VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

TECHNICAL FEATURES

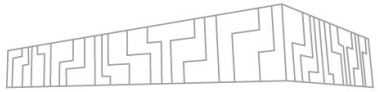
AND

THE USE OF GPU ACCELERATED PARTITION

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education

MINISTRY OF EDUCATION, YOUTH AND SPORTS

GPU ACCELERATED PARTITION

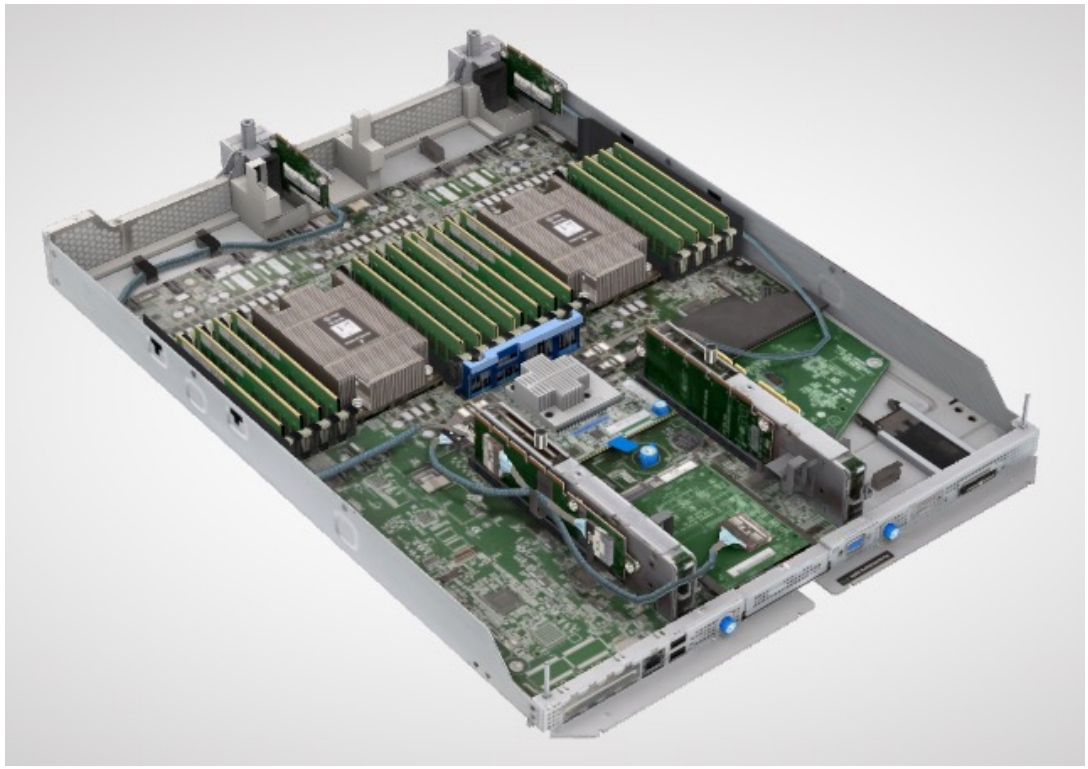


- Key properties
  - 72 nodes, each with
    - 2x AMD EPYC™ 7763, 64-core, 2.45 GHz processors
    - 1024 GB DDR4 3200MT/s of physical memory
    - 8x GPU accelerator NVIDIA A100
      - 40GB HBM2 memory per GPU
      - 320GB HBM2 memory in total
    - 4x 200 Gb/s Infiniband HDR links

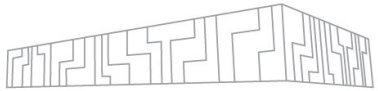
- In PBS
  - PBS queue: **qnvidia**
  - name of nodes: **acn[01-72]**

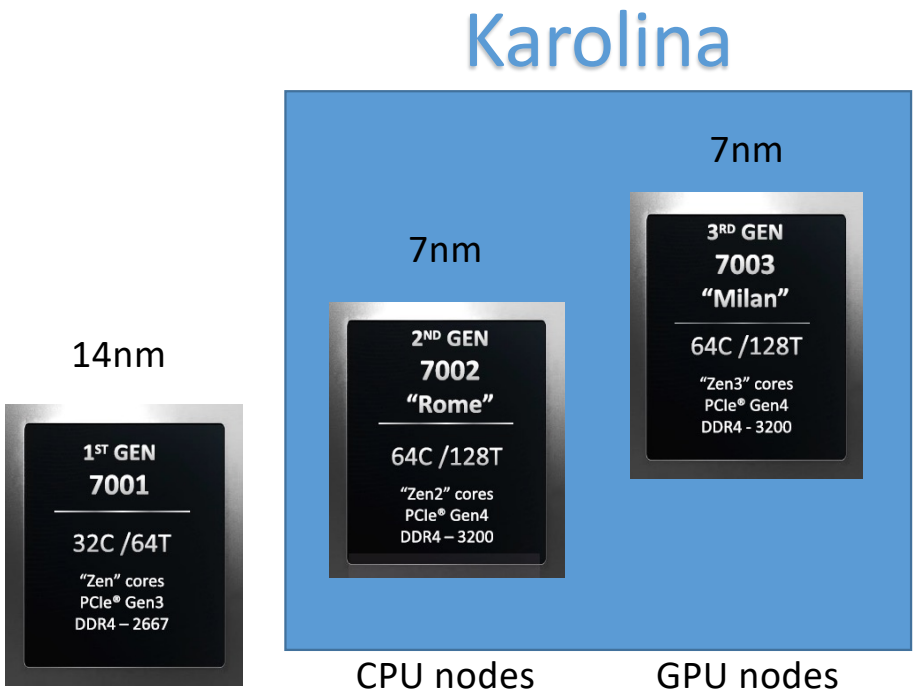
GPU ACCELERATED PARTITION

- CPU part of the node
  - 2x AMD EPYC™ 7763,
    - 64-core per socket
    - 2.45 GHz clock frequency
  - Memory
    - 1024 GB DDR4 3200MT/s
  - 4x 200 Gb/s Infiniband HDR links



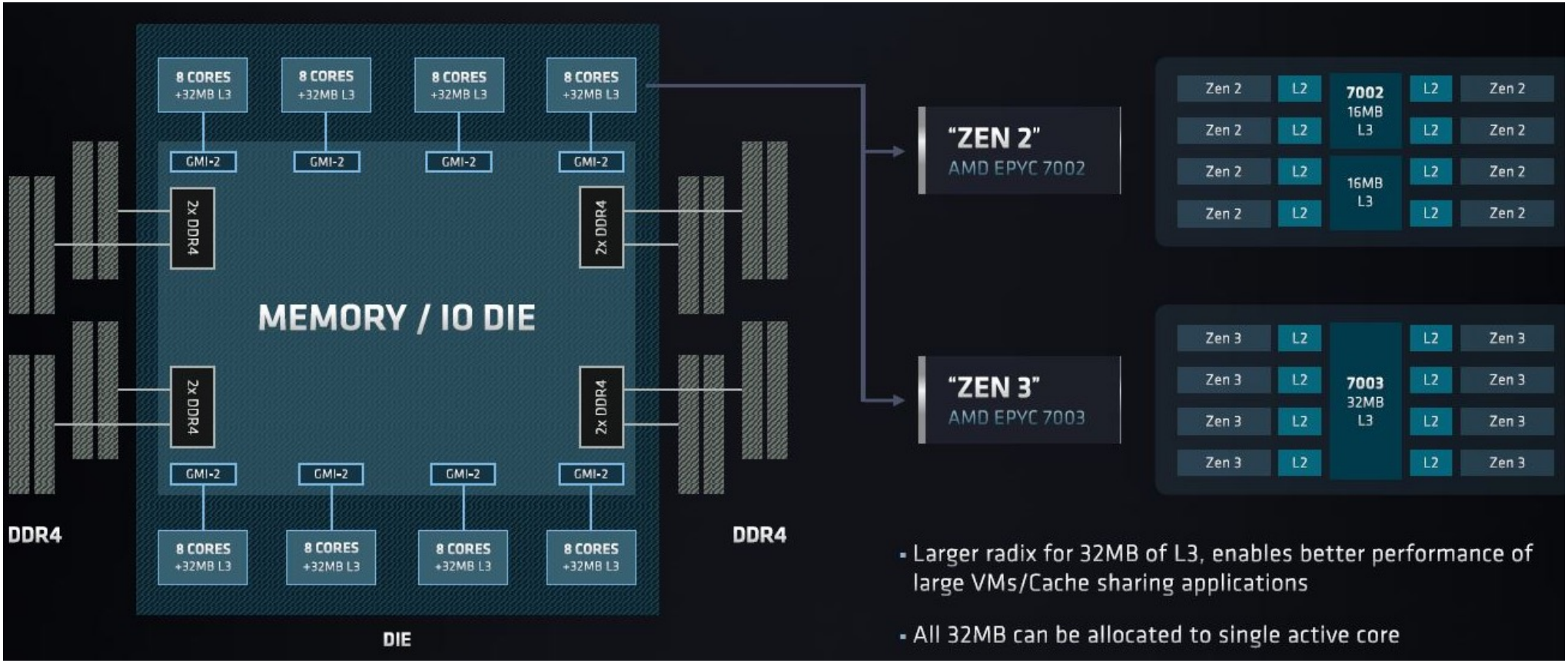
AMD EPYC SERVER CPU ROADMAP



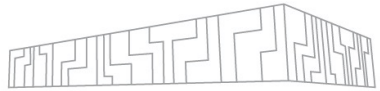


CATEGORY	EPYC 7002 (Rome)	EPYC 7003 (Milan)
Socket	SP3	SP3
Core / Process	Zen2 / 7nm	Zen3 / 7nm
Max Core Count / Threads	64 / 128	64 / 128
L3 Cache Size	256 MB	256 MB
CCX Arch	4 Cores + 16MB	8 Cores + 32MB
Memory	8 Ch DDR4-3200, NVDIMM-N	8 Ch DDR4-3200, NVDIMM-N
PCIe Tech & Lane Count	PCIe Gen4, 128L/Socket	PCIe Gen4, 128L/Socket
Security	SME, SEV	SME, SEV, SNP
Chipset	NA	NA
Power	120W - 280W	120W - 280W

“MILAN” BUILDS ON INFINITY ARCHITECTURE



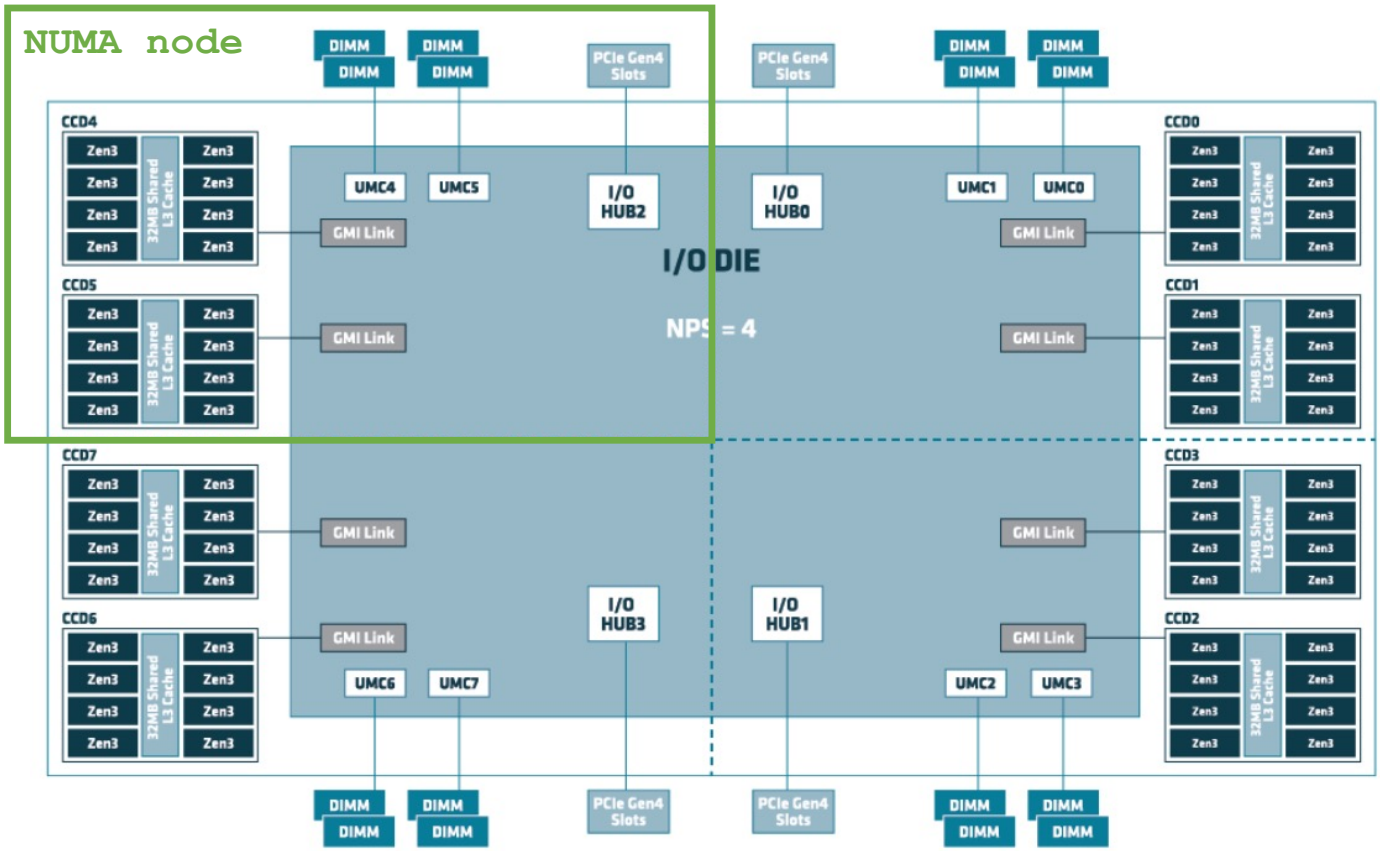
MEMORY BOUND WORKLOAD ON CPU



numactl -H

```
| node 0 cpus: 0   -  15
| node 1 cpus: 16  -  31
| node 2 cpus: 32  -  47
| node 3 cpus: 48  -  63
| node 4 cpus: 64  -  79
| node 5 cpus: 80  -  95
| node 6 cpus: 96  - 111
| node 7 cpus: 112 - 127
| node 0-7 size: 128 GB
```

