



Introduction to profiling

Presenter: Giacomo Capodaglio
AMD @ CASTIEL
Oct 28-30, 2025

AMD 
together we advance_

Profiling your application: what does it mean?

Profiling your application means looking for ways to improve its performance. The main focus is almost always to **reduce the overall runtime**.

Assumptions for this presentation

- Your application runs on GPU
- You know what is the main purpose of each GPU kernel in your application
- You are aware that moving data between CPU and GPU has a toll on the overall runtime
- *(Optional)* You have observed that your application performs better on non-AMD hardware, despite comparable specifications

What you'll learn in this presentation

Learning outcomes for this presentation

- **Profiling basics**
 - Learn how to measure where your application spends time on the GPU, including kernel execution and data transfers
- **Bottleneck identification**
 - Discover how to pinpoint basic performance limitations
- **Performance assessment and optimization**
 - Learn where to look for opportunities to improve performance and how to evaluate them

Before using profilers

- `rocm-smi` (later `amd-smi`) – CLI tool for monitoring overall GPU activity
- `export AMD_LOG_LEVEL=3` – HIP logging, very verbose
- `export LIBOMPTARGET_INFO=-1` – OpenMP offload activity (memory copies, kernel offload etc.)
- `export LIBOMPTARGET_KERNEL_TRACE=1` – kernel names, number of teams/threads, register usage

We'll focus on a Jacobi solver example

- Clone our HPCTrainingExamples repo and navigate to the HIP/Jacobi directory:

```
git clone https://github.com/amd/HPCTrainingExamples.git  
cd HPCTrainingExamples/HIP/jacobi
```

- This is a code that implements a Jacobi solver to solve Poisson's equation (a partial differential equation) discretized with a finite difference scheme
- HIP is used to offload the most relevant computations to the GPU
- MPI is used to handle parallel runs, though for **simplicity here we will just consider serial runs**

For more details, see the dedicated [blog post](#) on the Jacobi example

AMD tools overview

rocprofv3
(with ROCm 6.2+ stack)

rocprof-sys
(formerly omnitrace)

rocprof-compute
(formerly omniperf)

Hardware Counters

Raw collection of GPU counters and traces

Counter collection with user input files

Counter results printed to a CSV

Trace collection

Comprehensive trace collection

CPU

GPU

Performance Analysis

Automated collection of hardware counters

Analysis

Visualization

Traces and timelines

Trace collection support for

CPU copy

HIP API

HSA API

GPU Kernels

Supports

CPU copy

HIP API

HSA API

GPU Kernels

OpenMP®

MPI

Kokkos

p-threads

multi-GPU

Supports

Speed of Light

Memory chart

Rooflines

Kernel comparison

Visualization

Traces visualized with Perfetto

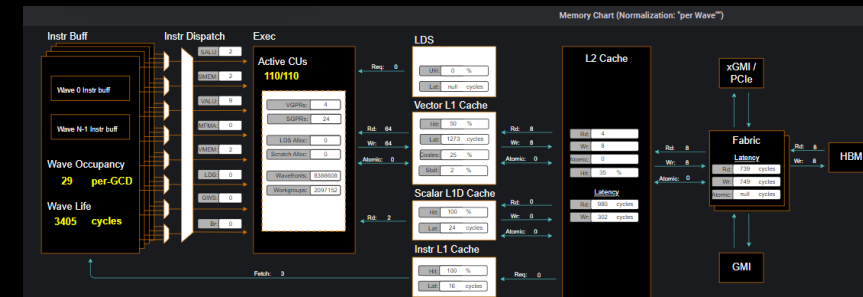
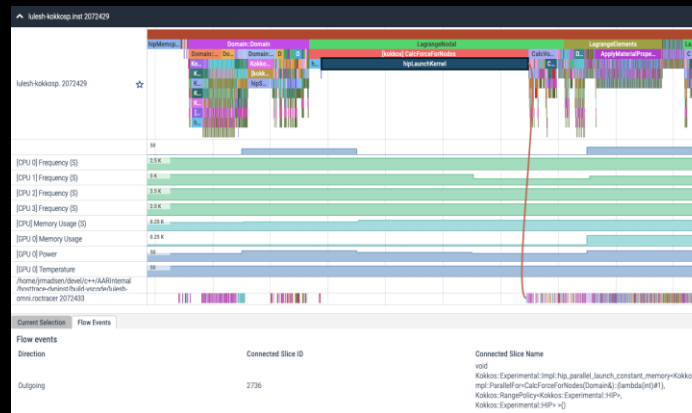
Visualization

