PAPI BOF

Performance **A**pplication **P**rogramming **I**nterface

Introduction
Methodology
GPU Tuning

Presenter
Dr. Anthony Castaldo

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

PAPI

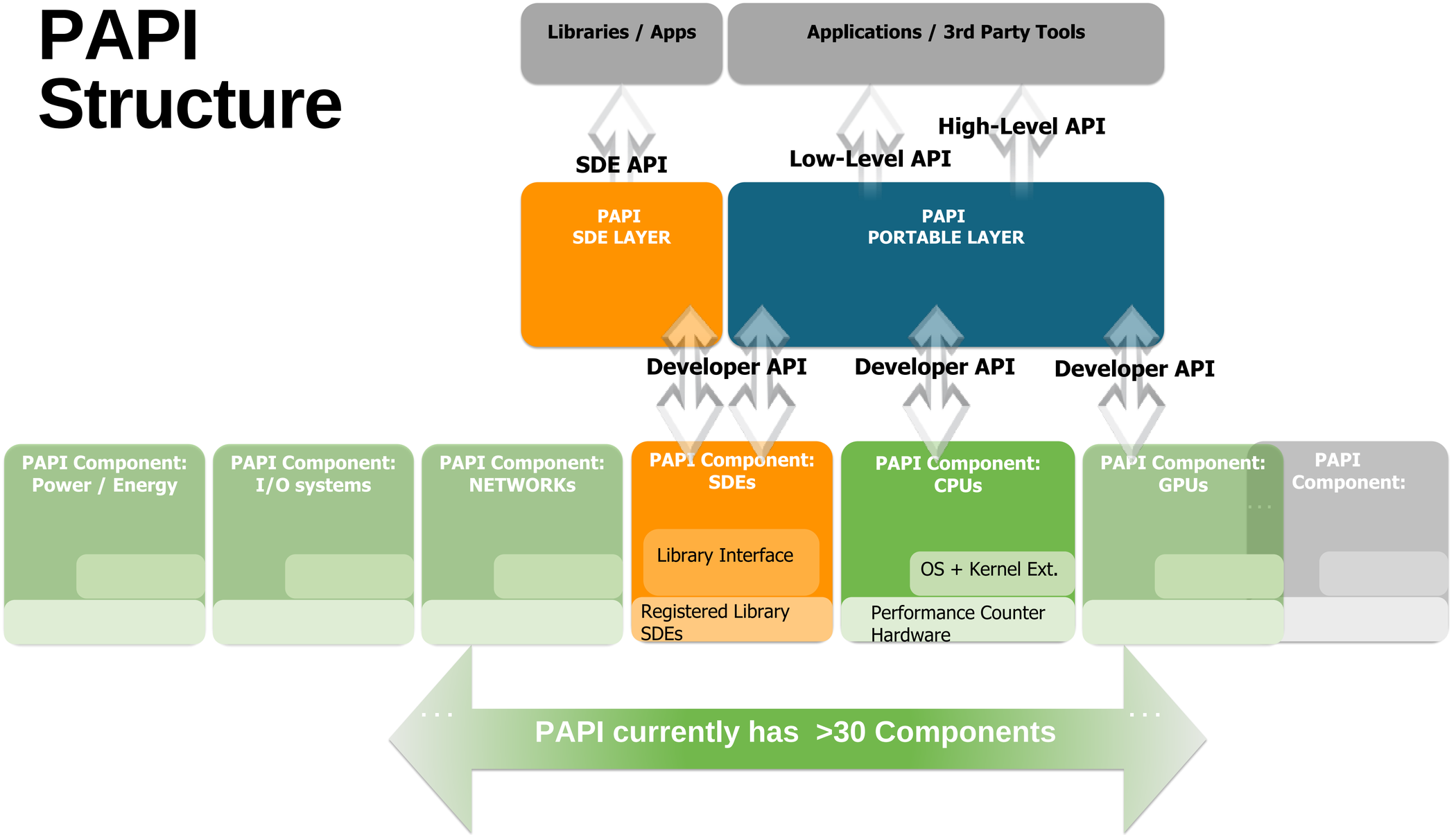
SUPPORTED ARCHITECTURES:

- AMD **up to Zen1, Zen2**, **power for Fam17h (Zen3 in progress)**
- AMD **GPUs MI50, MI60, MI100**, **power, temperature, fan**
- ARM Cortex A8, A9, A15, ARM64, **ARM uncore-support**
- CRAY: Gemini and Aries interconnects, power/energy
- IBM Blue Gene Series, Q: 5D-Torus, I/O system, EMON power/energy
- IBM Power Series, **PCP for POWER9-nest, power monitoring & capping on POWER9**
- Intel Sandy|Ivy Bridge, Haswell, Broadwell, Skylake, **Kaby-**, **Cascade-, Ice-lake,** KNC, KNL, KNM
- Intel RAPL (power/energy), **power capping**
- **Intel GPUs Gen9**
- InfiniBand
- Lustre FS
- NVIDIA Tesla, Kepler, Maxwell, Pascal, Volta, **Turing, Ampere**: support for multiple GPUs
- **NVIDIA: support for NVLink**
- NVIDIA NVML (power/energy); **power capping**
- Virtual Environments: VMware, KVM
- **Software-defined Event (SDE) Support**

Advantages of PAPI

- PAPI is a library that provides a **consistent interface** (and methodology) for hardware performance counters, found across the system: i. e., CPUs, GPUs, on-/off-chip Memory, Interconnects, I/O, File System, Energy/Power, etc.
- It is effectively a universal translator that handles 30+ APIs with one simple API.
- PAPI is supported – The team keeps up with changes to the APIs, new APIs, and we communicate with vendors to address issues with their API, or if possible to circumvent such issues. PAPI supports our users, we are active in our group and answer questions and help solve problems.
- PAPI strives to minimize overhead, so SW engineers can see, in near real time, the relationship between SW performance and HW events across the entire compute system.
- PAPI is a live open-source project, still being developed to support new hardware and devices as they are introduced. The repository is updated frequently, and we are developing new interfaces to analysis and graphing packages, and with SDE, new interfaces to libraries so they can report on their performance.

PAPI Structure



INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Installing the PAPI Library

- Download PAPI. ~$ git clone https://bitbucket.org/icl/papi.git
- Configure PAPI. ~/papi/src$ ./configure --with-components="…"
  "…" is a list of components desired, separated by spaces. See ~/papi/src/components, each subdirectory has a README.md explain the component. Many components require an env variable PAPI_[comp_name]_ROOT exported before this ./configure.
- Build PAPI. ~/papi/src$ make
- Verify PAPI. ~/papi/src$ utils/papi_component_avail
- Identify Events. ~/papi/src$ utils/papi_native_avail >native_avail.txt
- Instrument your code to use PAPI, and your Makefile to identify include file directories and library directories.
  -L~/papi/src -ldl -lpapi

PAPI Simple Methodology (1 of 2)

- Include “papi.h”. // in papi/src.
- Initialize PAPI. // Only once in the program.

```
PAPI_library_init( PAPI_VER_CURRENT );
```

- Create: one or more event sets (one per component).

```
PAPI_create_eventset(&EventSet);
```

- Add events by name to the event sets.

```
char eventname[]="rocm:::SQ_INSTS_GDS:device=0";
 PAPI_add_named_event(EventSet, eventname);
```

- Start the event sets. For kernels, keep PAPI_start() as close to the kernel start as possible. For cumulative counters, PAPI_start() will zero them.

```
PAPI_start(EventSet);
```

- Read each event and process values.

```
long long values[1]={0};
PAPI_read(EventSet, values);
```

- Stop the event set (with an optional final read); use NULL for 2nd arg to skip read.

```
PAPI_stop( EventSet, values);
```

PAPI Simple Methodology (2 of 2)

Optionally, for a stopped event set, it can be edited before restarting, to perhaps read different counters for a new stage of the application.