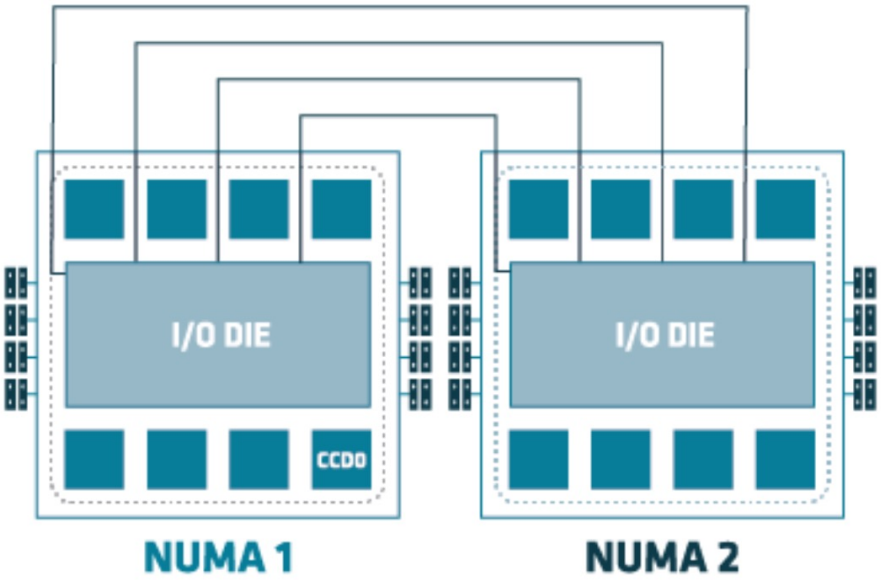
	0	1	2	3
0	10	12	12	12
1	12	10	12	12
2	12	12	10	12
3	12	12	12	10



https://developer.amd.com/spack/stream-benchmark/

DUAL-SOCKET CONFIGURATIONS (MILAN)

- Two EPYC 7003 Processors connect through 4 xGMI links



	0	1	2	3	4	5	6	7
0	10	12	12	12	32	32	32	32
1	12	10	12	12	32	32	32	32
2	12	12	10	12	32	32	32	32
3	12	12	12	10	32	32	32	32
4	32	32	32	32	10	12	12	12
5	32	32	32	32	12	10	12	12
6	32	32	32	32	12	12	10	12
7	32	32	32	32	12	12	12	10

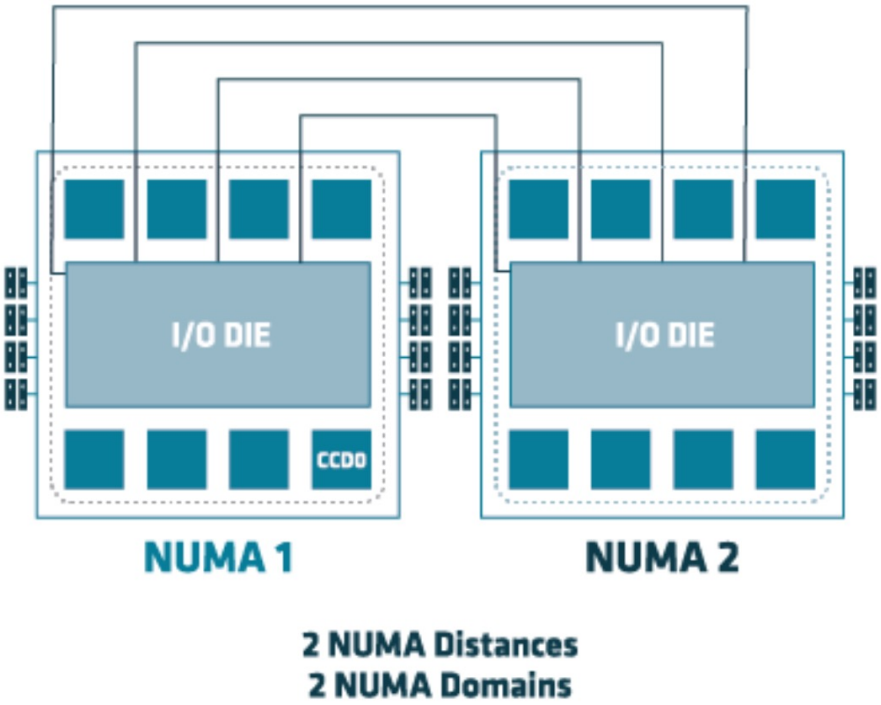
```
[lriha@cn103.barbora ~]$ numactl -H
available: 2 nodes (0-1)

node 0 cpus: 0 - 17
node 0 size: 95197 MB

node 1 cpus: 18 - 35
node 1 size: 96762 MB

node distances:
node   0   1
  0:  10  21
  1:  21  10
```

	0	1
0	10	21
1	21	10

Barbora node

DUAL-SOCKET CONFIGURATIONS (MILAN)

- Two EPYC 7003 Processors connect through 4 xGMI links



Numa	0	1	2	3	4	5	6	7
0	39,5	39,0	38,4	38,0	21,8	21,8	21,8	20,6
1	39,0	39,5	38,0	38,4	21,8	21,0	20,6	21,8
2	38,3	38,0	39,5	39,0	20,4	20,8	21,5	21,5
3	38,0	38,3	38,9	39,5	20,8	20,9	21,7	21,7
4	21,7	21,7	21,7	20,8	39,5	39,0	38,4	38,1
5	21,5	21,5	21,5	20,4	38,9	39,5	38,0	38,4
6	20,7	21,0	21,7	21,7	38,4	38,0	39,5	39,0
7	21,8	21,0	20,7	21,7	38,0	38,4	39,0	39,5

Socket 2

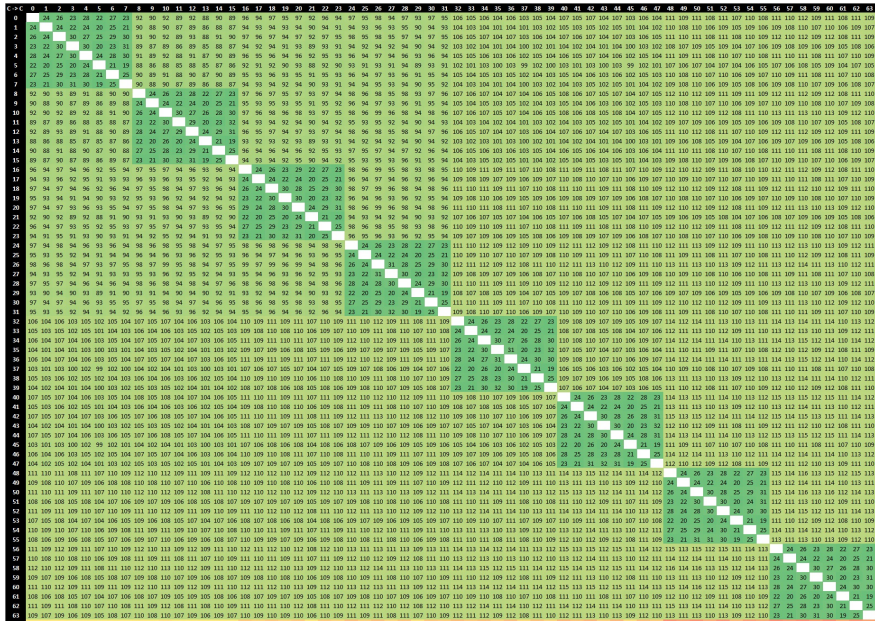
Bandwidth [GB/s]

Numa	0	1	2	3	4	5	6	7
0	90	98,5	107,1	110,4	188,9	192,4	184,1	188,6
1	109,6	91,6	110,8	106	192,5	197,5	190,9	192,4
2	118,7	110,8	91,5	97,9	181,1	190,5	189,6	190,7
3	126,8	106,5	100,8	90,1	193,7	198,5	196,9	201,1
4	204,7	190,6	189	188,8	90	98,4	106,7	110,2
5	206,6	197,6	194,1	194,6	97,9	91,5	110,8	106
6	203,7	189,1	189,5	192,4	106	110,8	91,5	97,9
7	201,3	193,1	193,6	198,4	110,1	106,5	98,6	90,1

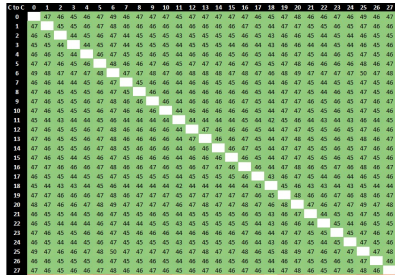
Latency [ns]

Measured by: Inte Memory Latency Checker - v3.9a

CHIPLET BASED VS MONOLITHIC CPU



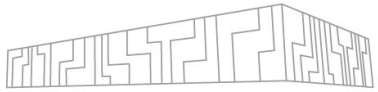
AMD EPYC 7763 (Milan)



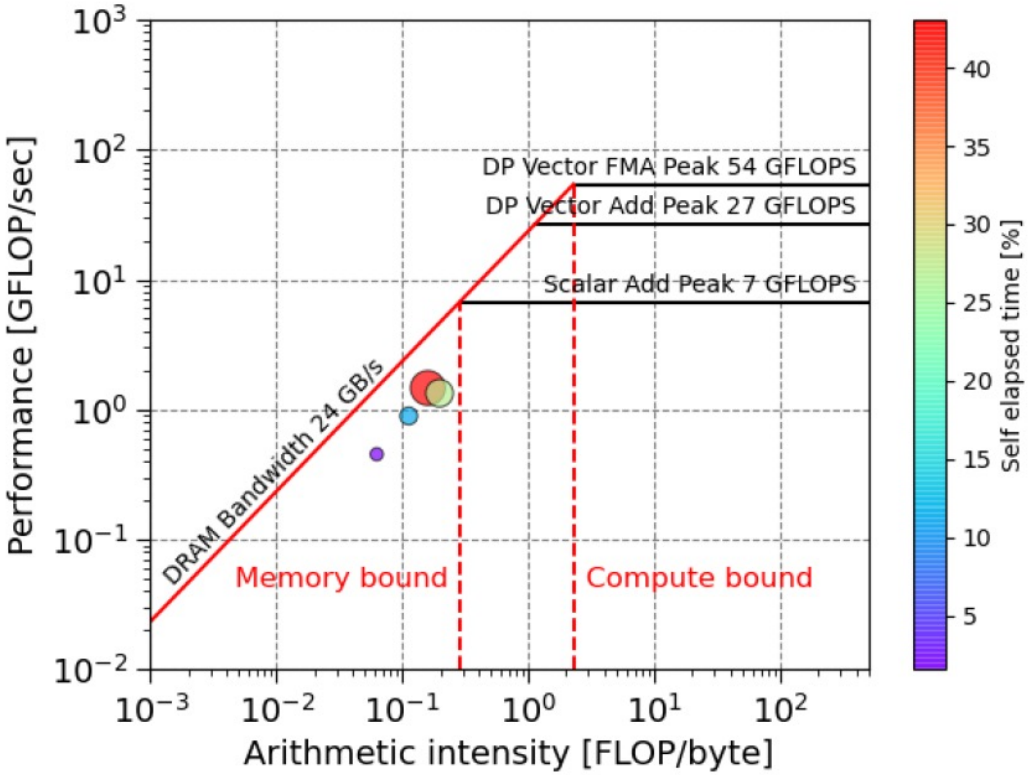
Intel Xeon Platinum 8280

https://www.anandtech.com/show/16529/amd-epyc-milan-review/4

MEMORY BANDWIDTH VS. CLOCK FREQ

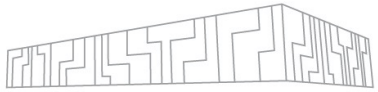


LAMMPS, EMA



- 2 competing system-level choke points:
  - Bandwidth to main memory
  - Compute Bound (frequency)
- These are mutually exclusive to each other
- Perform roofline analysis to confirm where hot-routine lands (red circle)
- It has performed this analysis on a number of popular HPC codes across CFD, Weather, Quantum Chemistry, Molecular Dynamics: Codes are memory bound or borderline
- HPL (compute bound) is *NOT* a good proxy for scoping job throughput on realistic workloads.
- Use memory bound synthetics: HPCG or STREAM