Traces visualized with Perfetto

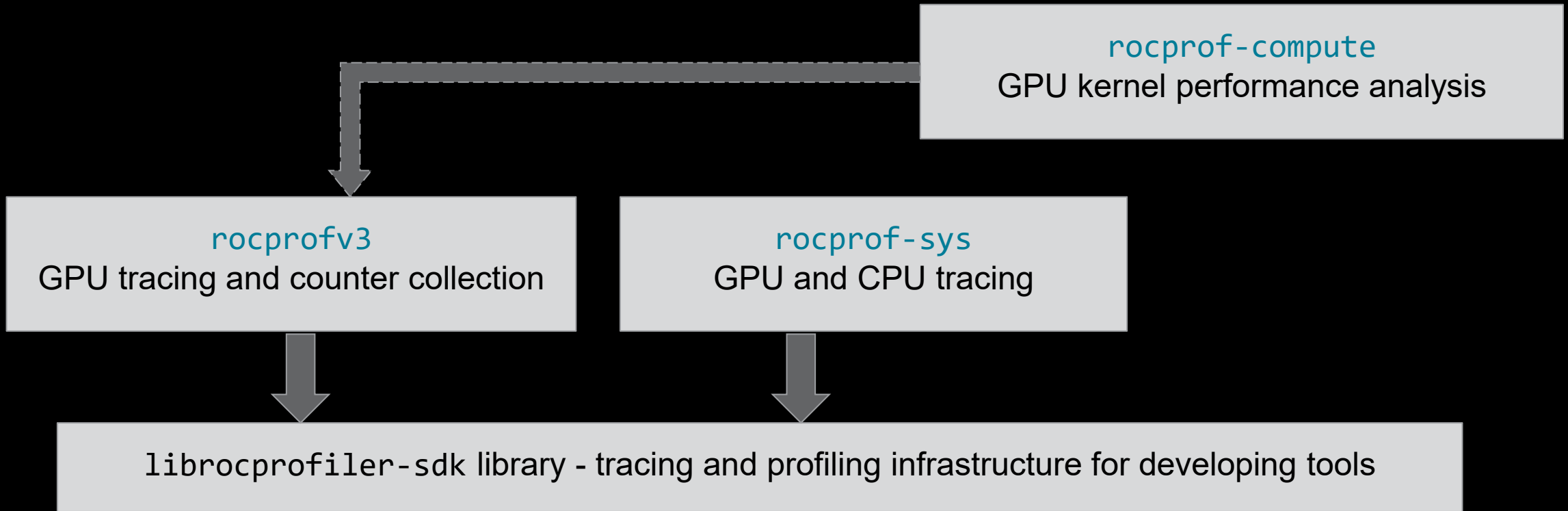
Visualization

With Grafana or standalone GUI

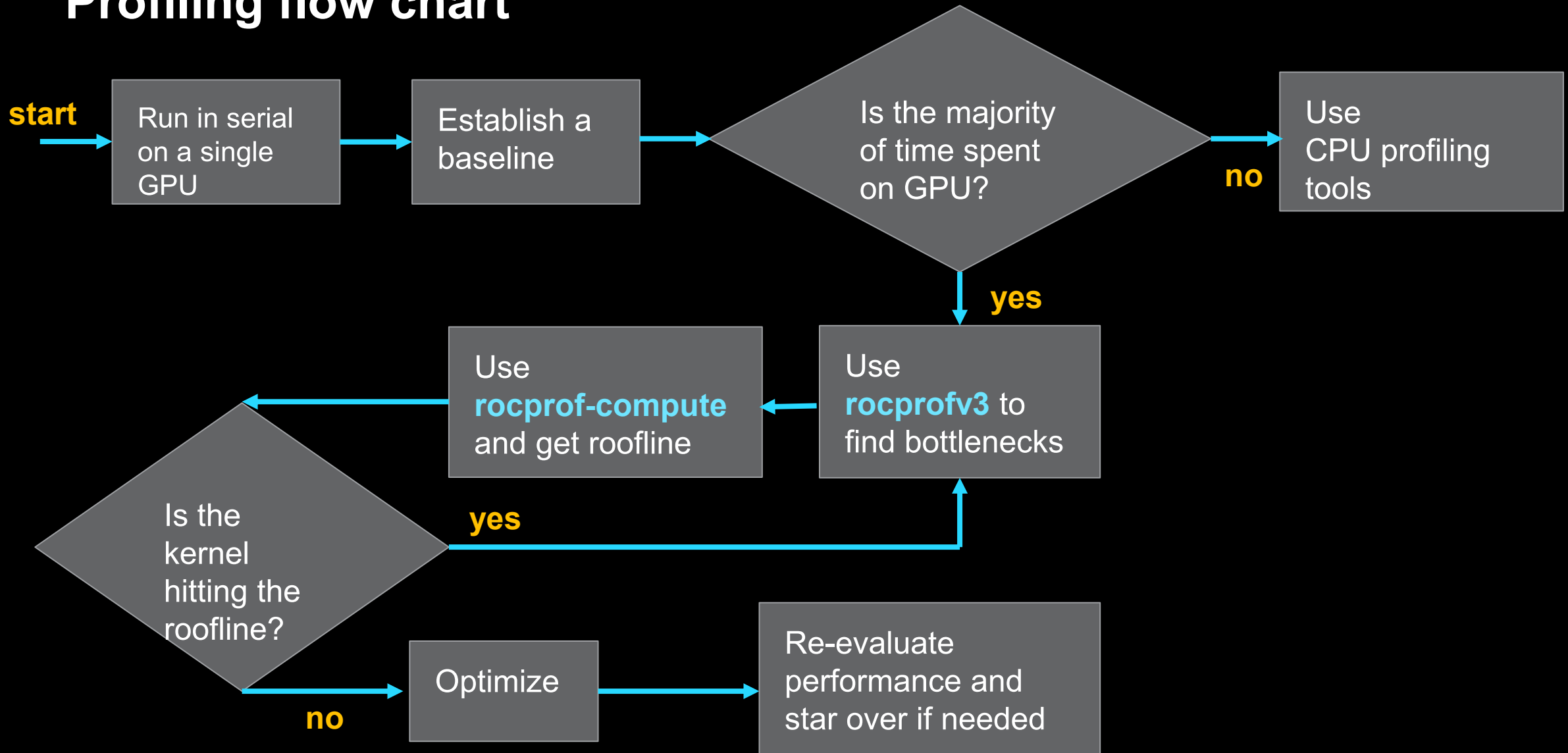
	A	B	C	D	E
1 Name	Calls	TotalDura	AverageN:	Percentage	
2 hipMemcpyAsync	99	3.22E+10	3.25E+08	44.14872	
3 hipEventSynchronize	330	2.42E+10	73394557	33.225	
4 hipMemsetAsync	87	7.76E+09	89232696	10.64953	
5 hipHostMalloc	9	5.41E+09	6.01E+08	7.415198	
6 hipDeviceSynchronize	28	1.32E+09	47006288	1.805515	
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 __hipPopCallConfigur	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	



