.0%)	Pct	100.0	100.0 (0.0%)	100.0	100.0 (-0.0%)
2.1.18	Scalar L1D Cache BW	65.47	157.47 (140.52%)	Gb/s	6092.8	6092.8 (0.0%)	1.07	2.58 (141.54%)
2.1.19	Vector L1D Cache Hit Rate	45.91	49.24 (7.26%)	Pct	100.0	100.0 (0.0%)	45.91	49.24 (7.26%)
2.1.20	Vector L1D Cache BW	630.67	1630.31 (158.5%)	Gb/s	11968.0	11968.0 (0.0%)	5.27	13.62 (158.49%)
2.1.21	L2 Cache Hit Rate	30.21	94.65 (213.29%)	Pct	100.0	100.0 (0.0%)	30.21	94.65 (213.29%)
2.1.22	L2-Fabric Read BW	33.39	42.4 (26.99%)	Gb/s	1638.4	1638.4 (0.0%)	2.04	2.59 (26.86%)
2.1.23	L2-Fabric Write BW	137.73	42.3 (-69.29%)	Gb/s	1638.4	1638.4 (0.0%)	8.41	2.58 (-69.3%)

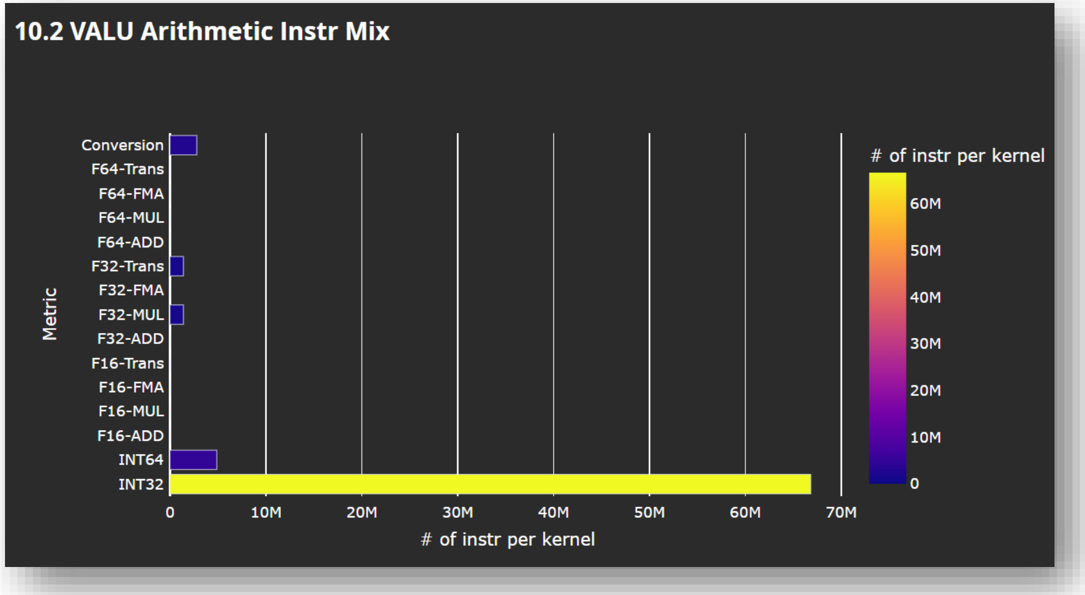
Big improvement in VALU IOPs and utilization!

Omniperf GUI

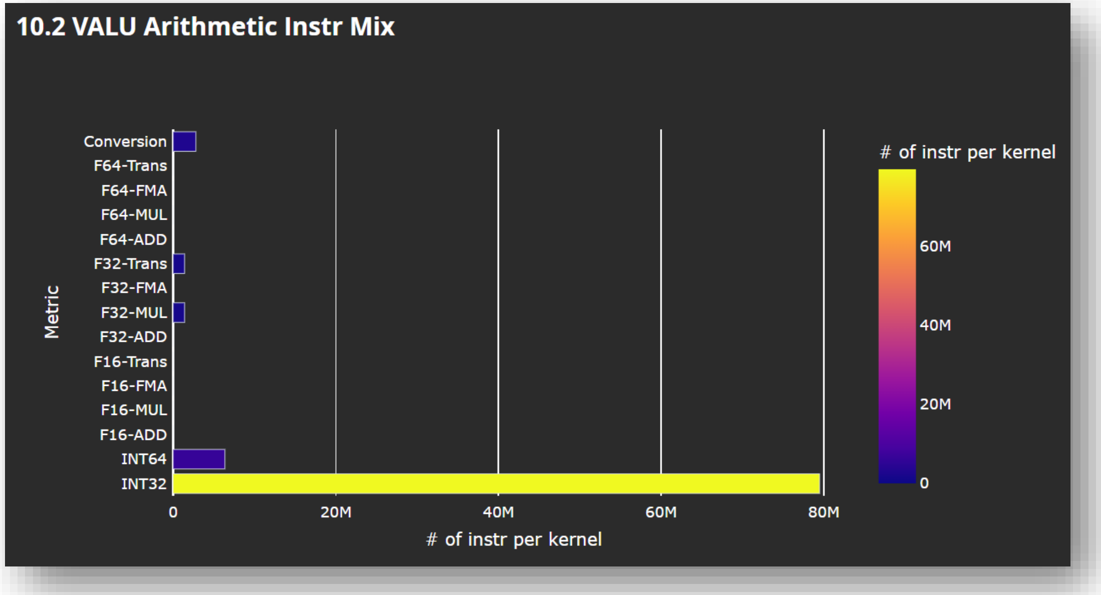
```
ssh -L 8050:login2:8050 mullowne@login2.crusher.olcf.gov
omniperf analyze –p workloads/MY_PROFILE/mi200/  --gui
```