MEMORY BOUND WORKLOAD ON CPU



Compiling STREAM benchmark for AMD CPUs

```
ml AOCC
clang -O3 -fopenmp -mcmodel=large -DSTREAM_TYPE=double -
mavx2 -DSTREAM_ARRAY_SIZE=250000000 -DNTIMES=10 -ffp-
contract=fast -fnt-store stream.c -o stream_c
```

Component/Application	Versions Applicable
STREAM	5.10
AOCC	3.1.0

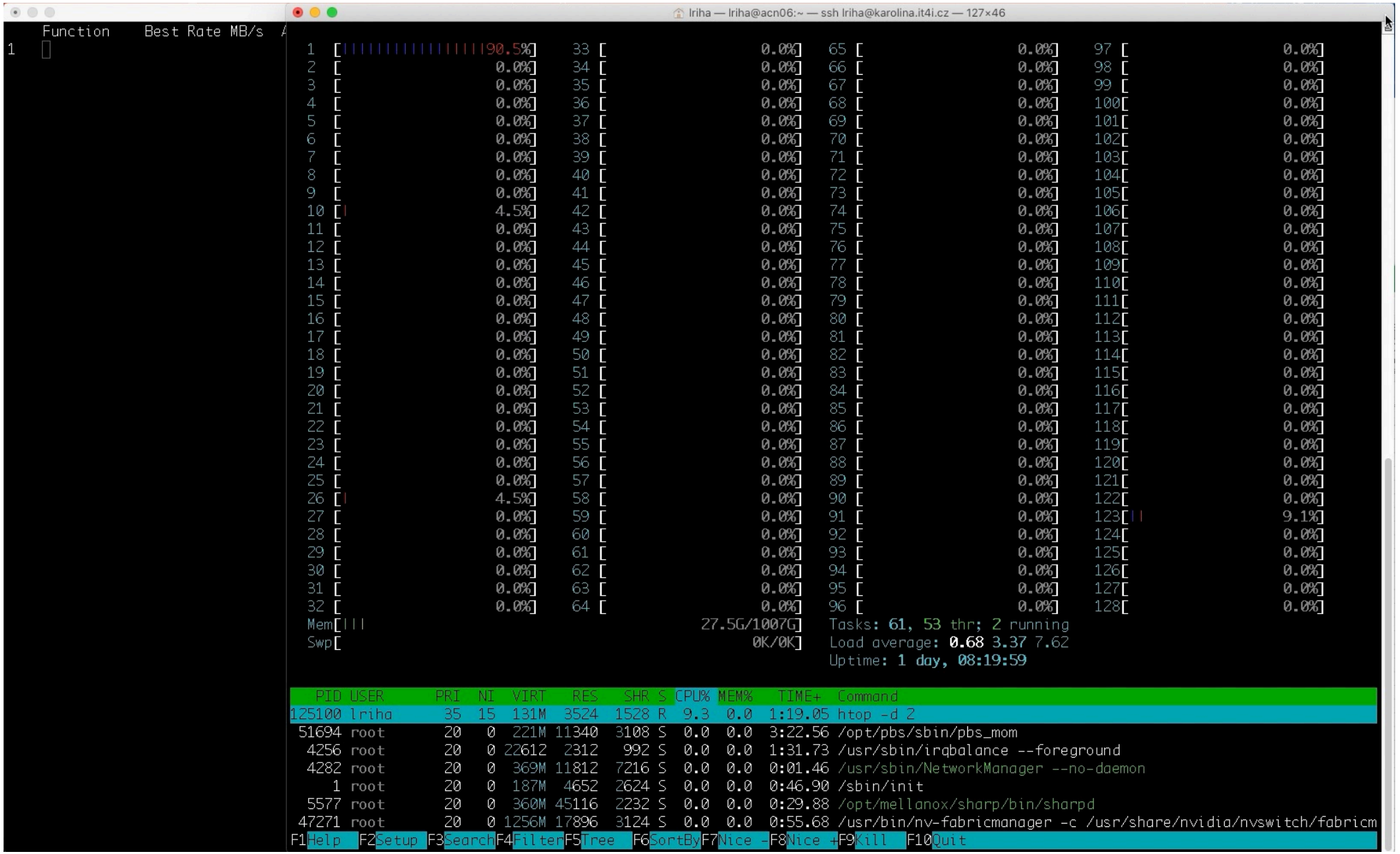
Running STREAM

```
$ export OMP_SCHEDULE=static
$ export OMP_DYNAMIC=false
$ export OMP_THREAD_LIMIT=256
$ export OMP_NESTED=FALSE
$ export OMP_STACKSIZE=256M

# Thread Binding Options for AMD EPYC 7742/7763 Processor
$ export GOMP_CPU_AFFINITY=0-127:8
$ export OMP_NUM_THREADS=16

$ echo "running for 1 thread per CCD"
$ stream_c.exe
```

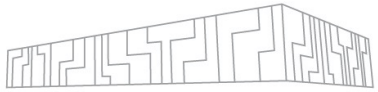
STREAM generally gives the better performance with 1 thread per CCD.
Binding options for AMD EPYC 7742 and AMD EPYC 7763 Processor to bind 1 thread per CCD: - ---
- **export GOMP_CPU_AFFINITY=0-127:8** and
- **export OMP_NUM_THREADS=16**

Basic Details of Flags used:

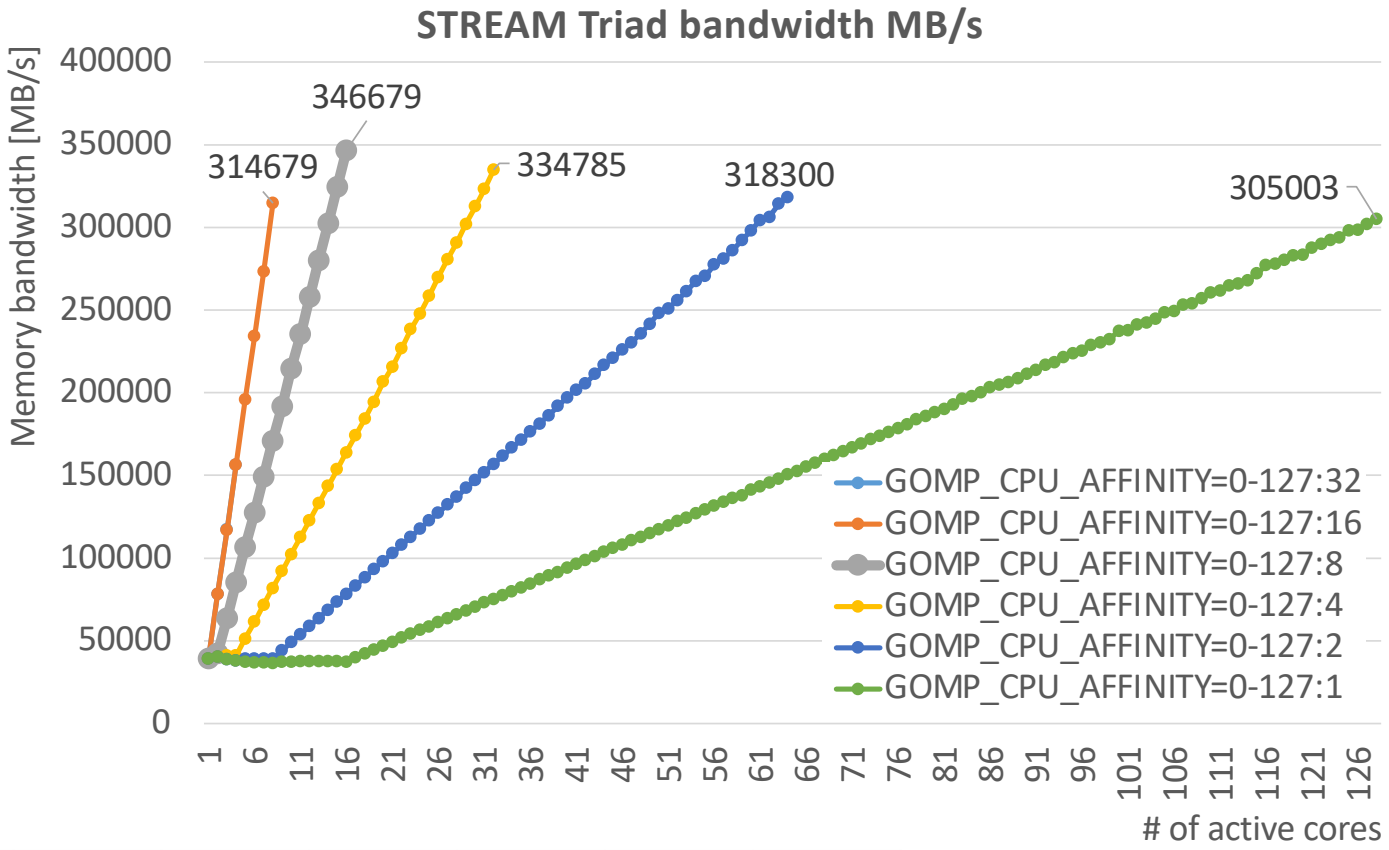
Mcmodel=large	Generate code for the large model. This model makes no assumptions about addresses and sizes of sections.
STREAM_ARRAY_SIZE= "250000000"	Sets the Array size for the STREAM benchmark. General recommendation is that "STREAM_ARRAY_SIZE" must be at least 4x the size of the sum of all the last-level caches in the system.
NTIMES=10	STREAM runs each kernel "NTIMES" times.
ffp-contract=fast	enables floating-point expression contraction such as forming of fused multiply-add operations if the target has native support for them.
fnt-store	generate non-temporal store instruction for array accesses in a loop with large trip count.


lriha — lriha@acn06:~ — ssh lriha@karolina.it4i.cz — 127×46
Function Best Rate MB/s
1 [|||||||||||||||90.5%] 33 [0.0%] 65 [0.0%] 97 [0.0%]
2 [0.0%] 34 [0.0%] 66 [0.0%] 98 [0.0%]
3 [0.0%] 35 [0.0%] 67 [0.0%] 99 [0.0%]
4 [0.0%] 36 [0.0%] 68 [0.0%] 100[0.0%]
5 [0.0%] 37 [0.0%] 69 [0.0%] 101[0.0%]
6 [0.0%] 38 [0.0%] 70 [0.0%] 102[0.0%]
7 [0.0%] 39 [0.0%] 71 [0.0%] 103[0.0%]
8 [0.0%] 40 [0.0%] 72 [0.0%] 104[0.0%]
9 [0.0%] 41 [0.0%] 73 [0.0%] 105[0.0%]
10 [| 4.5%] 42 [0.0%] 74 [0.0%] 106[0.0%]
11 [0.0%] 43 [0.0%] 75 [0.0%] 107[0.0%]
12 [0.0%] 44 [0.0%] 76 [0.0%] 108[0.0%]
13 [0.0%] 45 [0.0%] 77 [0.0%] 109[0.0%]
14 [0.0%] 46 [0.0%] 78 [0.0%] 110[0.0%]
15 [0.0%] 47 [0.0%] 79 [0.0%] 111[0.0%]
16 [0.0%] 48 [0.0%] 80 [0.0%] 112[0.0%]
17 [0.0%] 49 [0.0%] 81 [0.0%] 113[0.0%]
18 [0.0%] 50 [0.0%] 82 [0.0%] 114[0.0%]
19 [0.0%] 51 [0.0%] 83 [0.0%] 115[0.0%]
20 [0.0%] 52 [0.0%] 84 [0.0%] 116[0.0%]
21 [0.0%] 53 [0.0%] 85 [0.0%] 117[0.0%]
22 [0.0%] 54 [0.0%] 86 [0.0%] 118[0.0%]
23 [0.0%] 55 [0.0%] 87 [0.0%] 119[0.0%]
24 [0.0%] 56 [0.0%] 88 [0.0%] 120[0.0%]
25 [0.0%] 57 [0.0%] 89 [0.0%] 121[0.0%]
26 [| 4.5%] 58 [0.0%] 90 [0.0%] 122[0.0%]
27 [0.0%] 59 [0.0%] 91 [0.0%] 123[|| 9.1%]
28 [0.0%] 60 [0.0%] 92 [0.0%] 124[0.0%]
29 [0.0%] 61 [0.0%] 93 [0.0%] 125[0.0%]
30 [0.0%] 62 [0.0%] 94 [0.0%] 126[0.0%]
31 [0.0%] 63 [0.0%] 95 [0.0%] 127[0.0%]
32 [0.0%] 64 [0.0%] 96 [0.0%] 128[0.0%]
Mem[||| 27.5G/1007G] Tasks: 61, 53 thr; 2 running
Swp[0K/0K] Load average: 0.68 3.37 7.62
Uptime: 1 day, 08:19:59
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
125100 lriha 35 15 131M 3524 1528 R 9.3 0.0 1:19.05 htop -d 2
51694 root 20 0 221M 11340 3108 S 0.0 0.0 3:22.56 /opt/pbs/sbin/pbs_mom
4256 root 20 0 22612 2312 992 S 0.0 0.0 1:31.73 /usr/sbin/irqbalance --foreground
4282 root 20 0 369M 11812 7216 S 0.0 0.0 0:01.46 /usr/sbin/NetworkManager --no-daemon
1 root 20 0 187M 4652 2624 S 0.0 0.0 0:46.90 /sbin/init
5577 root 20 0 360M 45116 2232 S 0.0 0.0 0:29.88 /opt/mellanox/sharp/bin/sharpd
47271 root 20 0 1256M 17896 3124 S 0.0 0.0 0:55.68 /usr/bin/nv-fabricmanager -c /usr/share/nvidia/nvswitch/fabricm
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice - F8Nice + F9Kill F10Quit