AMD has three GPU profiling tools



Profiling flow chart



Establishing the baseline

```
git clone https://github.com/amd/HPCTrainingExamples.git
cd HPCTrainingExamples/HIP/jacobi
module load rocm amdclang openmpi
make -j
./Jacobi_hip -g 1 1
```

```
Topology size: 1 x 1
Local domain size (current node): 4096 x 4096
Global domain size (all nodes): 4096 x 4096
Rank 0 selecting device 0 on host ppac-pl1-s24-16
Starting Jacobi run.
Iteration: 0 - Residual: 0.022108
Iteration: 100 - Residual: 0.000625
Iteration: 200 - Residual: 0.000371
Iteration: 300 - Residual: 0.000274
Iteration: 400 - Residual: 0.000221
Iteration: 500 - Residual: 0.000187
Iteration: 600 - Residual: 0.000163
Iteration: 700 - Residual: 0.000145
Iteration: 800 - Residual: 0.000131
Iteration: 900 - Residual: 0.000120
Iteration: 1000 - Residual: 0.000111
Stopped after 1000 iterations with residue 0.000111
Total Jacobi run time: 0.8103 sec.
Measured lattice updates: 20.70 GLU/s (total), 20.70 GLU/s (per process)
Measured FLOPS: 351.96 GFLOPS (total), 351.96 GFLOPS (per process)
Measured device bandwidth: 1.99 TB/s (total), 1.99 TB/s (per process)
```

this is the baseline

Identifying the bottlenecks with rocprofv3

First, check that rocprofv3 is available and is taken from the ROCM_PATH

```
gcapodag@ppac-pl1-s24-16:~/repos/HPCTrainingExamples/HIP/jacobi$ which rocprofv3
/shared/apps/ubuntu/opt/rocm-6.4.1/bin/rocprofv3
```

Then, run the command below which will provide a summary of kernel activity on the GPU for the Jacobi example

```
rocprofv3 --kernel-trace --stats --summary --truncate-kernels --output-format csv
--output-directory output_dir -- ./Jacobi_hip -g 1 1
```

Additional rocprofv3 output displayed after the one shown in the previous slide:

```
E20251027 15:03:07.894726 136371406862592 output_stream.cpp:105] Opened result file: output_dir/ppac-pl1-s24-16/1195398_kernel_stats.csv
E20251027 15:03:07.897497 136371406862592 output_stream.cpp:105] Opened result file: output_dir/ppac-pl1-s24-16/1195398_kernel_trace.csv
E20251027 15:03:07.952638 136371406862592 output_stream.cpp:105] Opened result file: output_dir/ppac-pl1-s24-16/1195398_agent_info.csv
E20251027 15:03:07.955207 136371406862592 output_stream.cpp:105] Opened result file: output_dir/ppac-pl1-s24-16/1195398_domain_stats.csv
```

ROCPROFV3 SUMMARY:

NAME	DOMAIN	CALLS	DURATION (nsec)	AVERAGE (nsec)	PERCENT (INC)	MIN (nsec)	MAX (nsec)	STDDEV
NormKernel1(int, double, double, double const*, double*)	KERNEL_DISPATCH	1001	372710831	3.723e+05	48.765069	368682	387442	1.679e+03
JacobiIterationKernel(int, double, double, double const*, double const*, double*, double*)	KERNEL_DISPATCH	1000	208715494	2.087e+05	27.308102	185281	236602	7.921e+03
LocalLaplacianKernel(int, int, int, double, double, double const*, double*)	KERNEL_DISPATCH	1000	168627641	1.686e+05	22.063052	142681	198201	8.366e+03
HaloLaplacianKernel(int, int, int, double, double, double const*, double const*, double*)	KERNEL_DISPATCH	1000	9219780	9.220e+03	1.206306	8321	13400	5.671e+02
__amd_rocclr_copyBuffer	KERNEL_DISPATCH	1001	2578227	2.576e+03	0.337332	1880	4520	2.424e+02
NormKernel2(int, double const*, double*)	KERNEL_DISPATCH	1001	2443780	2.441e+03	0.319741	2120	3560	2.258e+02
__amd_rocclr_fillBufferAligned	KERNEL_DISPATCH	1	3040	3.040e+03	0.000398	3040	3040	0.000e+00

Analysis of rocprofv3 command

rocprofv3 **--kernel-trace**

For collecting Kernel Dispatch Traces

--stats

For collecting statistics of enabled tracing types. Must be combined with one or more tracing options. No default kernel stats unlike previous rocprof versions

--summary

Output single summary of tracing data at the conclusion of the profiling session

--truncate-kernels

Truncate the demangled kernel names. In earlier rocprof versions, this was known as `--basenames [on/off]`

--output-directory output_dir

For adding output path where the output files will be saved. If nothing specified default path is ``%hostname%/pid%``

--output-format csv

For adding output format (supported formats: csv, json, pftrace, otf2, rocpd)