- Add more events by name.
  PAPI_add_named_event(EventSet, new_eventname);
- Delete events by name.
  PAPI_remove_named_event(EventSet, eventname);
- Delete all events in an event set.
  PAPI_cleanup_eventset(EventSet);

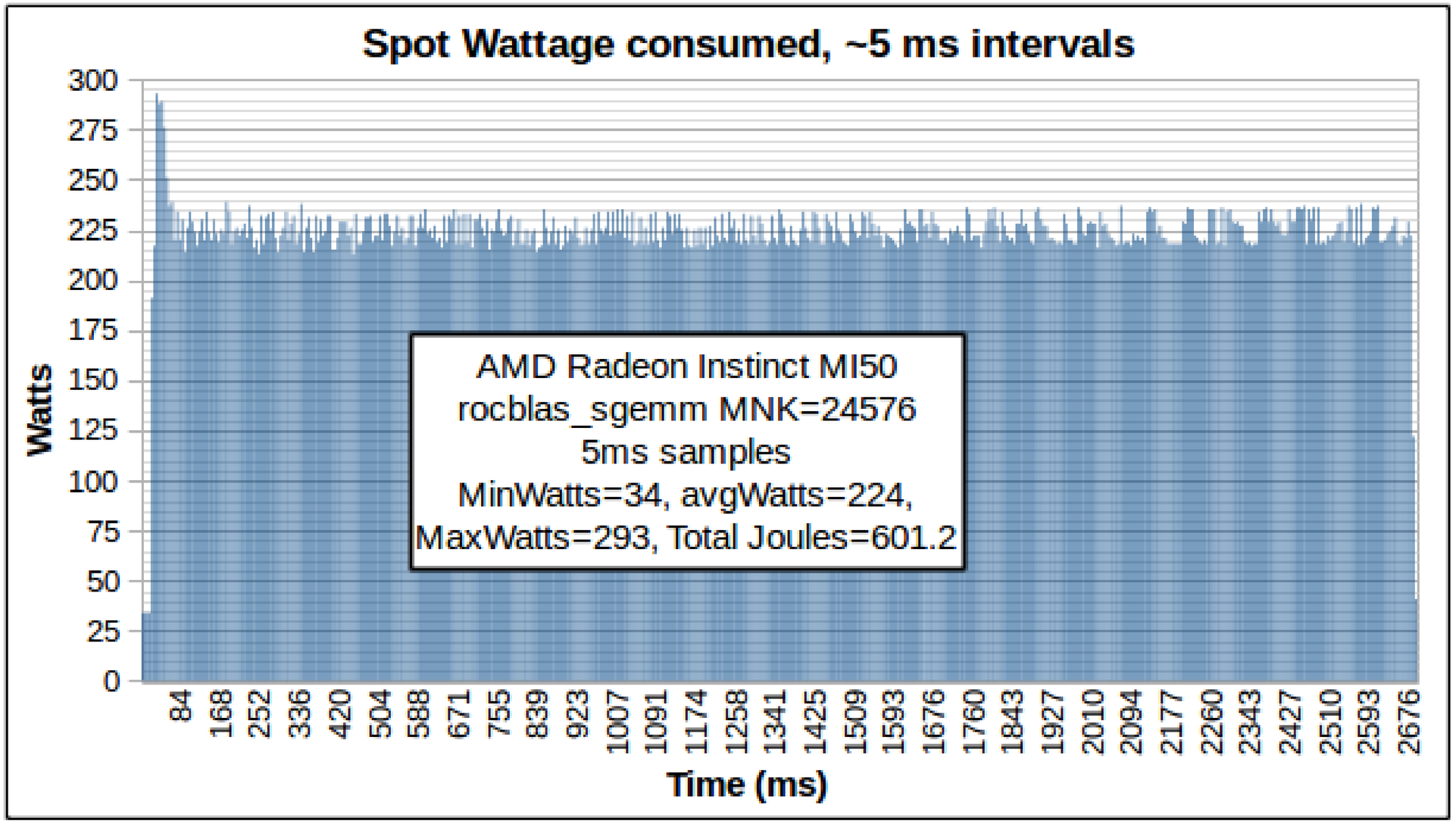
After edits, PAPI_start() the event set again, and PAPI_read() as often as desired.
When done with an event set, it can be destroyed:

```
PAPI_destroy_eventset(&EventSet);  // Notice we pass the address.
```