MEMORY BOUND WORKLOAD ON CPU

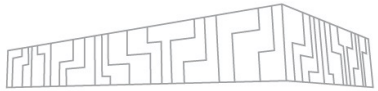


- Maximum memory bandwidth can be reached with only 16 OMP threads / CPU cores if placed correctly
- More threads improve compute performance, but reduces memory bandwidth up to 12%
- STREAM generally gives the better performance with 1 thread per CCD



GOMP_CPU_AFFINITY	0-127:32	0-127:16	0-127:8	0-127:4	0-127:2	0-127:1
# of active CPU cores	4	8	16	32	64	128
Max bandwidth [GB/s]	153,1	307,3	338,6	326,9	310,8	297,9
Efficiency	45,2%	90,8%	100,0%	96,6%	91,8%	88,0%

MEMORY BOUND WORKLOAD ON CPU



Maximum memory bandwidth

GOMP_CPU_AFFINITY	0-127:32	0-127:16	0-127:8	0-127:4	0-127:2	0-127:1
# of active CPU cores	4	8	16	32	64	128
Max bandwidth [GB/s]	153,1	307,3	338,6	326,9	310,8	297,9
Efficiency	45,2%	90,8%	100,0%	96,6%	91,8%	88,0%

AOCC compiler:

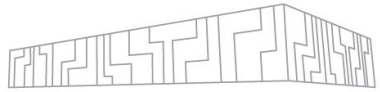
```
clang -O3 -fopenmp -mcmodel=large -DSTREAM_TYPE=double
-DSTREAM_ARRAY_SIZE=250000000 -DNTIMES=10
-mavx2 -ffp-contract=fast -fnt-store
stream.c -o stream_c
```

GOMP_CPU_AFFINITY	0-127:32	0-127:16	0-127:8	0-127:4	0-127:2	0-127:1
# of active CPU cores	4	8	16	32	64	128
Max bandwidth [GB/s]	107,2	212,7	248,2	239,9	231,8	227,2
Efficiency	43,2%	85,7%	100,0%	96,7%	93,4%	91,5%

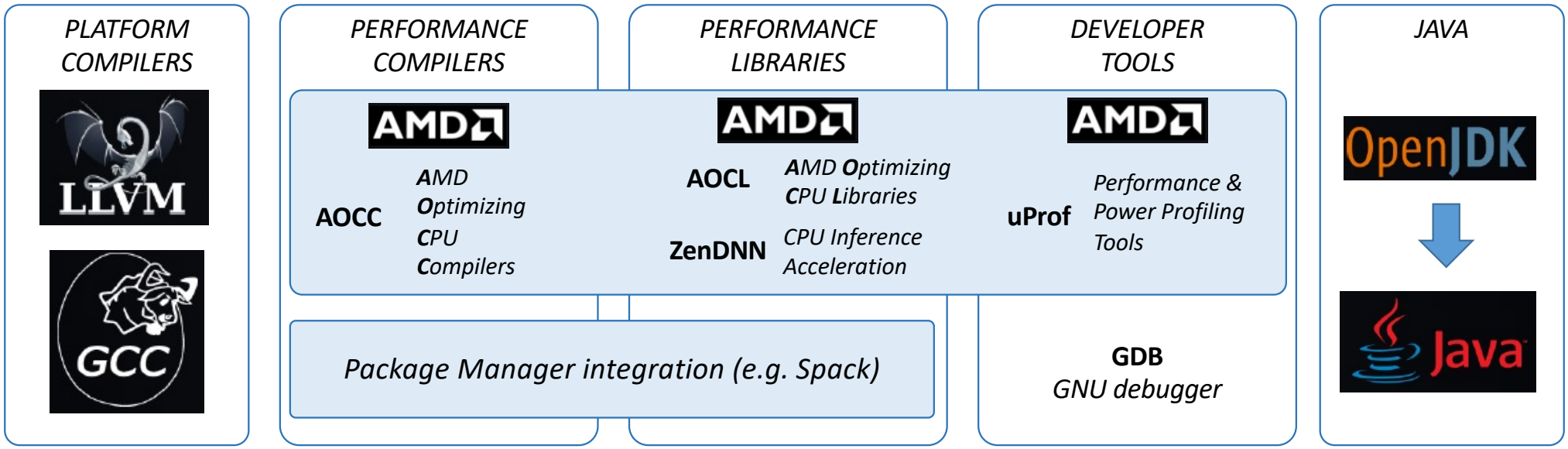
GCC compiler – settings from STREAM Makefile

```
gcc   -O2 -fopenmp -mcmodel=large -DSTREAM_TYPE=double
-DSTREAM_ARRAY_SIZE=250000000 -DNTIMES=10
stream.c -o stream_gcc
```

SOFTWARE DEVELOPMENT ENVIRONMENT

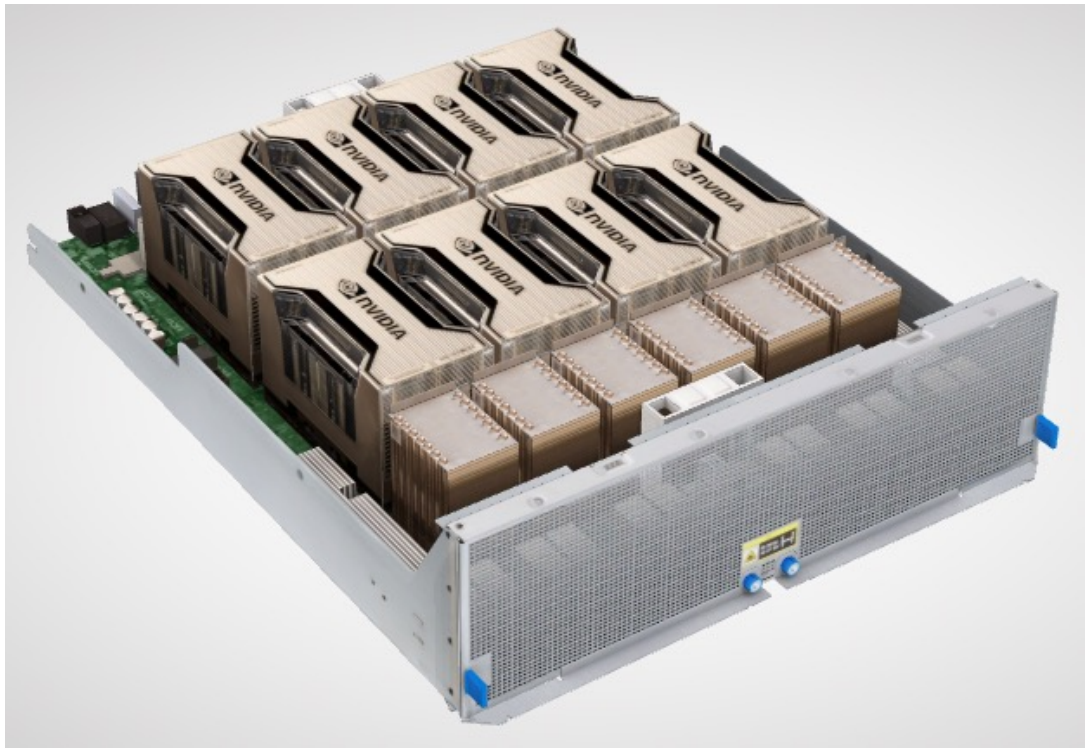


- Use AMD tools for best performance and code efficiency on EPYC CPUs
  - Compilers – focus on delivering the best out-of-the-box code generation for C, C++, Fortran, Java
  - Libraries – support common kernels for core math, solvers and FFT
  - Profiling tools – enable developers to access the full capabilities of EPYC CPUs
    - All tools are available at https://developer.amd.com/ and of course as modules

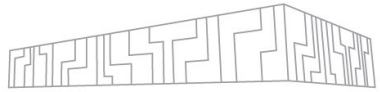


GPU ACCELERATED PARTITION

- NVIDIA HGX A100
  - 8-GPUs connected with NVSwitch



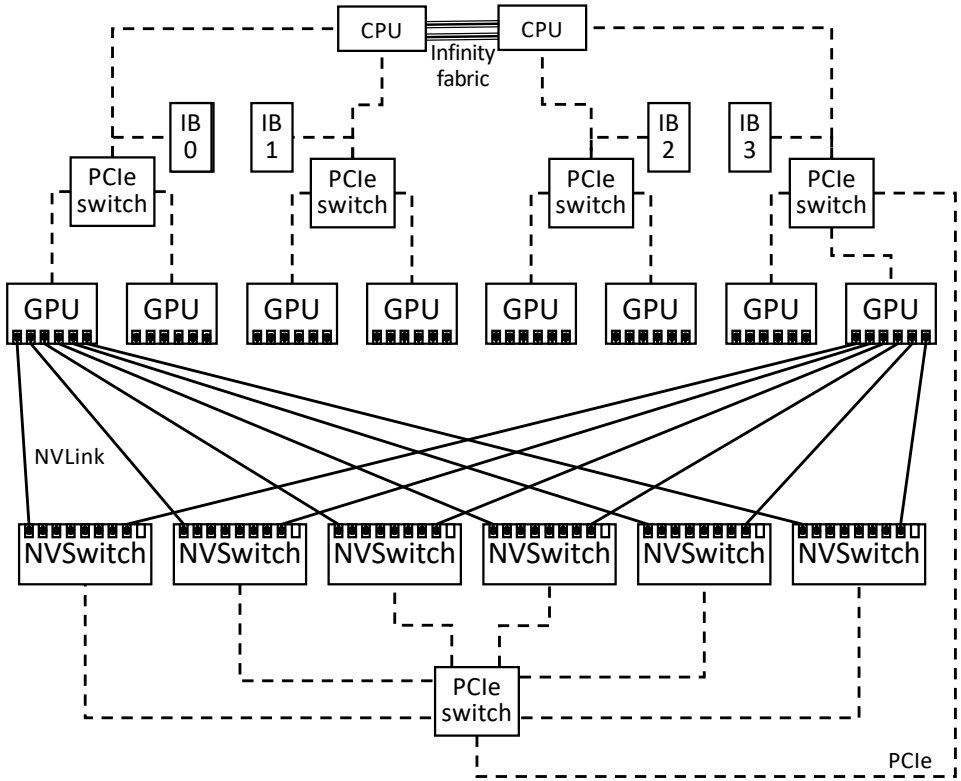
GPU ACCELERATED PARTITION



	A100 PCIe	4-GPU	8-GPU	16-GPU
GPUs	1x NVIDIA A100 PCIe	HGX A100 4-GPU	HGX A100 8-GPU	2x HGX A100 8-GPU
Form factor	PCIe	4x NVIDIA A100 SXM	8x NVIDIA A100 SXM	16x NVIDIA A100 SXM
HPC and AI compute **FP64** **TF32*/FP16*** **INT8*** * with sparsity	19.5TF 312TF*/624TF* 1.2POPS*	78TF 1.25PF*/2.5PF* 5POPS*	156TF 2.5PF*/5PF* 10POPS*	312TF 5PF*/10PF* 20POPS*
Memory	40 or 80GB per GPU	Up to 320GB	Up to 640GB	Up to 1,280GB
NVLink	Third generation	Third generation	Third generation	Third generation
NVSwitch	N/A	N/A	Second generation	Second generation
NVSwitch GPU-to-GPU bandwidth	N/A	N/A	600GB/s	600GB/s
Total aggregate bandwidth	600GB/s	2.4TB/s	4.8TB/s	9.6TB/s