-- ./Jacobi_hip -g 1 1

Analyzing rocprofv3 data

```

E20251023 13:46:04.459158 136071387085056 output_stream.cpp:105] Opened result file: outdir/jacobi_kernel_stats.csv
E20251023 13:46:04.461616 136071387085056 output_stream.cpp:105] Opened result file: outdir/jacobi_kernel_trace.csv
E20251023 13:46:04.501219 136071387085056 output_stream.cpp:105] Opened result file: outdir/jacobi_agent_info.csv
E20251023 13:46:04.503833 136071387085056 output_stream.cpp:105] Opened result file: outdir/jacobi_domain_stats.csv

```

ROCPROFV3 SUMMARY:

NAME	DOMAIN	CALLS	DURATION (nsec)	AVERAGE (nsec)	PERCENT (INC)	MIN (nsec)	MAX (nsec)	STDDEV
NormKernel1	KERNEL_DISPATCH	1001	369918437	3.695e+05	47.820858	365482	387642	2.186e+03
JacobiIterationKernel	KERNEL_DISPATCH	1000	215790073	2.158e+05	27.896059	192521	246522	8.323e+03
LocalLaplacianKernel	KERNEL_DISPATCH	1000	171197017	1.712e+05	22.131333	146321	196762	8.786e+03
HaloLaplacianKernel	KERNEL_DISPATCH	1000	9649622	9.650e+03	1.247446	8200	13880	9.085e+02
NormKernel2	KERNEL_DISPATCH	1001	4123387	4.119e+03	0.533047	2320	6920	7.465e+02
__amd_rocclr_copyBuffer	KERNEL_DISPATCH	1001	2869142	2.866e+03	0.370906	2080	4360	2.788e+02
__amd_rocclr_fillBufferAligned	KERNEL_DISPATCH	1	2720	2.720e+03	0.000352	2720	2720	0.000e+00

```

Topology size: 1 x 1
Local domain size (current node): 4096 x 4096
Global domain size (all nodes): 4096 x 4096
Rank 0 selecting device 0 on host ppac-pl1-s24-16
Starting Jacobi run.
Iteration: 0 - Residual: 0.022108
Iteration: 100 - Residual: 0.000625
Iteration: 200 - Residual: 0.000371
Iteration: 300 - Residual: 0.000274
Iteration: 400 - Residual: 0.000221
Iteration: 500 - Residual: 0.000187
Iteration: 600 - Residual: 0.000163
Iteration: 700 - Residual: 0.000145
Iteration: 800 - Residual: 0.000131
Iteration: 900 - Residual: 0.000120
Iteration: 1000 - Residual: 0.000111

```

Stopped after 1000 iterations with residue 0.000111

Total Jacobi run time: 0.8103 sec.

```

Measured lattice updates: 20.70 GLU/s (total), 20.70 GLU/s (per process)
Measured FLOPS: 351.96 GFLOPS (total), 351.96 GFLOPS (per process)
Measured device bandwidth: 1.99 TB/s (total), 1.99 TB/s (per process)