view in browser at https://127.0.0.1:8050/

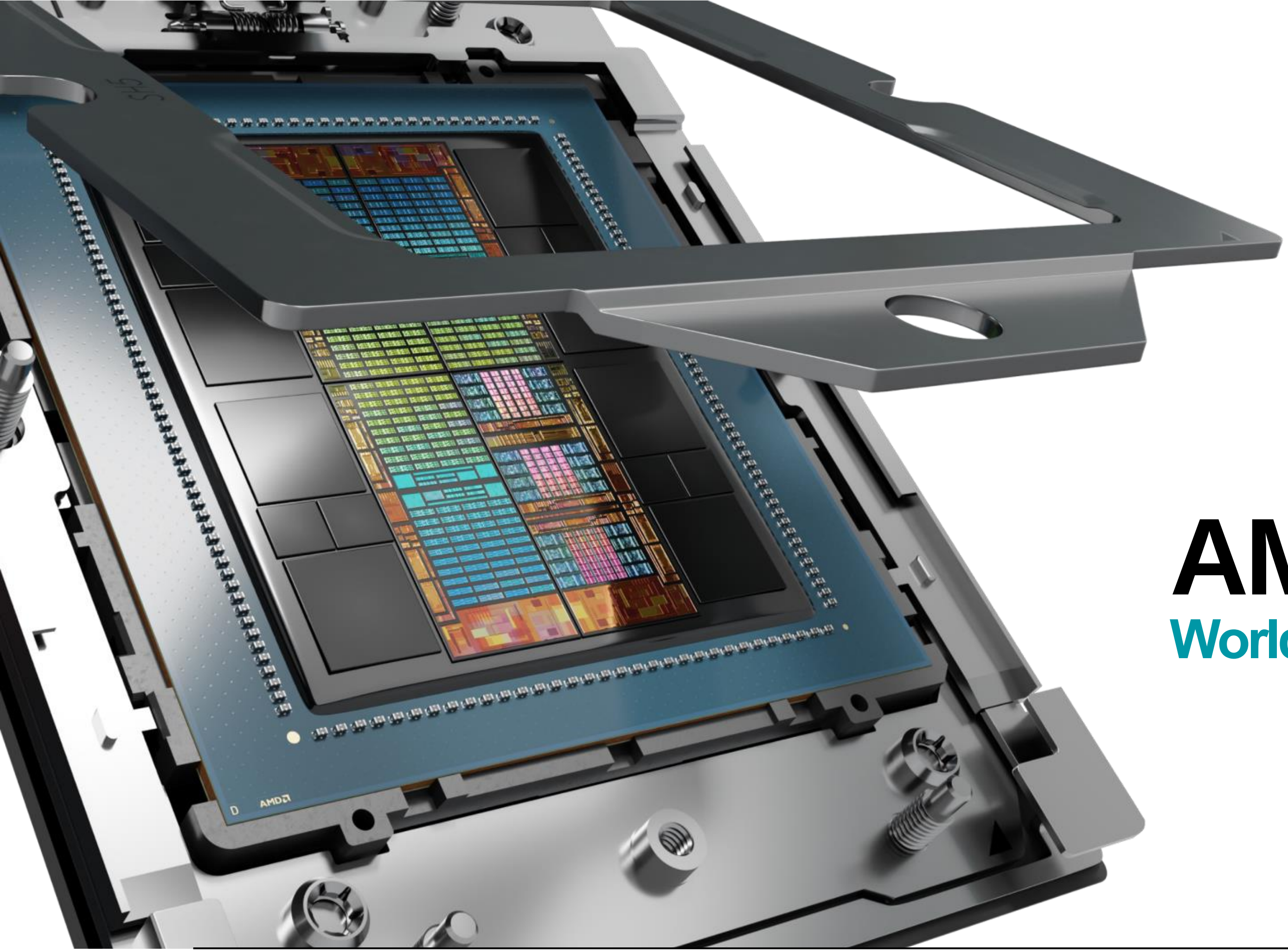
Original



DO Loop Swap



Although INT32 ops increased by 15%, the INT32 processing rate increased by 170%!