https://www.nvidia.com/en-us/data-center/hgx/

GPU ACCELERATED PARTITION

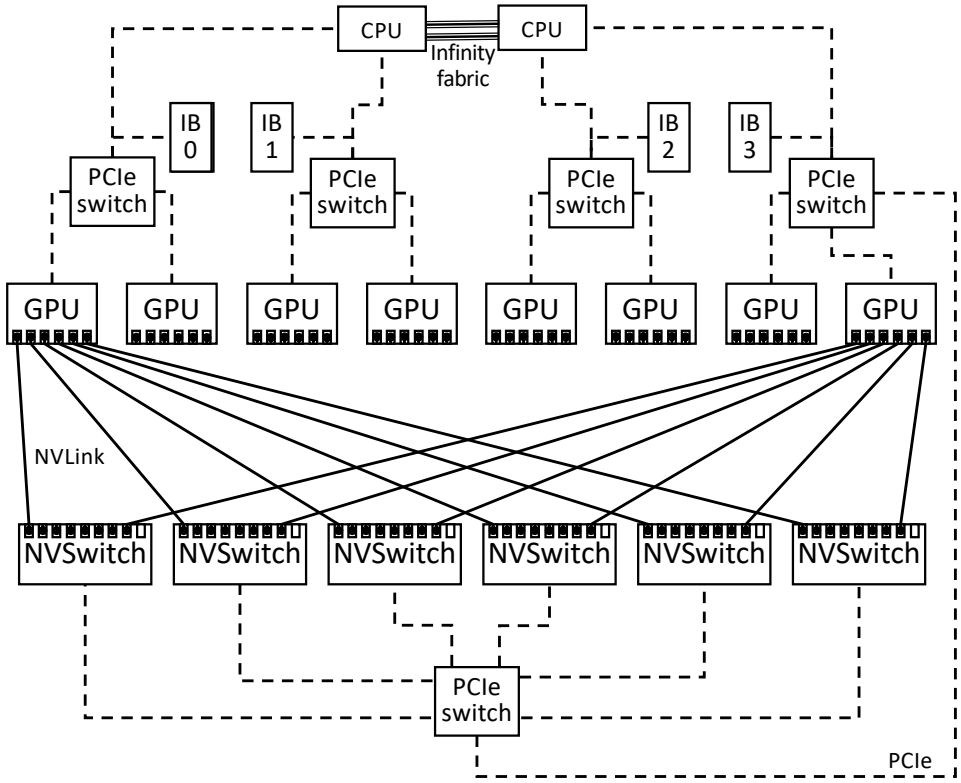


	A100 40GB SXM
FP64	9.7 TFLOPS
FP64 Tensor Core	19.5 TFLOPS
FP32	19.5 TFLOPS
Tensor Float 32 (TF32)	156 TFLOPS | 312 TFLOPS*
BFLOAT16 Tensor Core	312 TFLOPS | 624 TFLOPS*
FP16 Tensor Core	312 TFLOPS | 624 TFLOPS*
INT8 Tensor Core	624 TOPS | 1248 TOPS*
GPU Memory	40GB HBM2
GPU Memory Bandwidth	**1,555GB/s**
ıx Thermal Design Power (TDP)	400W
Multi-Instance GPU	Up to 7 MIGs @ 5GB
Form Factor	SXM
Interconnect	NVLink: **600GB/s** PCIe Gen4: 64GB/s

* With sparsity
** SXM4 GPUs via HGX A100 server boards; PCIe GPUs via NVLink Bridge for up to two GPUs

https://www.nvidia.com/en-us/data-center/a100/

NVLINK GPU INTERCONNECT IN DGX-A100



Bandwidth and latency for accessing remote memory over NVLink 3.0 for all combinations of GPUs

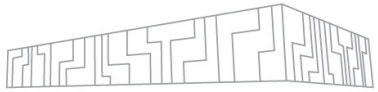
Unidir - Bandwith [GB/s]

GPU	0	1	2	3	4	5	6	7
0	1180	244	255	251	255	255	249	255
1	251	1202	256	245	256	256	252	257
2	248	256	1195	255	252	255	255	248
3	252	257	257	1198	253	255	255	249
4	244	255	256	249	1173	254	249	253
5	251	256	255	251	256	1198	255	252
6	256	251	255	255	253	254	1195	248
7	257	256	248	255	257	251	255	1206

GPU	0	1	2	3	4	5	6	7
0	4.1	11.1	11.1	10.2	10.0	9.9	9.9	10.0
1	11.1	4.2	11.1	9.6	9.7	9.6	9.8	9.7
2	11.0	11.0	4.1	10.0	10.0	9.9	9.9	9.9
3	11.1	10.1	9.6	4.2	10.0	10.0	9.9	10.0
4	11.6	10.0	9.7	9.8	4.4	9.7	9.7	9.7
5	11.7	10.1	9.8	9.8	9.7	4.4	9.7	9.7
6	11.6	11.3	10.6	9.8	9.8	9.9	4.4	9.7
7	11.7	10.3	9.8	9.8	9.7	9.7	9.8	4.4

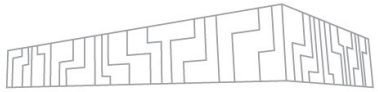
Latency [us]

GPU NODE TOPOLOGY



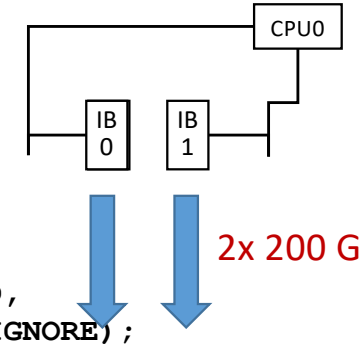
```
[lriha@acn07.karolina ~]$ nvidia-smi topo -m

        GPU0  GPU1  GPU2  GPU3  GPU4  GPU5  GPU6  GPU7  mlx5_0  mlx5_1  mlx5_2  mlx5_3  CPU Affinity  NUMA Affinity
GPU0     X    NV12  NV12  NV12  NV12  NV12  NV12  NV12  SYS     PXB     SYS     SYS     48-63         3
GPU1    NV12   X    NV12  NV12  NV12  NV12  NV12  NV12  SYS     PXB     SYS     SYS     48-63         3
GPU2    NV12  NV12   X    NV12  NV12  NV12  NV12  NV12  PXB     SYS     SYS     SYS     16-31         1
GPU3    NV12  NV12  NV12   X    NV12  NV12  NV12  NV12  PXB     SYS     SYS     SYS     16-31         1
GPU4    NV12  NV12  NV12  NV12   X    NV12  NV12  NV12  SYS     SYS     SYS     PXB     112-127       7
GPU5    NV12  NV12  NV12  NV12  NV12   X    NV12  NV12  SYS     SYS     SYS     PXB     112-127       7
GPU6    NV12  NV12  NV12  NV12  NV12  NV12   X    NV12  SYS     SYS     PXB     SYS     80-95         5
GPU7    NV12  NV12  NV12  NV12  NV12  NV12  NV12   X    SYS     SYS     PXB     SYS     80-95         5

mlx5_0  SYS   SYS   PXB   PXB   SYS   SYS   SYS   SYS    X      SYS     SYS     SYS
mlx5_1  PXB   PXB   SYS   SYS   SYS   SYS   SYS   SYS    SYS     X      SYS     SYS
mlx5_2  SYS   SYS   SYS   SYS   SYS   SYS   PXB   PXB    SYS     SYS     X      SYS
mlx5_3  SYS   SYS   SYS   SYS   PXB   PXB   SYS   SYS    SYS     SYS     SYS     X

Legend:

  X    = Self
  SYS  = Connection traversing PCIe as well as the SMP interconnect between NUMA nodes (e.g., QPI/UPI)
  NODE = Connection traversing PCIe as well as the interconnect between PCIe Host Bridges within a NUMA
node
  PHB  = Connection traversing PCIe as well as a PCIe Host Bridge (typically the CPU)
  PXB  = Connection traversing multiple PCIe bridges (without traversing the PCIe Host Bridge)
  PIX  = Connection traversing at most a single PCIe bridge
  NV#  = Connection traversing a bonded set of # NVLinks
```

RUNNING MPI ON ALL 4 IB LINKS



- 4x 200 Gb/s Infiniband HDR links should provide close to 80 GB/s
  - however, they are on different sockets – one needs 2 MPI processes to reach 80 GB/s

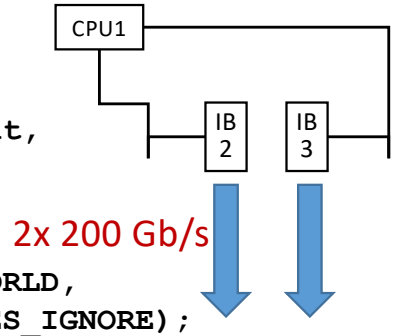
```
// node 1
if (rank == 0) {
    MPI_Recv(buffer_temp,
                buffer_count,
                MPI_BYTE,
                2,
                0,
                MPI_COMM_WORLD,
                MPI_STATUSES_IGNORE);
}
```



38 GB/s – two links

```
// node 2
if (rank == 2) {
    MPI_Send(buffer_temp,
                buffer_count,
                MPI_BYTE,
                0,
                0,
                MPI_COMM_WORLD);
}
```

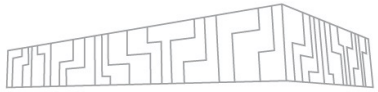
```
// node 1
if (rank == 1) {
    MPI_Recv(buffer_temp,
                buffer_count,
                MPI_BYTE,
                3,
                0,
                MPI_COMM_WORLD,
                MPI_STATUSES_IGNORE);
}
```



38 GB/s two links

```
// node 2
        if (rank == 3) {
            MPI_Send(buffer_temp,
                        buffer_count,
                        MPI_BYTE,
                        1,
                        0,
                        MPI_COMM_WORLD);
        }
```

RUNNING MPI ON ALL 4 IB LINKS



- 4x 200 Gb/s Infiniband HDR links should provide close to 80 GB/s
  - however, they are on different sockets – one needs 2 MPI processes to reach 80 GB/s

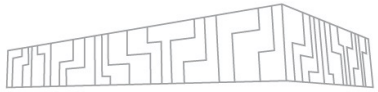
```
#qsub -q qnvidia -A PROJ_ID -l select=2:mpiprocs=2:ompthreads=64:ncpus=128 -I -l walltime=02:00:00
```

```
ml OpenMPI/4.1.1-GCC-10.3.0
mpic++ -fopenmp -o mpi_test_ompi mpi_test.cpp

mpirun -np 4 \
-bind-to core -cpu-list 16,80 --report-bindings \
-x UCX_MAX_EAGER_RAILS=2 -x UCX_MAX_RNDV_RAILS=2 \
-x UCX_NET_DEVICES=mlx5_0:1,mlx5_1:1,mlx5_2:1,mlx5_3:1 \
./mpi_test_ompi

mpirun -np 4 \
-bind-to core -cpu-list 16,80 \
./mpi_test_ompi

mpirun -np 4 \
-bind-to core -cpu-list 48,112 \
./mpi_test_ompi
```

Unified Communication X – UCX

- an open-source communication framework
- takes care of multi rail support
- loaded as module with MPI

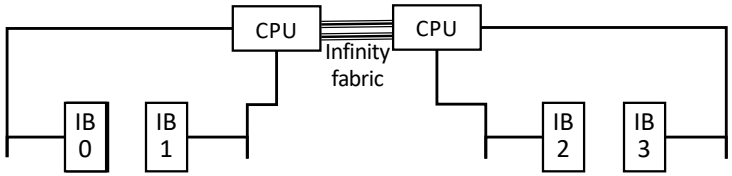
RUNNING MPI ON ALL 4 IB LINKS



- 4x 200 Gb/s Infiniband HDR links should provide close to 80 GB/s
  - however, they are on different sockets – one needs 2 MPI processes to reach 80 GB/s

```
qsub -q qnvidia -A PROJ_ID -l select=2:mpiprocs=2:ompthreads=64:ncpus=128 -I -l walltime=02:00:00
```

```
ml OpenMPI/4.1.1-GCC-10.3.0
mpic++ -fopenmp -o mpi_test_ompi mpi_test.cpp
```

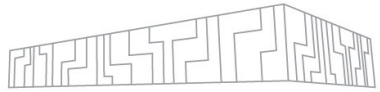



```
mpirun -np 4 \
-bind-to core -cpu-list 16,80 --report-bindings \
-x UCX_NET_DEVICES=mlx5_0:1,mlx5_1:1,mlx5_2:1,mlx5_3:1 \
-x UCX_MAX_EAGER_RAILS=2 \
-x UCX_MAX_RNDV_RAILS=2 \
./mpi_test_ompi
```

```
mpirun -np 4 \
-bind-to core -cpu-list 16,80 \
./mpi_test_ompi
```

```
[lriha@acn23.karolina mpi_test]$ ./run.sh
acn24.karolina.it4i.cz
acn23.karolina.it4i.cz
acn23.karolina.it4i.cz
acn24.karolina.it4i.cz
Start from processor acn23.karolina.it4i.cz, rank 1 out of 4 processors
Start from processor acn24.karolina.it4i.cz, rank 2 out of 4 processors
Start from processor acn24.karolina.it4i.cz, rank 3 out of 4 processors
Start from processor acn23.karolina.it4i.cz, rank 0 out of 4 processors
It: 0: rank 0, time: 0.217189, buffer_count: 1073741824, bw: 9.2 [GB/s]
It: 1: rank 0, time: 0.025723, buffer_count: 1073741824, bw: 77.8 [GB/s]
It: 2: rank 0, time: 0.025700, buffer_count: 1073741824, bw: 77.8 [GB/s]
It: 3: rank 0, time: 0.025712, buffer_count: 1073741824, bw: 77.8 [GB/s]
```