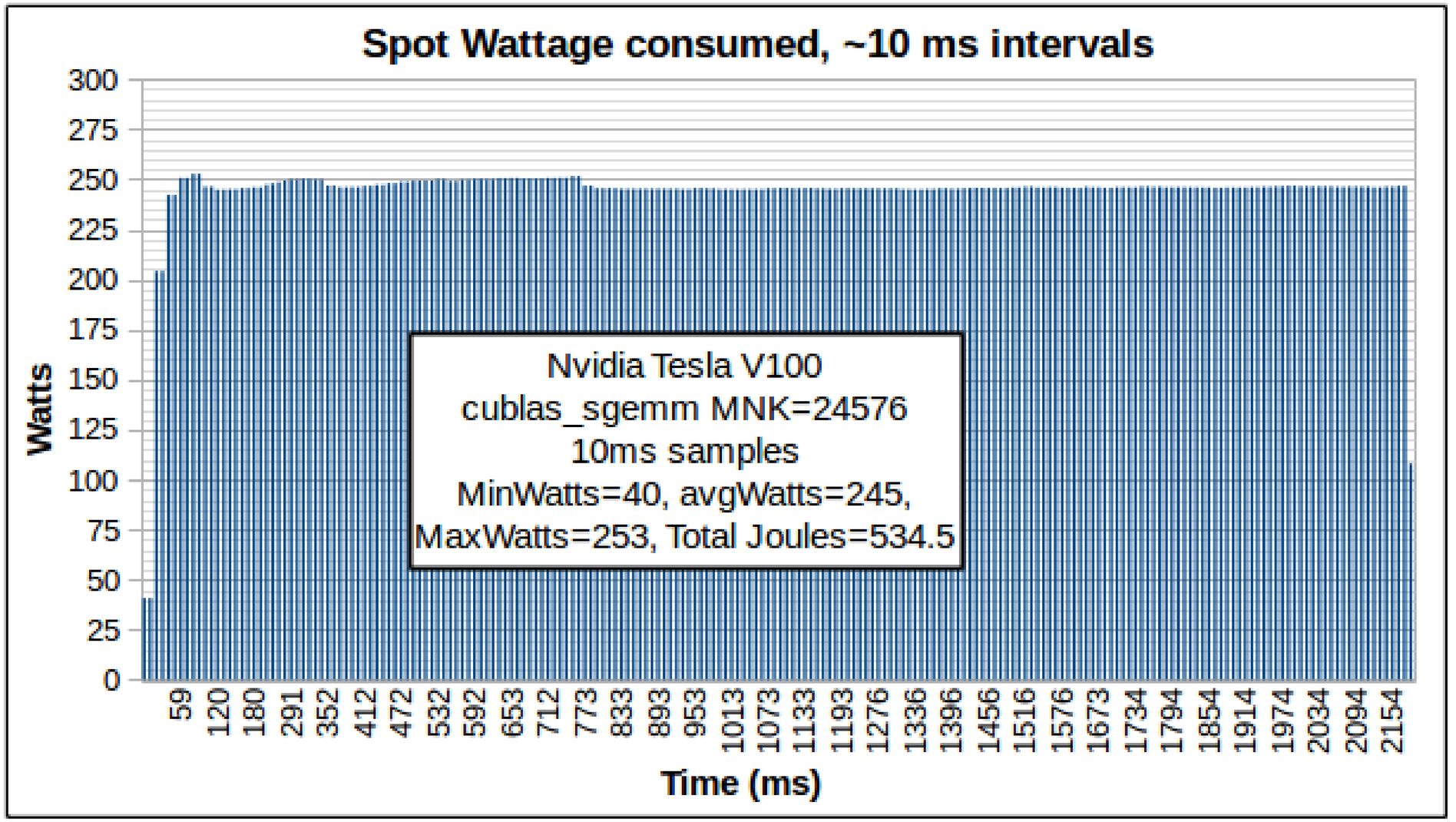
And finally, when done with PAPI, Shut it down, to clean up all memory allocations:

```
PAPI_shutdown(); // Returns no value.
```