AMD Instinct™ MI300A

World's first APU accelerator for AI and HPC

AMD CDNA 3 Next-Gen Accelerator Architecture

ZEN 4 24 CPU Cores

128 GB HBM3

5nm and 6nm Process Technology

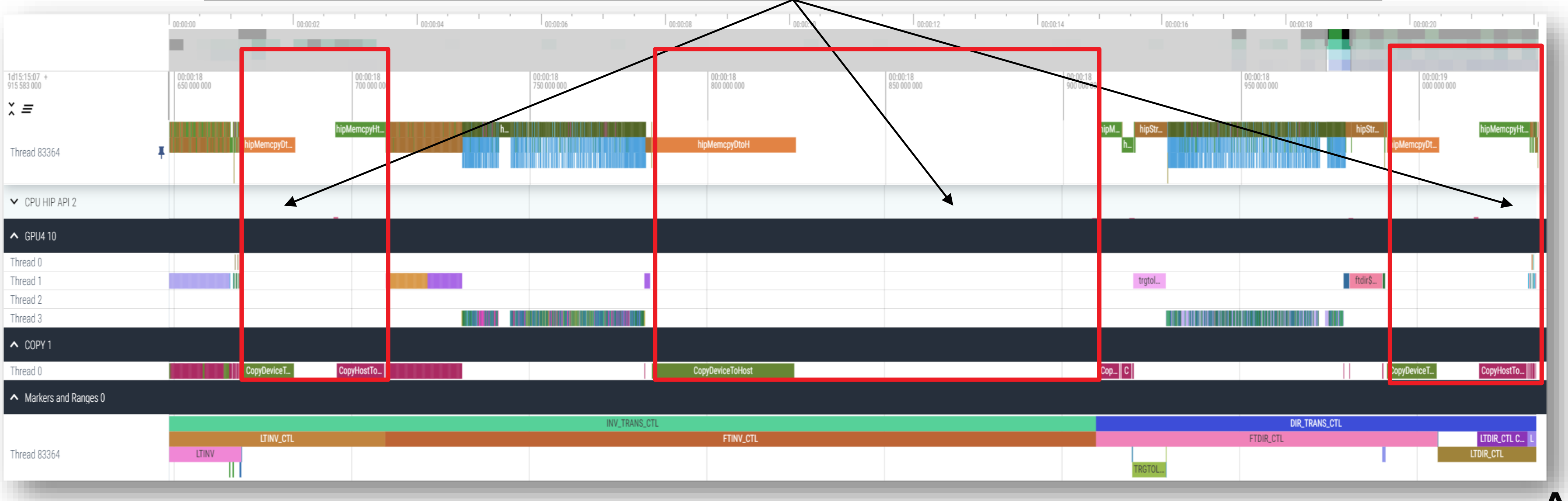
Shared Memory CPU + GPU

MI300A Impact on ECTrans

- GPU/CPU computations can use the same, system-allocated memory
- OpenMP® MAP, ALLOC, … will effectively become No-Ops
  - Double memory and data movement hits associated with Apps that use HIP API will be eliminated
  - Backward-compatibility and performance advantages are built into the Compiler design

Timelines on MI250X show large regions where the device is unused

- In each case, large data transfers before and after.
- These transfers disappear with OpenMP® Offload on MI300A!



Summary

- AMD-ECMWF collaboration is a long-term engagement
- AMD profiling tools can be used effectively to accelerate key regions of offloaded code
  - ... even for people without significant knowledge of a codebase
- MI300A offers a high potential for additional GPU acceleration, while continuing
  - code portability
  - backward compatibility

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED ‘AS IS.” AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Third-party content is licensed to you directly by the third party that owns the content and is not licensed to you by AMD. ALL LINKED THIRD-PARTY CONTENT IS PROVIDED “AS IS” WITHOUT A WARRANTY OF ANY KIND. USE OF SUCH THIRD-PARTY CONTENT IS DONE AT YOUR SOLE DISCRETION AND UNDER NO CIRCUMSTANCES WILL AMD BE LIABLE TO YOU FOR ANY THIRD-PARTY CONTENT. YOU ASSUME ALL RISK AND ARE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY ARISE FROM YOUR USE OF THIRD-PARTY CONTENT.

AMD