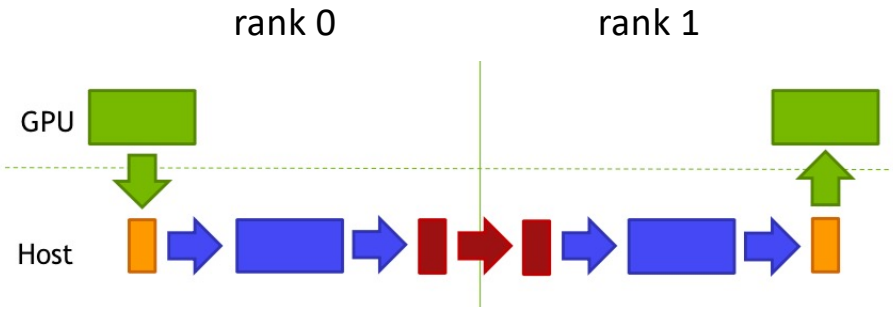
SINGLE NODE MULTI-GPU WITH OPENMP

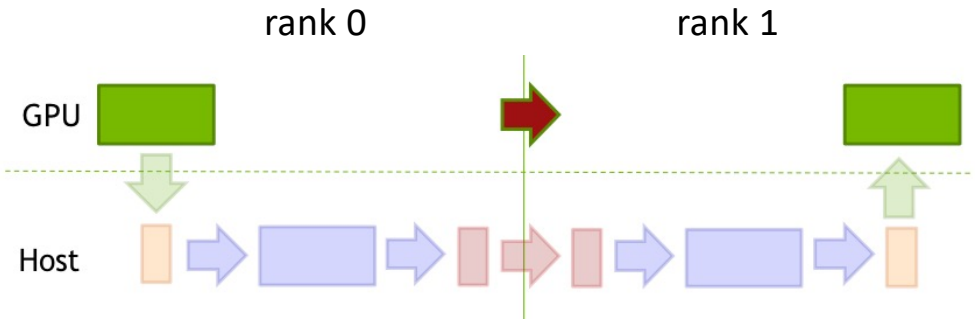


```
omp_set_num_threads( num_gpus); // create as many CPU threads as there are CUDA
devices

#pragma omp parallel
{
        unsigned int cpu_thread_id = omp_get_thread_num();

        cudaSetDevice( cpu_thread_id);

        GPUkernel<<<gpu_blocks, gpu_threads>>>( ... );
}
```

Source: https://github.com/NVIDIA/cuda-samples/tree/master/Samples/cudaOpenMP

CUDA AWARE MPI






```
cudaMemcpy(s_buf_h,s_buf_d,size,cudaMemcpyDeviceToHost);
MPI_Send(s_buf_h,size,MPI_BYTE,1,tag,MPI_COMM_WORLD);

MPI_Recv(r_buf_h,size,MPI_BYTE,0,tag,MPI_COMM_WORLD,&stat);
cudaMemcpy(r_buf_d,r_buf_h,size,cudaMemcpyHostToDevice);
```

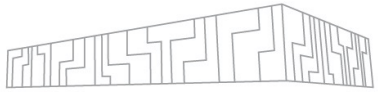
REGULAR MPI GPU TO REMOTE GPU

```
MPI_Send(s_buf_d,size,MPI_BYTE,1,tag,MPI_COMM_WORLD);

MPI_Recv(r_buf_d,size,MPI_BYTE,0,tag,MPI_COMM_WORLD,&stat);
```

MPI GPU TO REMOTE GPU

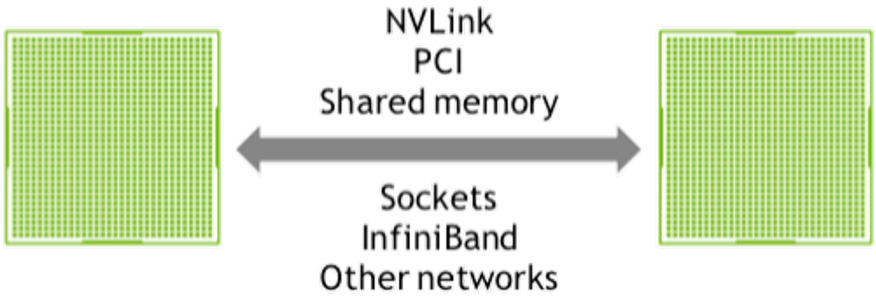
Source: Multi-GPU Programming with CUDA, GPUDirect, NCCL, NVSHMEM, and MPI; Akhil Langer, Senior Software Engineer, NVIDIA

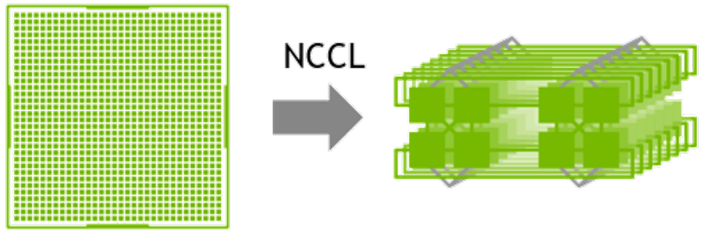
NCCL



NCCL : NVIDIA Collective Communication Library

Communication library running on GPUs, for GPU buffers.



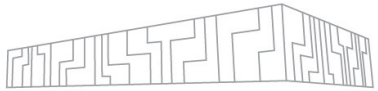


1 GPU multi-GPU, multi-node

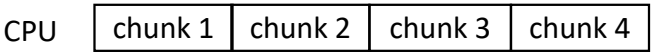
```
#include <nccl.h>
ncclComm_t comm[4];
ncclCommInitAll(comm, 4, {0, 1, 2, 3});
foreach g in (GPUs) { // or foreach thread
  cudaSetDevice(g);
  double *d_send, *d_recv;
  // allocate d_send, d_recv; fill d_send with data
  ncclAllReduce(d_send, d_recv, N, ncclDouble, ncclSum, comm[g], stream[g]);
  // consume d_recv
}
```

Source: NCCL: ACCELERATED MULTI-GPU COLLECTIVE COMMUNICATIONS, Cliff Woolley, Sr. Manager, Developer Technology Software, NVIDIA

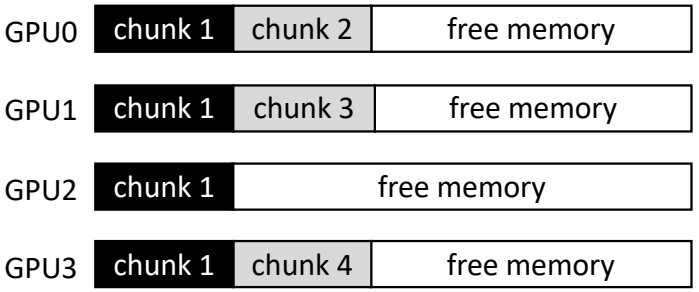
ADVANCED METHOD BASED ON MEMORY ACCESS ANALYSIS IMPLEMENTATION USING CUDA UNIFIED MEMORY



Data structure memory allocation in CPU memory



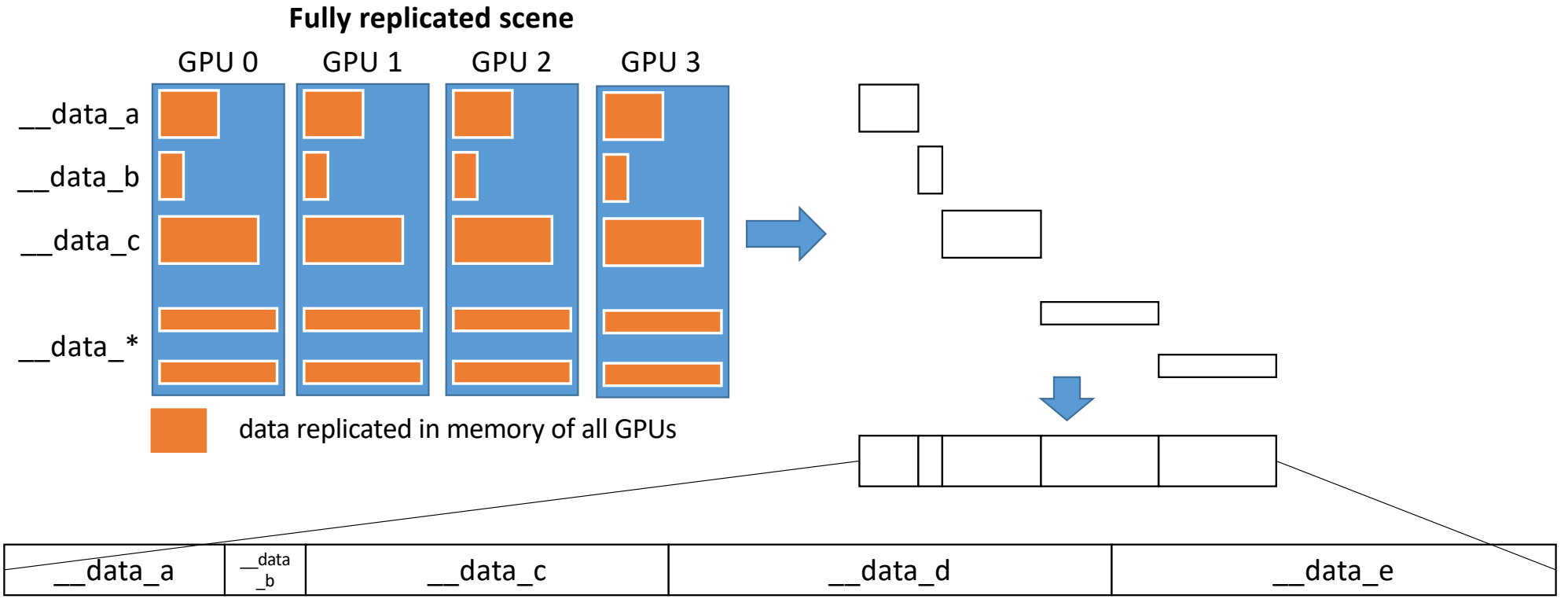
Partially replicated and distributed data structure




```
size_t size = 4 * 64 * 1024 * 1024; //size of data struct
char *data_struct = NULL;

cudaMallocManaged(&data_struct, size);

for (int gpu = 0; gpu < gpu_count; gpu++)
  cudaMemAdvise((char *)data_struct, size, cudaMemAdviseSetAccessedBy, gpu);

size_t csize = 64 * 1024 * 1024; // chunk size

//set chunk 1 to be replicated in memory of all GPUs
cudaMemAdvise(data_struct + 0*csize, csize, cudaMemAdviseSetReadMostly, 0));
for (int gpu = 0; gpu < gpu_count; gpu++)
  cudaMemPrefetchAsync(data_struct + 0*csize, csize, gpu);

//set chunk 2 to located on GPU0 only
cudaMemAdvise(data_struct + 1*csize, csize, cudaMemAdviseSetPreferredLocation, 0));
cudaMemPrefetchAsync(data_struct + 1*csize, csize, 0);

//set chunk 3 to located on GPU1 only
cudaMemAdvise(data_struct + 2*csize, csize, cudaMemAdviseSetPreferredLocation, 1));
cudaMemPrefetchAsync(data_struct + 2*csize, csize, 1);

//set chunk 4 to located on GPU3 only
cudaMemAdvise(data_struct + 3*csize, csize, cudaMemAdviseSetPreferredLocation, 3));
cudaMemPrefetchAsync(data_struct + 3*csize, csize, 3);
```

```
cudaMemAdvise ( const void* devPtr, size_t count, cudaMemoryAdvise advice, int device );

cudaMemPrefetchAsync ( const void* devPtr, size_t count, int dstDevice, cudaStream_t stream = 0 );
```

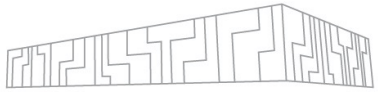
ADVANCED METHOD BASED ON MEMORY ACCESS ANALYSIS
MEMORY ACCESS PATTERN



Method work on a memory management level

- it does not differentiate what content is stored in a data structure
- works with all data structures equally
- all allocations are divided into chunks of fixed sizes

ADVANCED METHOD BASED ON MEMORY ACCESS ANALYSIS
MEMORY ACCESS PATTERN DURING PATH TRACING



__data_a	__data_b	__data_c	__data_d	__data_e

Division of all data structures into chunks

Step 1: Memory access pattern analysis

- based on 1 sample per pixel pre pass, which can be executed on CPU or on fully distributed data in GPU memories

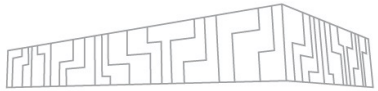
Memory access counters of all individual chunks

Step 2: Identify chunks which will be replicated

- chunks with the highest amount of memory access are marked for replication on all GPUs
- number of chunks to be replicated is based on the scene size and total amount of GPU memory