Output – rocm_smi



Output – NVML



Finding Optimal Power Settings

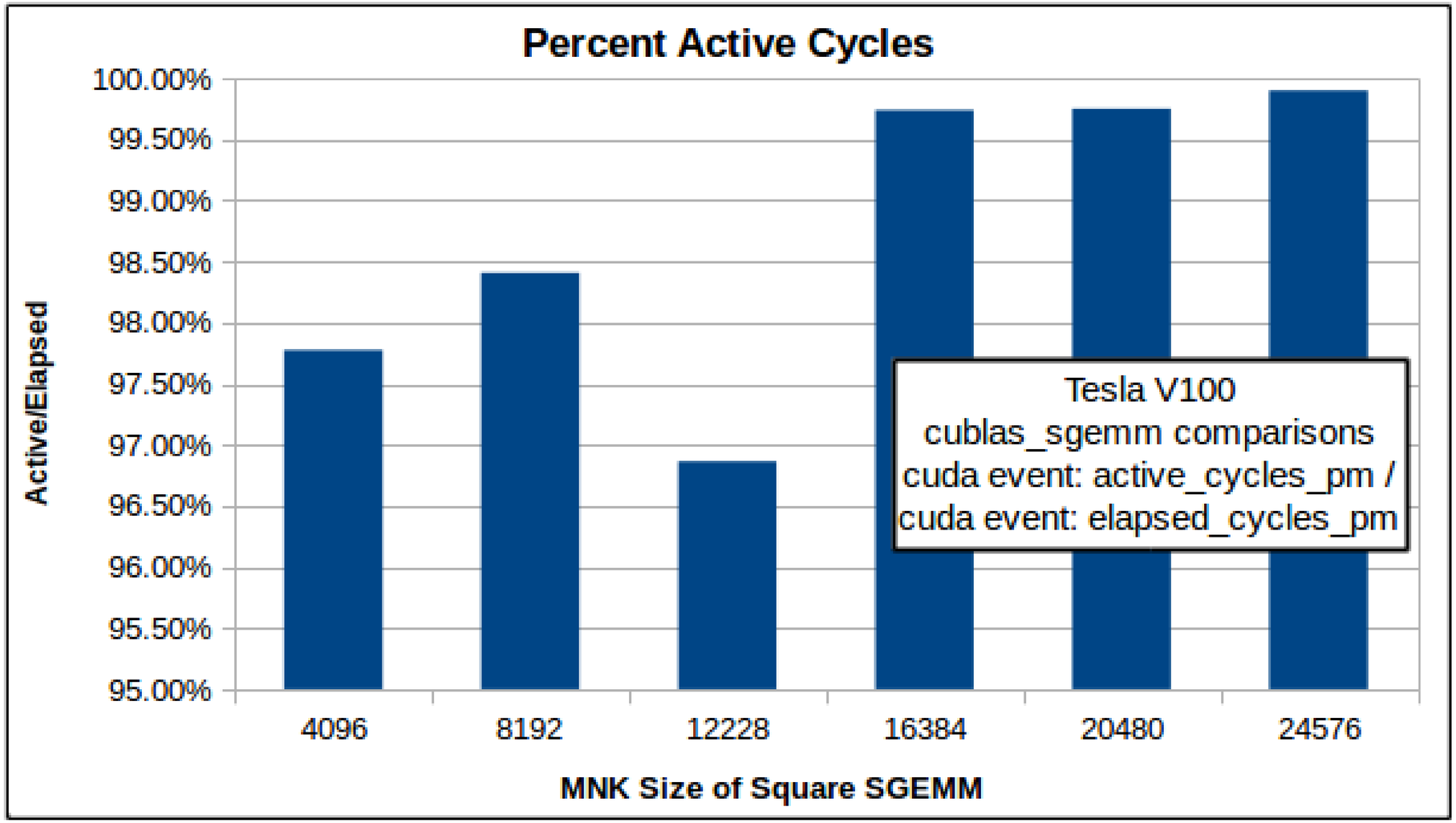
- Powercapping is potentially an energy saver if runtime effects are tolerable. In some apps, reduced power has very little runtime effect.
- Runtime increases are not always proportional to the reduced power. A 20% power reduction may barely affect the runtime of one app, or may increase the runtime of another app by 30%.
- The minimum **Total** Joules expended is NOT always at the lowest power. In dense GEMM experiments, I often find the minimum in the upper half of the range, not in the low end.
- You should do a survey, running your kernel across the range of power caps, at perhaps 10 or 20 settings, and computing the total Joules expended at each cap.