```

Approx: **0.779403662 sec**

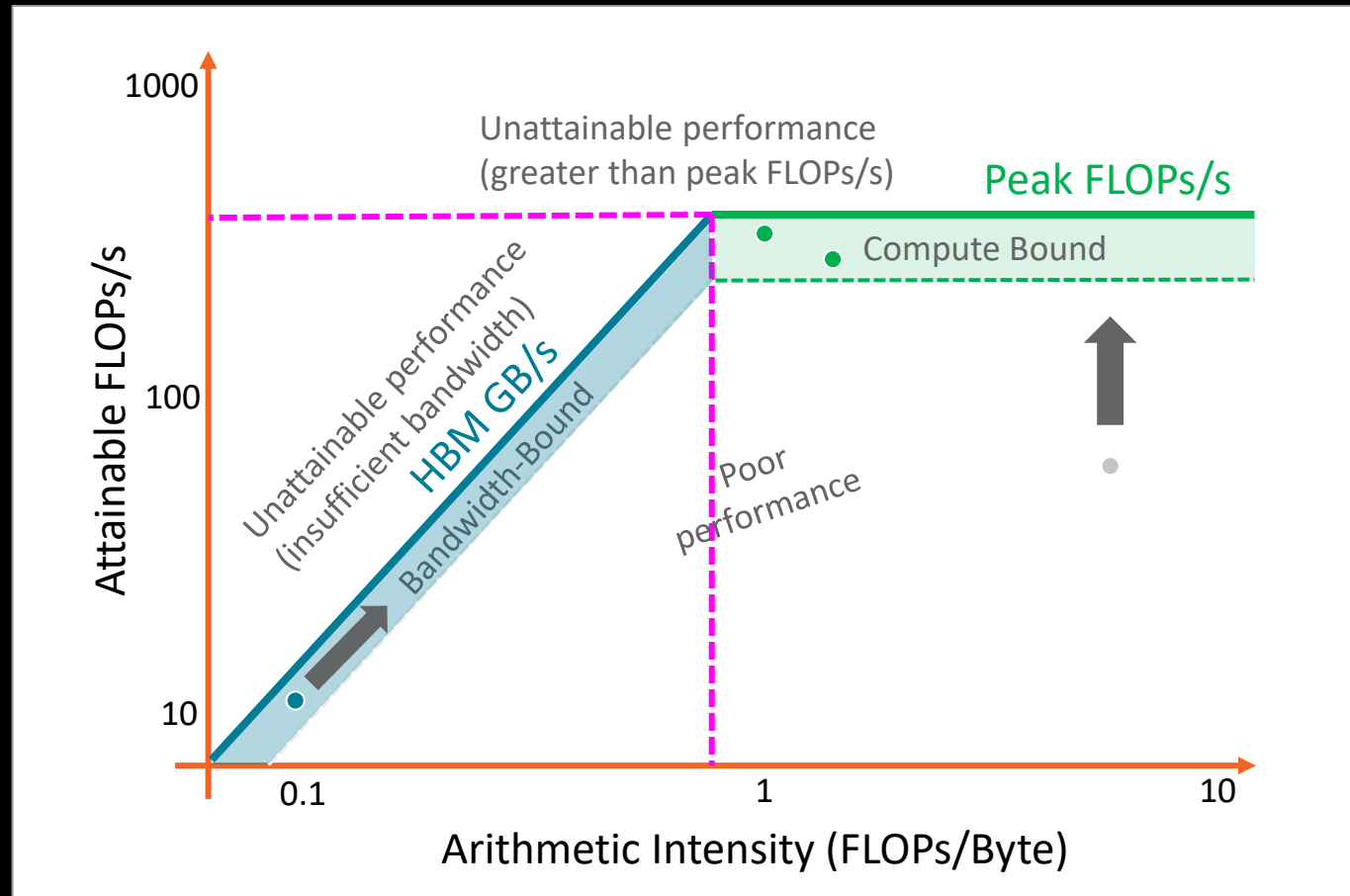
Time spent on GPU

About **96%** of the overall time is spent on GPU

Total time of simulation

Background – What is roofline?

- Attainable FLOPs/s =
 - $\min \left\{ \begin{array}{l} \text{Peak FLOPs/s} \\ AI * \text{Peak GB/s} \end{array} \right.$
- Machine balance:
 - Where $AI = \frac{\text{Peak FLOPs/s}}{\text{Peak GB/s}}$
- Five performance regions:
 - Unattainable compute
 - Unattainable bandwidth
 - Compute bound
 - Bandwidth bound
 - Poor performance



Use rocprof-compute to get a roofline

ROC_PROF_V3 SUMMARY:

NAME	DOMAIN	CALLS	DURATION (nsec)	AVERAGE (nsec)	PERCENT (INC)	MIN (nsec)	MAX (nsec)	STDDEV
NormKernel1	KERNEL_DISPATCH	1001	369918437	3.695e+05	47.820858	365482	387642	2.186e+03
JacobiIterationKernel	KERNEL_DISPATCH	1000	215790073	2.158e+05	27.896059	192521	246522	8.323e+03
LocalLaplacianKernel	KERNEL_DISPATCH	1000	171197017	1.712e+05	22.131333	146321	196762	8.786e+03
HaloLaplacianKernel	KERNEL_DISPATCH	1000	9649622	9.650e+03	1.247446	8200	13880	9.085e+02
NormKernel2	KERNEL_DISPATCH	1001	4123387	4.119e+03	0.533047	2320	6920	7.465e+02
__amd_rocclr_copyBuffer	KERNEL_DISPATCH	1001	2869142	2.866e+03	0.370906	2080	4360	2.788e+02
__amd_rocclr_fillBufferAligned	KERNEL_DISPATCH	1	2720	2.720e+03	0.000352	2720	2720	0.000e+00

module load rocm amdclang openmpi rocprofiler-compute

rocprof-compute profile --name jacobi

Assign a name to workload

--roof-only

Profile roofline data only

--device 0

Target GPU device ID. (DEFAULT: ALL)

--kernel NormKernel1

Kernel filtering

-- ./Jacobi_hip -g 1 1

Roofline for NormKernel11



What is occupancy

Occupancy = # resident wavefronts / maximum # wavefronts the GPU can have in-flight

From rocminfo on MI300A

```
Compute Unit:          228
SIMDs per CU:          4
Shader Engines:        24
Shader Arrs. per Eng.: 1
WatchPts on Addr. Ranges:4
Coherent Host Access:  TRUE
Memory Properties:     APU
Features:              KERNEL_DISPATCH
Fast F16 Operation:    TRUE
Wavefront Size:        64(0x40)
Workgroup Max Size:    1024(0x400)
Workgroup Max Size per Dimension:
  x                    1024(0x400)
  y                    1024(0x400)
  z                    1024(0x400)
Max Waves Per CU:      32(0x20)
Max Work-item Per CU:  2048(0x800)
Grid Max Size:         4294967295(0xffffffff)
Grid Max Size per Dimension:
  x                    2147483647(0x7fffffff)
  y                    65535(0xffff)
  z                    65535(0xffff)
```