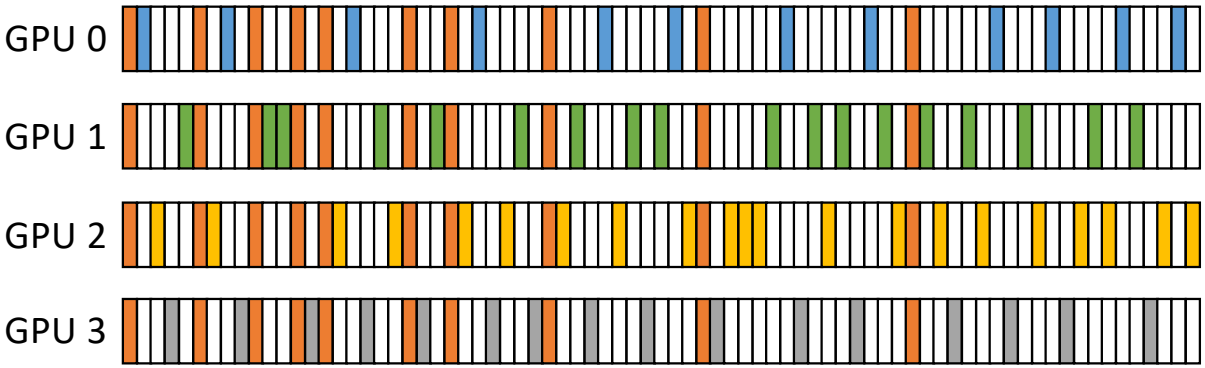
Memory access counters of all individual chunks

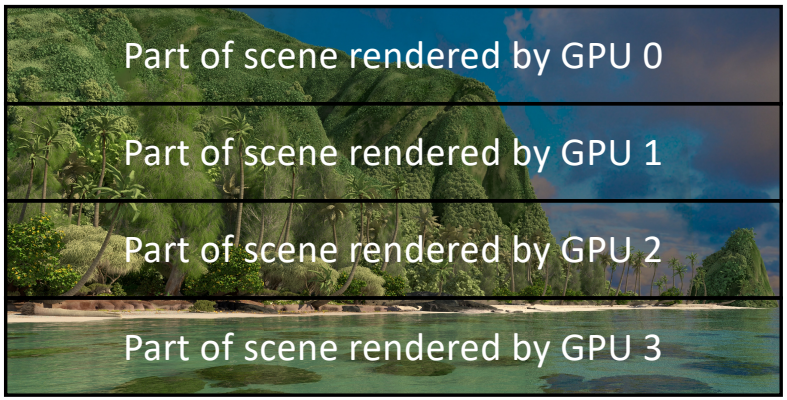
ADVANCED METHOD BASED ON MEMORY ACCESS ANALYSIS
MEMORY ACCESS PATTERN DURING PATH TRACING



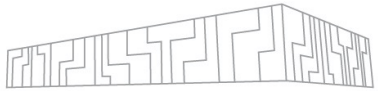
Step 3: Distribution of the non-replicated chunks

- we have to assign chunk to the GPU that has the highest number of access to it
- based on scene partitioning each part will be assigned to one GPU
- we have to record memory access counters for each part of the scene independently
  - this again can be done on both CPU or multiple GPUs with fully distributed data in a round robin fashion
- chunks with no memory accesses are distributed in a round robin fashion



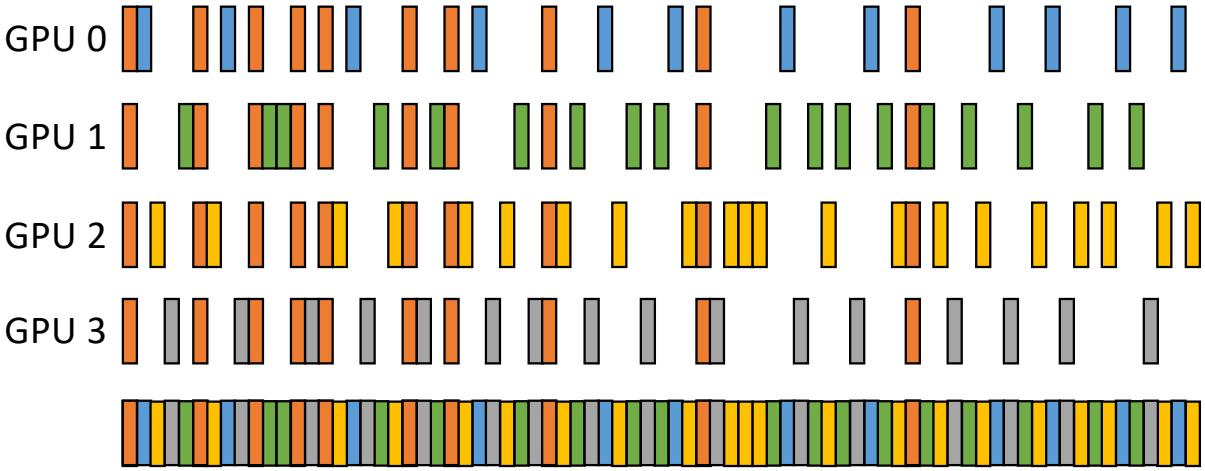


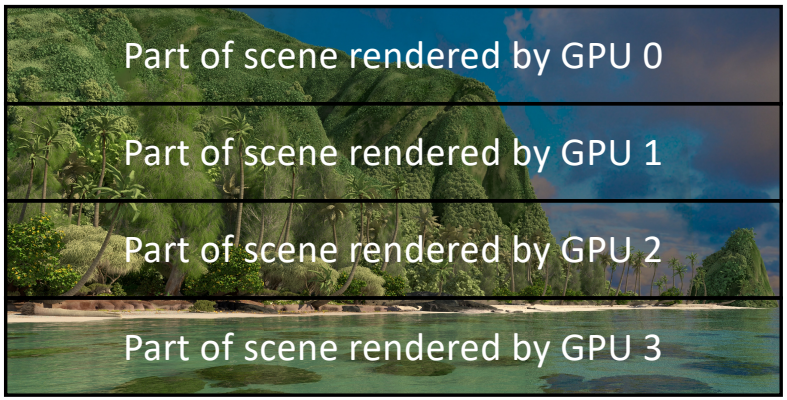
ADVANCED METHOD BASED ON MEMORY ACCESS ANALYSIS
MEMORY ACCESS PATTERN DURING PATH TRACING



Step 3: Distribution of the non-replicated chunks

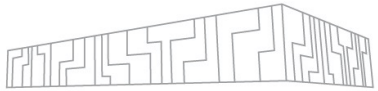
- we have to assign chunk to the GPU that has the highest number of access to it
- based on scene partitioning each part will be assigned to one GPU
- we have to record memory access counters for each part of the scene independently
  - this again can be done on both CPU or multiple GPUs with fully distributed data in a round robin fashion
- chunks with no memory accesses are distributed in a round robin fashion