Output – Cuda

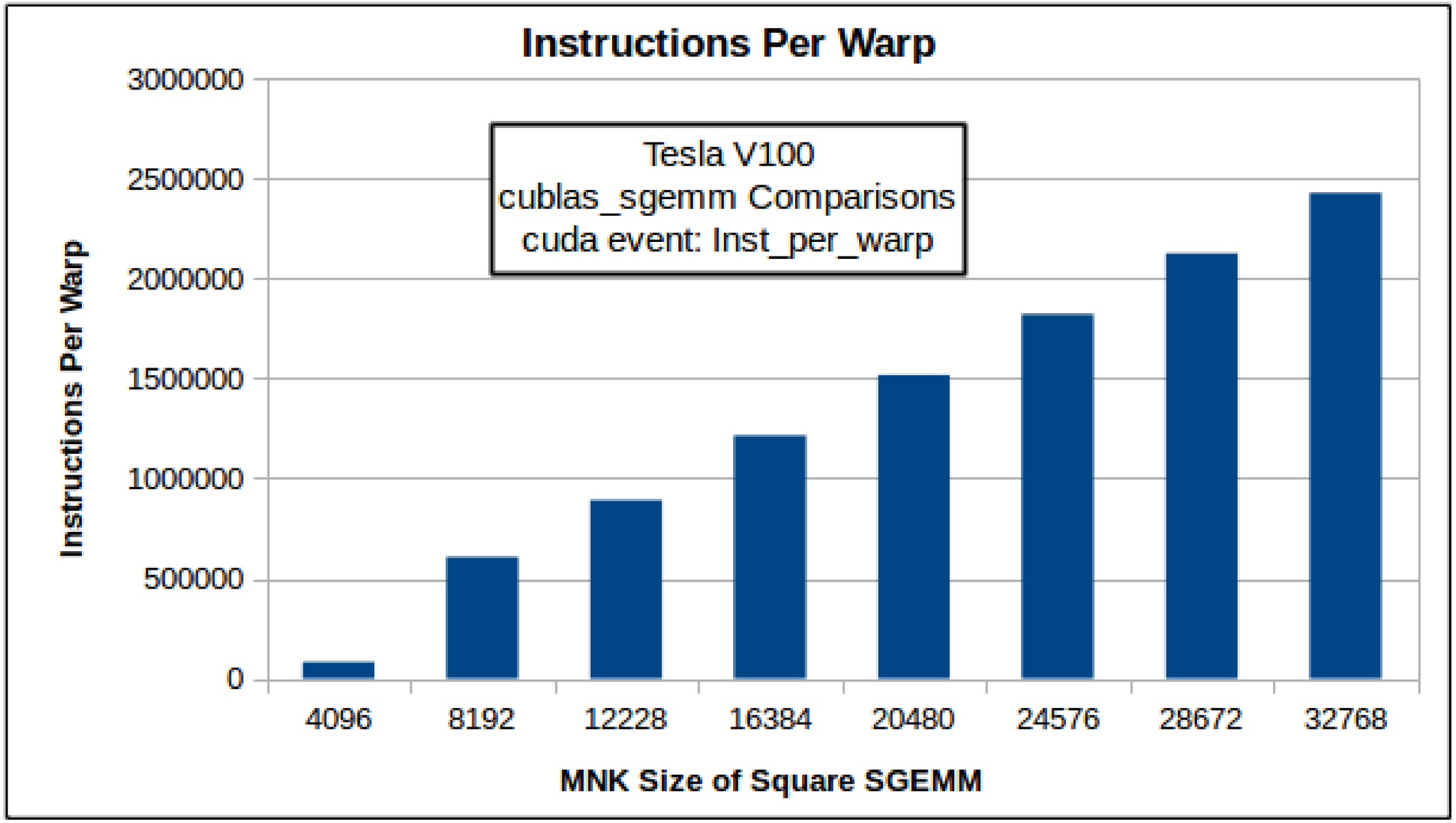
These are final samples, over the course of each square sgemm run, in this case the cycles active, and the elapsed cycles. We can use these to plot the ratio of active to elapsed.