Note: higher occupancy can help hide latency

A wave is 64 work items (threads) so Max Waves per CU = 2048 / 64 = 32

Analyze hardware resource usage

performance monitoring counters

```
rocprofv3 --pmc OccupancyPercent --truncate-kernels --output-directory output_dir
--output-format csv -- ./Jacobi_hip -g 1 1
```

```
cd output_dir/ppac-pl1-s24-16
```

```
cat *.csv | grep NormKernel1
```

```

gcapodag@ppac-pl1-s24-16:~/repos/HPCTrainingExamples/HIP/Jacobi/output_dir/ppac-pl1-s24-16$ cat *.csv | grep NormKernel1
2,2,"Agent 4",4,1198678,1198678,16384,12,"NormKernel1",128,1024,0,8,0,32,"OccupancyPercent",3.399454,983598106356863,983598106741105
8,8,"Agent 4",4,1198678,1198678,16384,12,"NormKernel1",128,1024,0,8,0,32,"OccupancyPercent",3.372814,983598108197474,983598108564796
14,14,"Agent 4",4,1198678,1198678,16384,12,"NormKernel1",128,1024,0,8,0,32,"OccupancyPercent",3.382125,983598109215480,983598109581482
20,20,"Agent 4",4,1198678,1198678,16384,12,"NormKernel1",128,1024,0,8,0,32,"OccupancyPercent",3.386065,983598110215967,983598110582129
26,26,"Agent 4",4,1198678,1198678,16384,12,"NormKernel1",128,1024,0,8,0,32,"OccupancyPercent",3.387556,983598111214453,983598111579655
32,32,"Agent 4",4,1198678,1198678,16384,12,"NormKernel1",128,1024,0,8,0,32,"OccupancyPercent",3.397065,983598112209059,983598112572662
38,38,"Agent 4",4,1198678,1198678,16384,12,"NormKernel1",128,1024,0,8,0,32,"OccupancyPercent",3.391827,983598113170305,983598113535948
44,44,"Agent 4",4,1198678,1198678,16384,12,"NormKernel1",128,1024,0,8,0,32,"OccupancyPercent",3.376448,983598114175272,983598114541994

```

LDS block size SGPR (count)
Workgroup size VGPR (count)

The **occupancy is low** and the VGPR and SGPR usage is also low (each CU in MI300A has a maximum of 800 32-bit SGPRs available per wavefront), this means the low occupancy is not caused by register pressure:

<https://rocm.blogs.amd.com/software-tools-optimization/register-pressure/README.html>

Optimize the NormKernel1 kernel

```

7 // #define OPTIMIZED
8
9 #ifdef OPTIMIZED
10 #define block_size 1024
11 #else
12 #define block_size 128
13 #endif
14
15 __global__ void NormKernel1(const int N,
16                             const dfloat dx, const dfloat dy,
17                             const dfloat*__restrict__ U,
18                             dfloat*__restrict__ tmp) {
19     __shared__ dfloat s_dot[block_size];
20
21     const int t = threadIdx.x;
22     const int i = blockIdx.x;
23
24     int id = i * block_size + t;
25
26     s_dot[t] = 0.0;
27     for ( ; id < N ; id += gridDim.x * block_size ) {
28         s_dot[t] += U[id] * U[id] * dx * dy;
29     }
30     __syncthreads();
31
32     for (int k = block_size / 2; k > 0; k /= 2 ) {
33         if ( t < k ) {
34             s_dot[t] += s_dot[t + k];
35         }
36         __syncthreads();
37     }
38
39     if (t==0)
40         tmp[i] = s_dot[0];
41 }
42

```

see [HIP/jacobi/Norm.hip](#)

```

84 size_t grid_size = (mesh.N+block_size-1)/block_size;
85 grid_size = (grid_size < block_size) ? grid_size : block_size;
86
87 hipLaunchKernelGGL((NormKernel1),
88                    dim3(grid_size),
89                    dim3(block_size),
90                    0, stream,
91                    mesh.N, mesh.dx, mesh.dy, U, d_tmp);
92

```

Each CU has 64KiB of LDS available

LDS is shared among workgroups

We are currently using a workgroup size (`block_size`) of 128 (also seen in previous slide) to allocate `block_size` floats so:

$128 \text{ [float]} * 8 \text{ [bytes/float]} = 1024 \text{ bytes} = 1 \text{ KiB}$

hence if we had no limit on the number of workgroups per CU, we could use a max of 64 workgroups per CU with this LDS usage

but

with 2048 being the max # of threads per CU and 128 being the current workgroup size, we can have 16 workgroups per CU max

Proposed optimization

block_size = 128 wavefronts per workgroup: $128 / 64 = 2$

allocated LDS per workgroup:

$128 \text{ [float]} * 8 \text{ [bytes/float]} = 1024 \text{ bytes} = 1 \text{ KiB}$

total LDS available on MI300A = 64KiB (with no constraints, could use 64 workgroups per CU max)

max workgroups per CU:

$2048 \text{ (max \# of threads / CU)} / 128 \text{ (\# threads / block)} = 16 \text{ (\# workgroup / CU)}$

Current implementation

Note: both cases have a max of 32 waves per CU (in theory)

block_size = 1024 wavefronts per workgroup: $1024 / 64 = 16$

allocated LDS per workgroup:

$1024 \text{ [float]} * 8 \text{ [bytes/float]} = 8192 \text{ bytes} = 8 \text{ KiB}$

total LDS available on MI300A = 64KiB (with no constraints, could use 8 workgroups per CU max)

max workgroups per CU:

$2048 \text{ (max \# of threads / CU)} / 1024 \text{ (\# threads / block)} = 2 \text{ (\# workgroup / CU)}$

Improved implementation

The max number of waves per CU is the same (32) but in the second case there are more waves per workgroup that can share LDS and hide latency

Evaluate performance after optimization -- occupancy

Uncomment line 7 in Norm.hip, compile and run again:

```
rocprofv3 --pmc OccupancyPercent --truncate-kernels --output-directory output_dir2  
--output-format csv -- ./Jacobi_hip -g 1 1
```

```
cd output_dir2/ppac-pl1-s24-16
```