Final data distribution per chunk

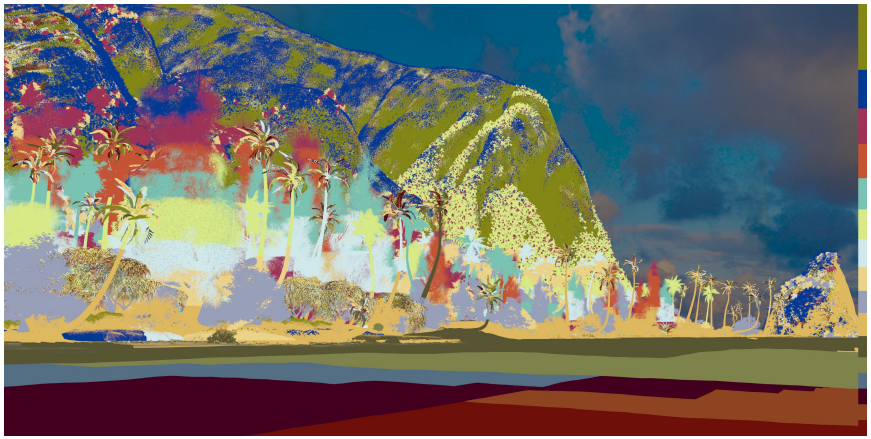
ADVANCED METHOD BASED ON MEMORY ACCESS ANALYSIS VERTICES – DATA DISTRIBUTION



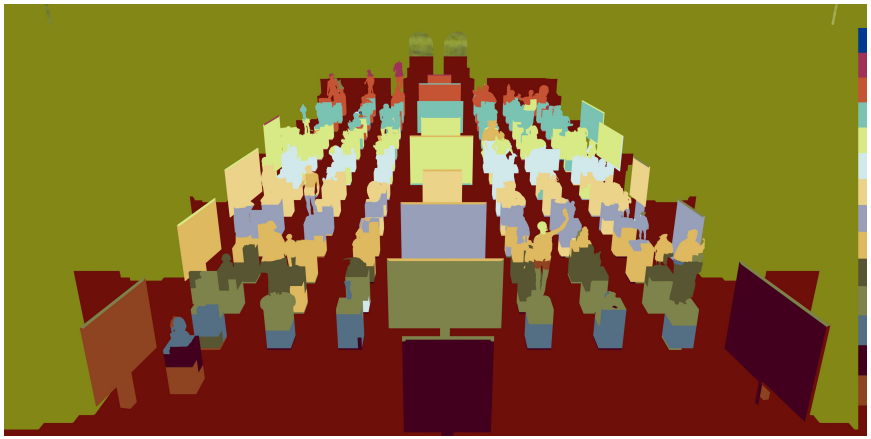
Replication 0%

Replication 10%

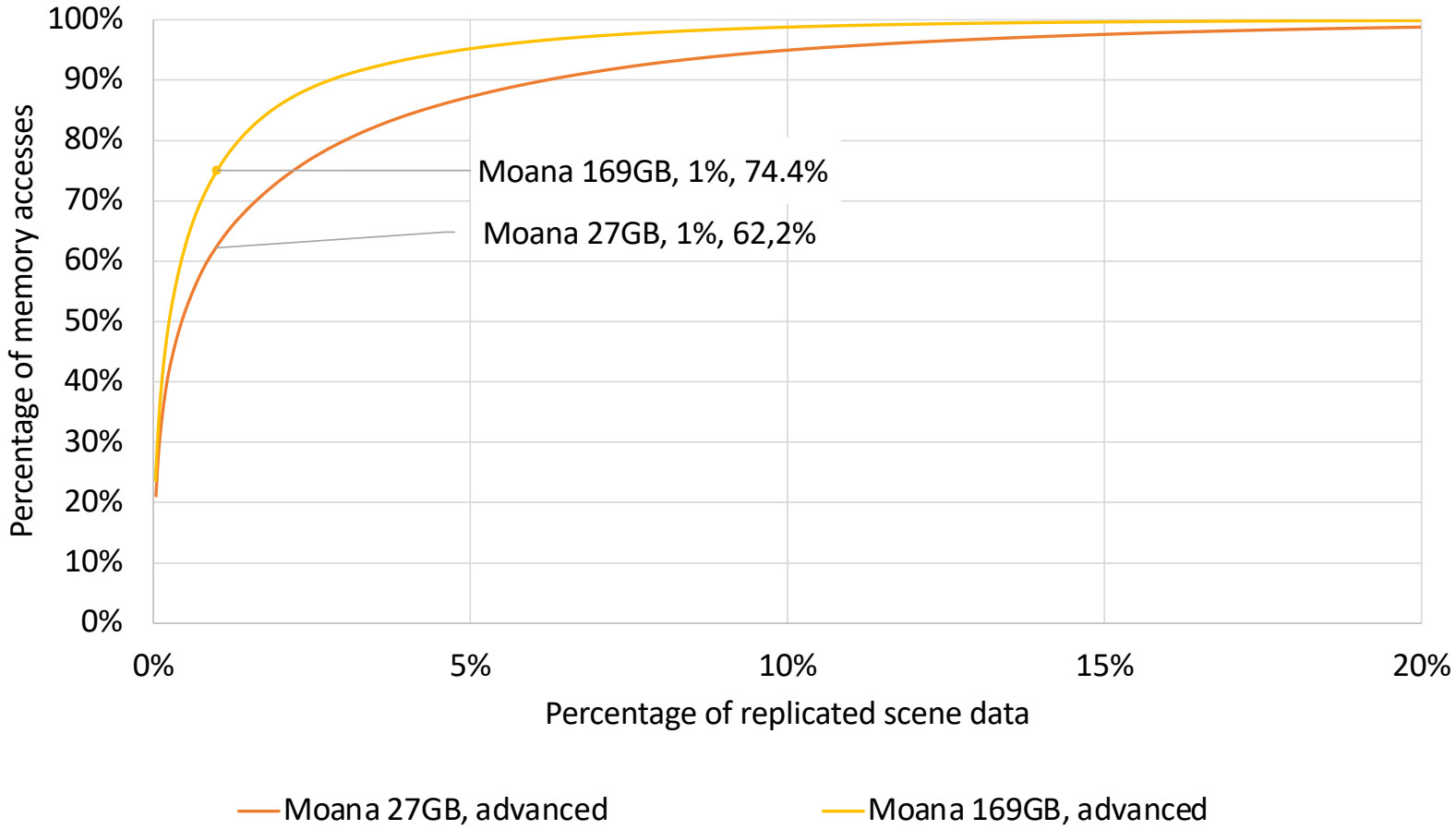
Moana Island Scene



Museum

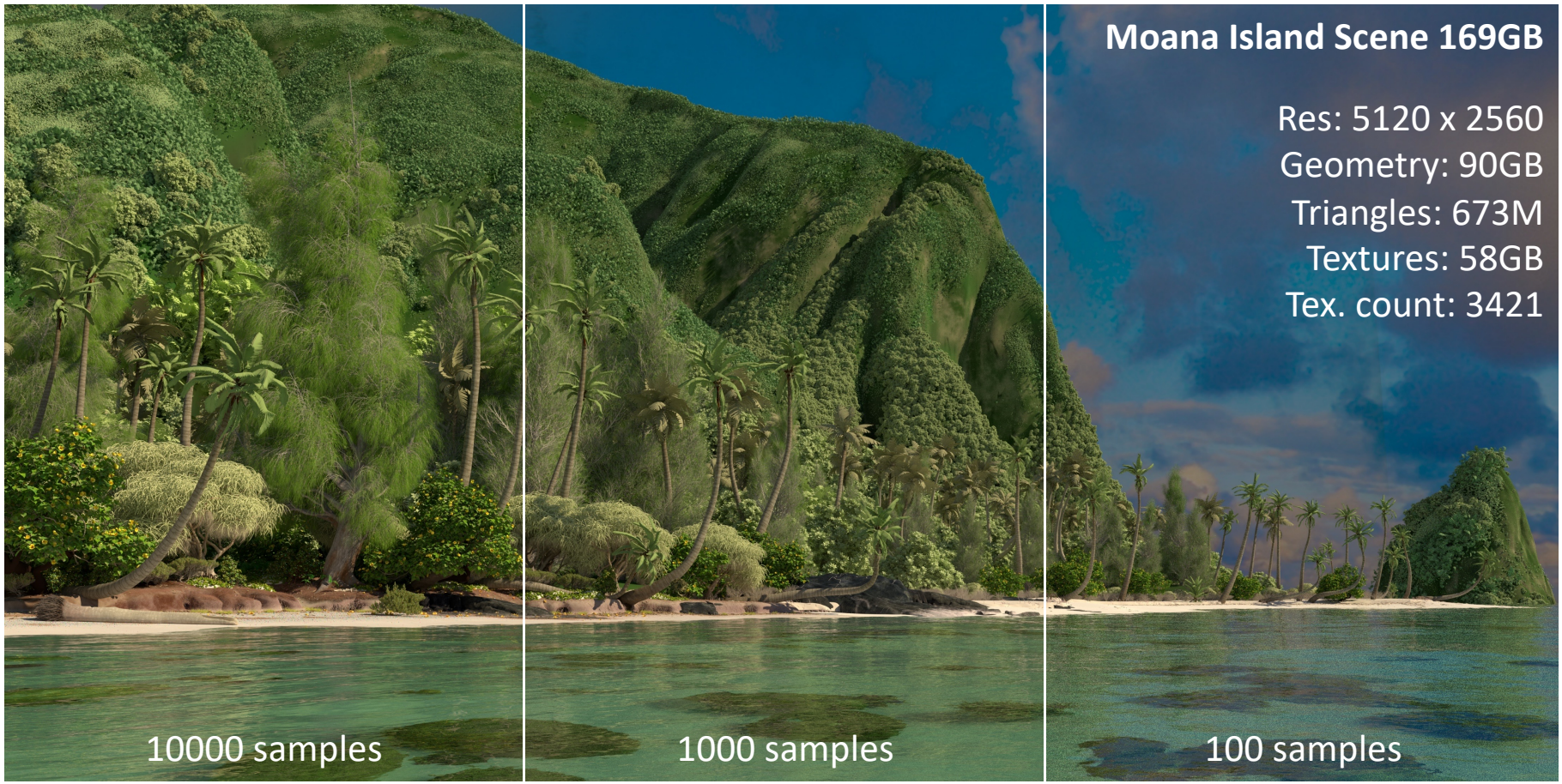


ADVANCED METHOD BASED ON MEMORY ACCESS ANALYSIS
MEMORY ACCESS PATTERN DURING PATH TRACING



Source: Jaros M., Riha L., Strakos P., Spetko M.: *GPU Accelerated Path Tracing of Massive Scenes*, ACM Transactions on Graphics (TOG), 2021, DOI: http://dx.doi.org/10.1145/3447807

RENDERING OF MASSIVE SCENES



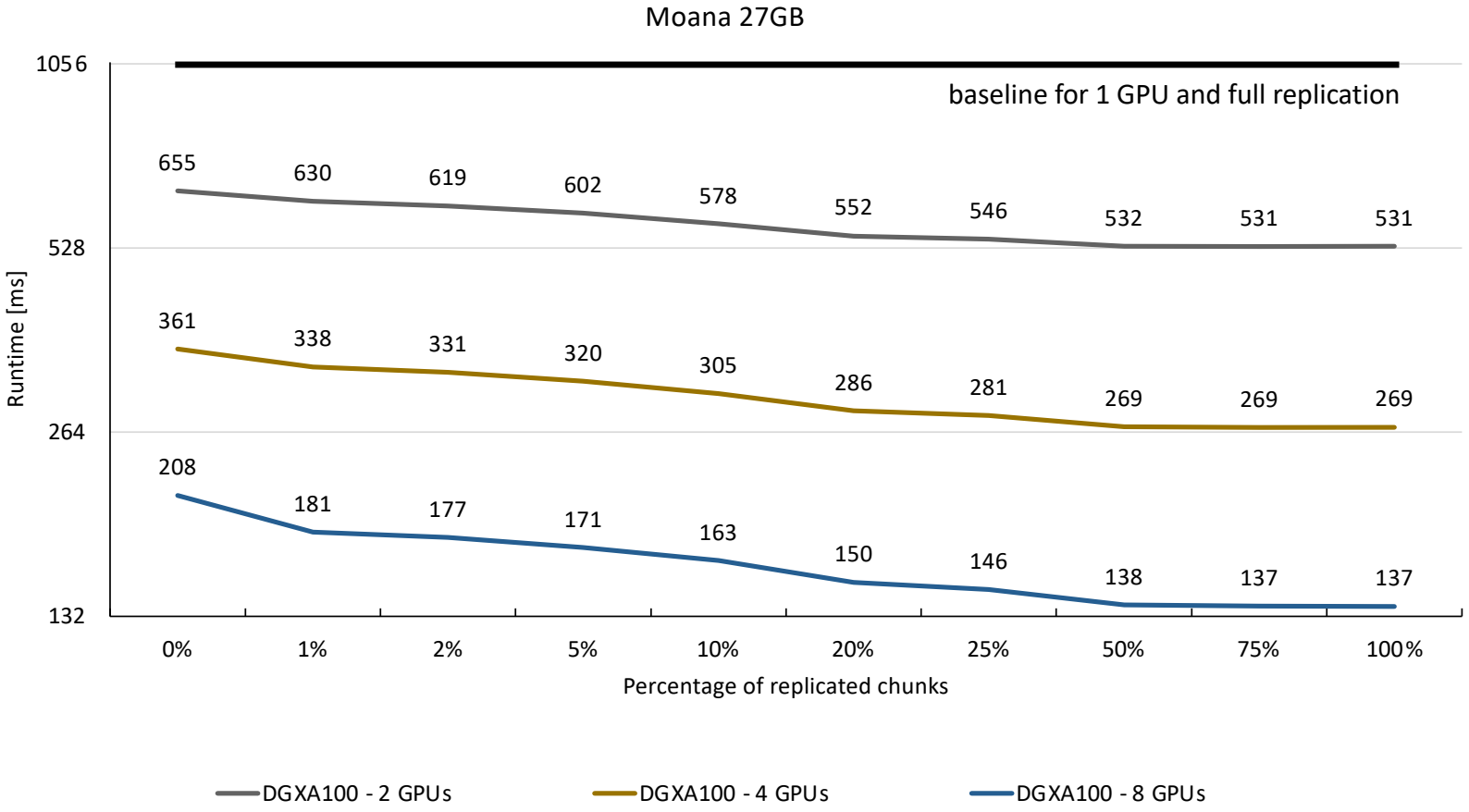
DGX-2: 30 min 52 sec
DGX-A100: 26 min 35 sec

DGX-2: 3 min 1 sec
DGX-A100: 2 min 41 sec

DGX-2: 18.7 sec
DGX-A100: 16.1 sec

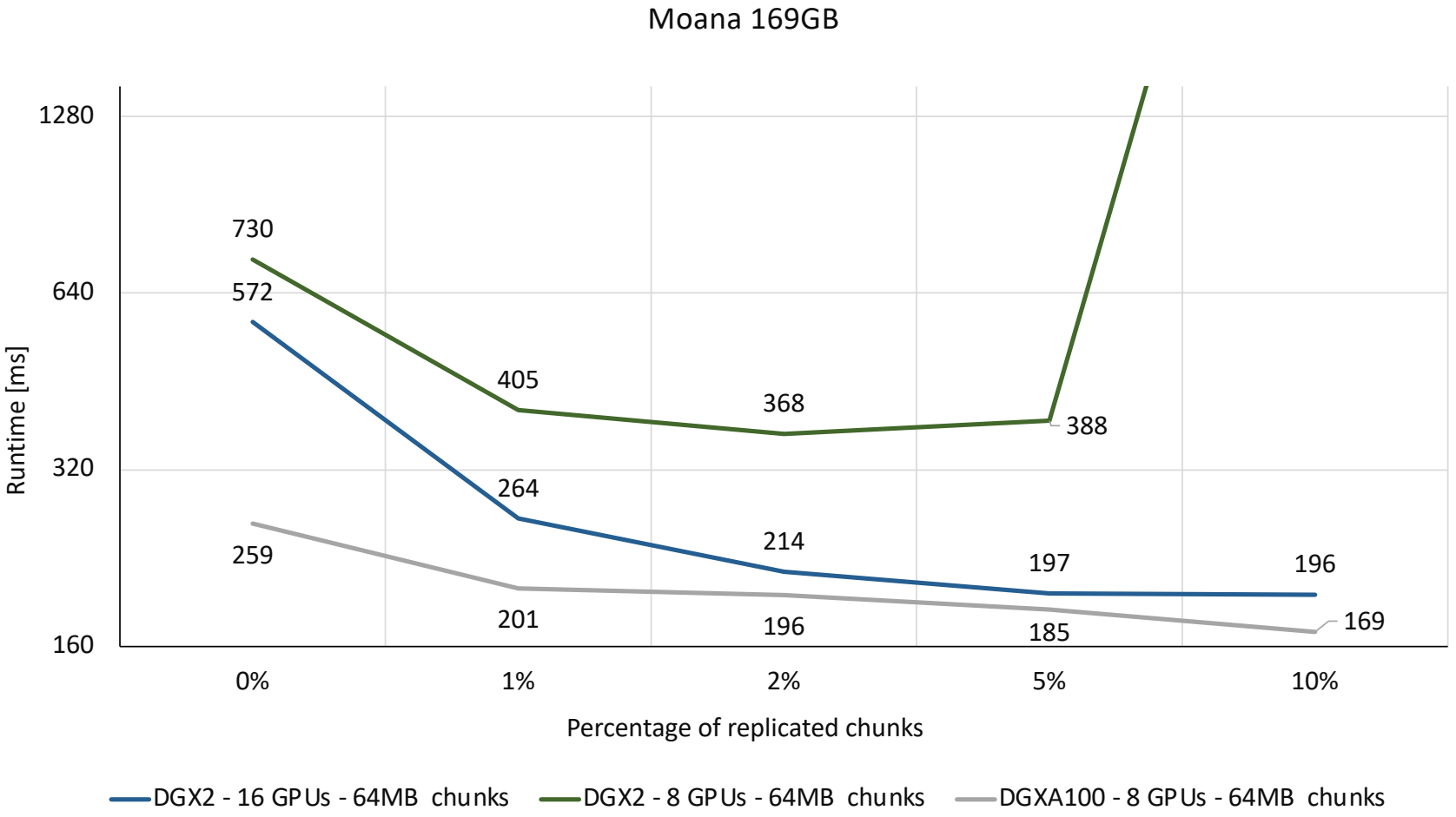
Source: Jaros M., Riha L., Strakos P., Spetko M.: *GPU Accelerated Path Tracing of Massive Scenes*, ACM Transactions on Graphics (TOG), 2021, DOI: http://dx.doi.org/10.1145/3447807

ADVANCED METHOD BASED ON MEMORY ACCESS ANALYSIS SCALABILITY ANALYSIS ON SMALL SCENE - DGX A100



Source: Jaros M., Riha L., Strakos P., Spetko M.: *GPU Accelerated Path Tracing of Massive Scenes*, ACM Transactions on Graphics (TOG), 2021, DOI: http://dx.doi.org/10.1145/3447807

RENDERING OF MASSIVE SCENES



Source: Jaros M., Riha L., Strakos P., Spetko M.: *GPU Accelerated Path Tracing of Massive Scenes*, ACM Transactions on Graphics (TOG), 2021, DOI: http://dx.doi.org/10.1145/3447807

Lubomir Riha
lubomir.riha@vsb.cz

IT4Innovations National Supercomputing Center
VSB – Technical University of Ostrava
Studentská 6231/1B
708 00 Ostrava-Poruba, Czech Republic
www.it4i.cz

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education

MINISTRY OF EDUCATION, YOUTH AND SPORTS