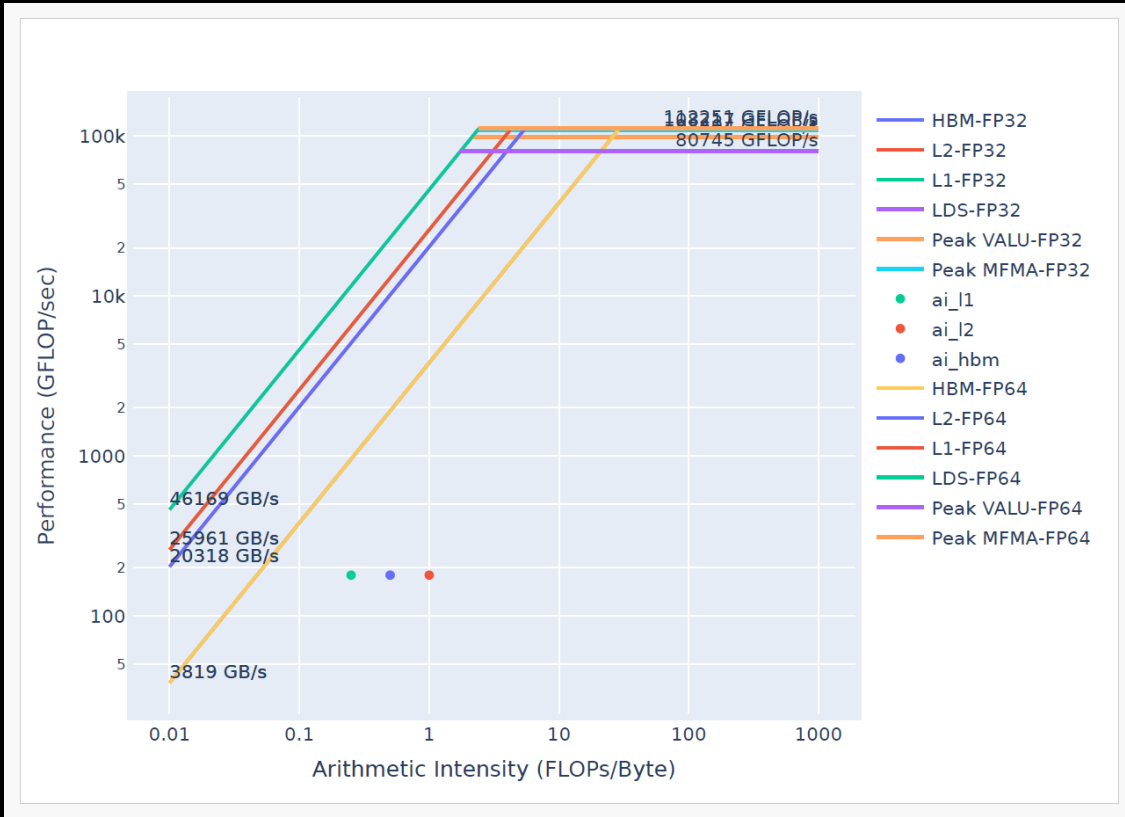
```
cat *.csv | grep NormKernel1
```

```
2,2,"Agent 4",4,1208534,1208534,1048576,12,"NormKernel1",1024,8192,0,8,0,32,"OccupancyPercent",72.160256,1009838126887412,1009838126933852  
8,8,"Agent 4",4,1208534,1208534,1048576,12,"NormKernel1",1024,8192,0,8,0,32,"OccupancyPercent",71.750717,1009838128583062,1009838128635662  
14,14,"Agent 4",4,1208534,1208534,1048576,12,"NormKernel1",1024,8192,0,8,0,32,"OccupancyPercent",68.789347,1009838129286106,1009838129335787  
20,20,"Agent 4",4,1208534,1208534,1048576,12,"NormKernel1",1024,8192,0,8,0,32,"OccupancyPercent",70.380214,1009838129988551,1009838130036311  
26,26,"Agent 4",4,1208534,1208534,1048576,12,"NormKernel1",1024,8192,0,8,0,32,"OccupancyPercent",69.907577,1009838130671355,1009838130719955  
32,32,"Agent 4",4,1208534,1208534,1048576,12,"NormKernel1",1024,8192,0,8,0,32,"OccupancyPercent",69.133842,1009838131352439,1009838131401440  
38,38,"Agent 4",4,1208534,1208534,1048576,12,"NormKernel1",1024,8192,0,8,0,32,"OccupancyPercent",69.910233,1009838132026284,1009838132074524  
44,44,"Agent 4",4,1208534,1208534,1048576,12,"NormKernel1",1024,8192,0,8,0,32,"OccupancyPercent",69.628028,1009838132684528,1009838132731648
```

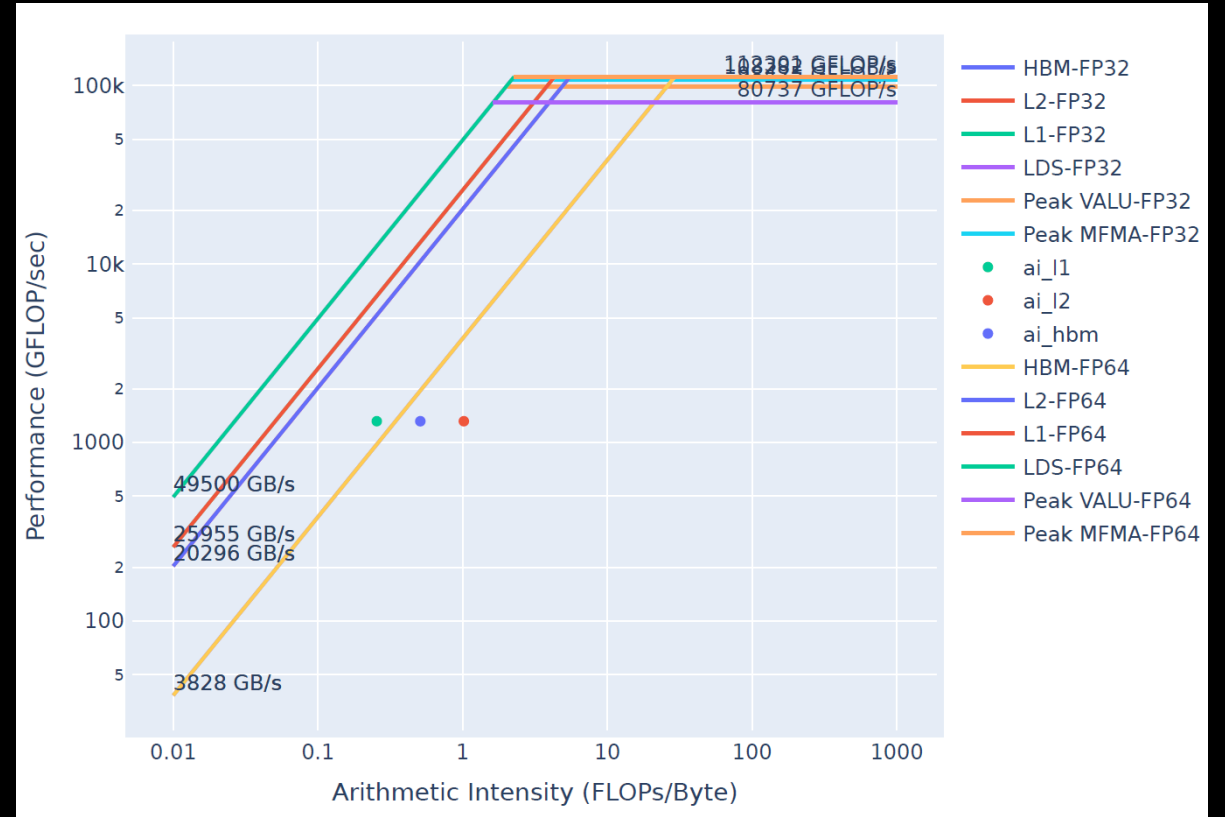
The **occupancy is now around 70%** (it was around 3.3% before)

Evaluate performance after optimization -- roofline

Before (block_size 128)



After (block_size 1024)



Evaluate performance after optimization -- timings

Before (block_size 128)

```
Topology size: 1 x 1
Local domain size (current node): 4096 x 4096
Global domain size (all nodes): 4096 x 4096
Rank 0 selecting device 0 on host ppac-pl1-s24-16
Starting Jacobi run.
Iteration: 0 - Residual: 0.022108
Iteration: 100 - Residual: 0.000625
Iteration: 200 - Residual: 0.000371
Iteration: 300 - Residual: 0.000274
Iteration: 400 - Residual: 0.000221
Iteration: 500 - Residual: 0.000187
Iteration: 600 - Residual: 0.000163
Iteration: 700 - Residual: 0.000145
Iteration: 800 - Residual: 0.000131
Iteration: 900 - Residual: 0.000120
Iteration: 1000 - Residual: 0.000111
Stopped after 1000 iterations with residue 0.000111
Total Jacobi run time: 0.8103 sec.
Measured lattice updates: 20.70 GLU/s (total), 20.70 GLU/s (per process)
Measured FLOPS: 351.96 GFLOPS (total), 351.96 GFLOPS (per process)
Measured device bandwidth: 1.99 TB/s (total), 1.99 TB/s (per process)
```

After (block_size 1024)

```
Topology size: 1 x 1
Local domain size (current node): 4096 x 4096
Global domain size (all nodes): 4096 x 4096
Rank 0 selecting device 0 on host ppac-pl1-s24-16
Starting Jacobi run.
Iteration: 0 - Residual: 0.022108
Iteration: 100 - Residual: 0.000625
Iteration: 200 - Residual: 0.000371
Iteration: 300 - Residual: 0.000274
Iteration: 400 - Residual: 0.000221
Iteration: 500 - Residual: 0.000187
Iteration: 600 - Residual: 0.000163
Iteration: 700 - Residual: 0.000145
Iteration: 800 - Residual: 0.000131
Iteration: 900 - Residual: 0.000120
Iteration: 1000 - Residual: 0.000111
Stopped after 1000 iterations with residue 0.000111
Total Jacobi run time: 0.4252 sec.
Measured lattice updates: 39.46 GLU/s (total), 39.46 GLU/s (per process)
Measured FLOPS: 670.82 GFLOPS (total), 670.82 GFLOPS (per process)
Measured device bandwidth: 3.79 TB/s (total), 3.79 TB/s (per process)
```

There has been a **2x speedup in time to solution**

Additional resources

Three-part profiling guide on ROCm blogs:

❖ Intro:

<https://rocm.blogs.amd.com/software-tools-optimization/profiling-guide/intro/README.html>

❖ Novice (the one shown here):

<https://rocm.blogs.amd.com/software-tools-optimization/profiling-guide/novice/README.html>

❖ Advanced:

<https://rocm.blogs.amd.com/software-tools-optimization/profiling-guide/advanced/README.html>



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.

© 2025 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, AMD CDNA, AMD ROCm, AMD Instinct, and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.

Git and the Git logo are either registered trademarks or trademarks of Software Freedom Conservancy, Inc., corporate home of the Git Project, in the United States and/or other countries

AMD 