sgemm MNK	active_cycles_pm	elapsed_cycles_pm
4096	1.112E+09	1.138E+09
8192	8.669E+09	8.809E+09
12228	29.127E+09	3.007E+10
16384	69.328E+09	6.951E+10
20480	135.403E+09	1.357E+11
24576	233.974E+09	2.342E+11

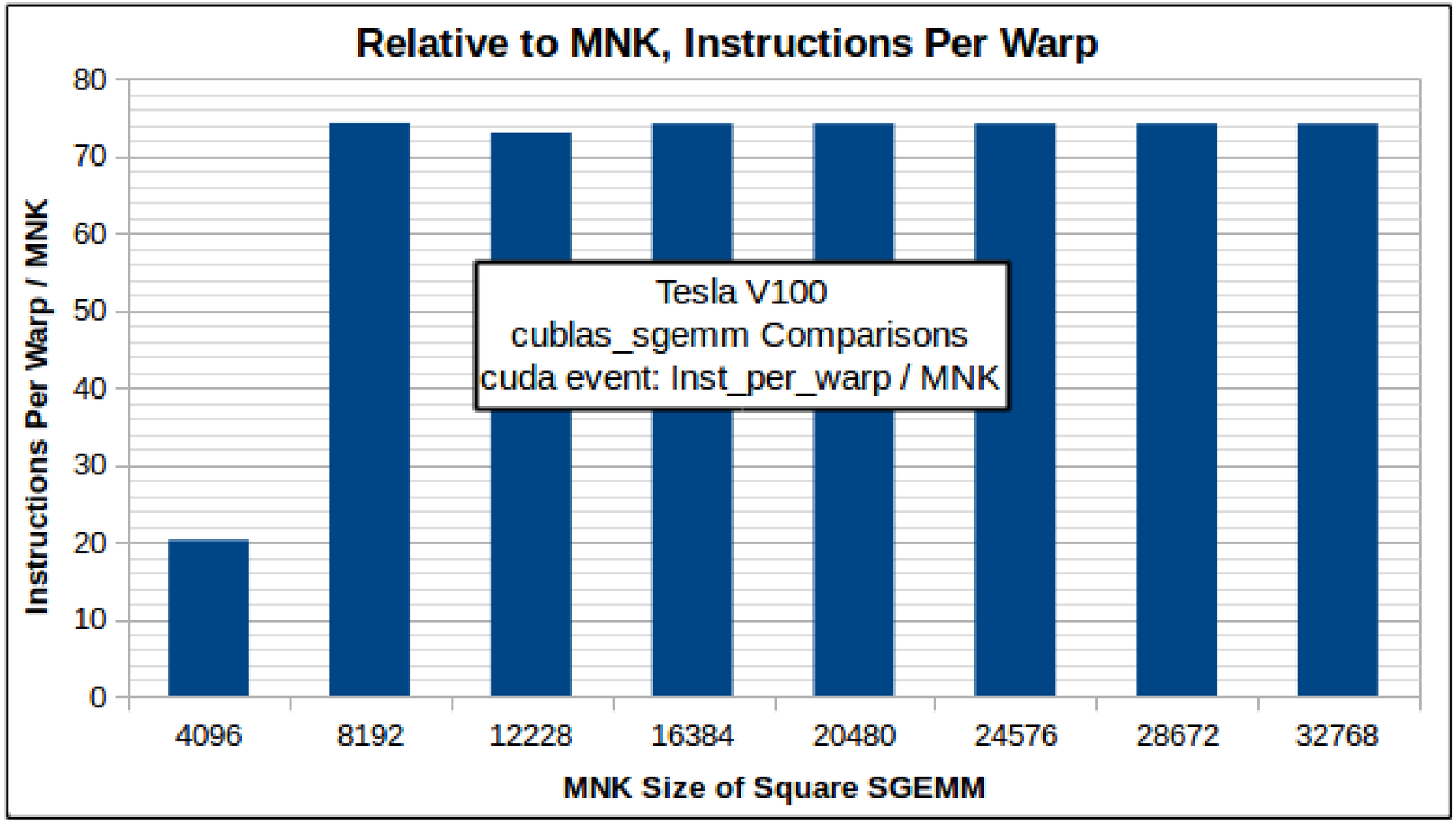
Output – Cuda



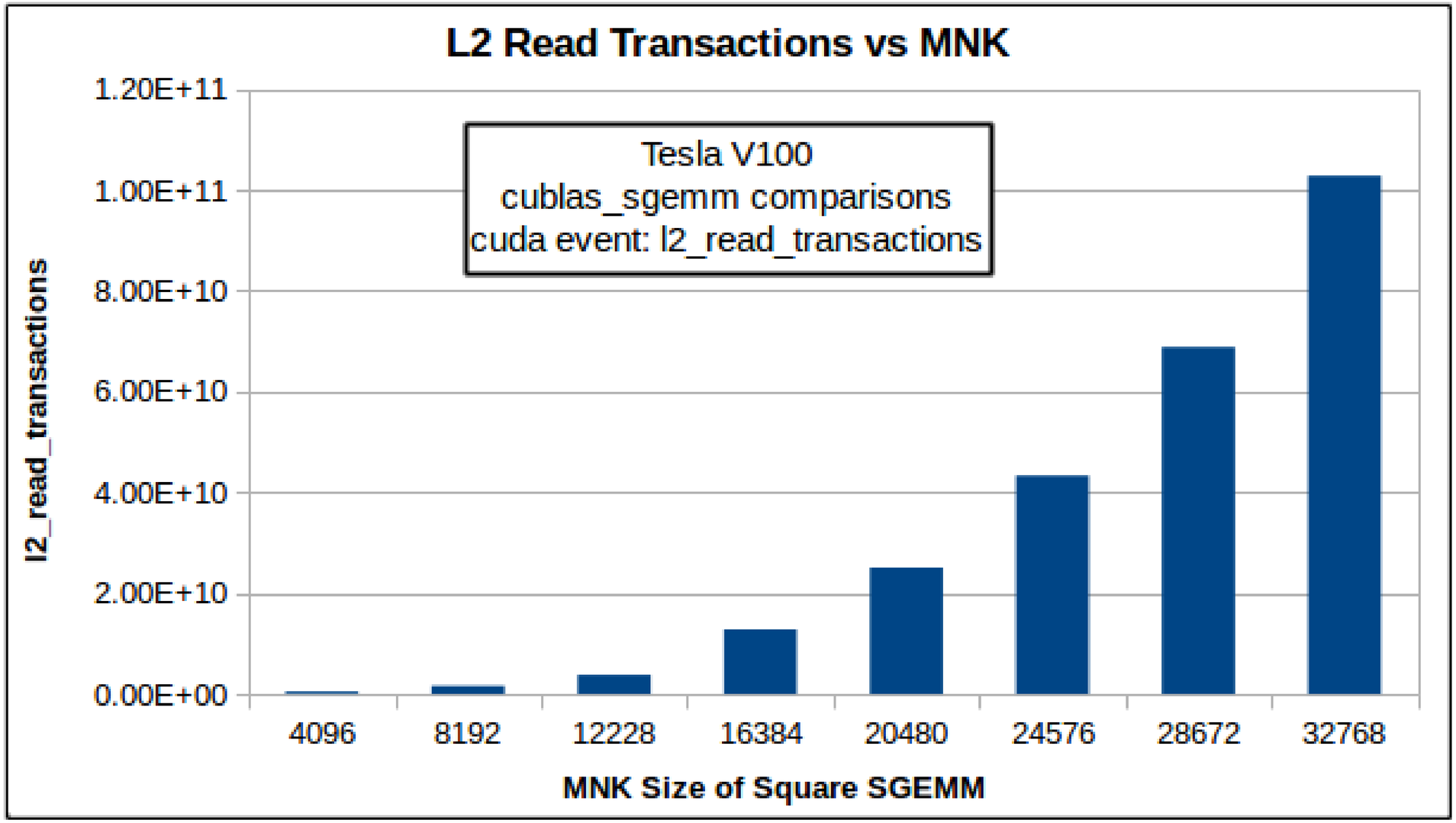
Output – Cuda



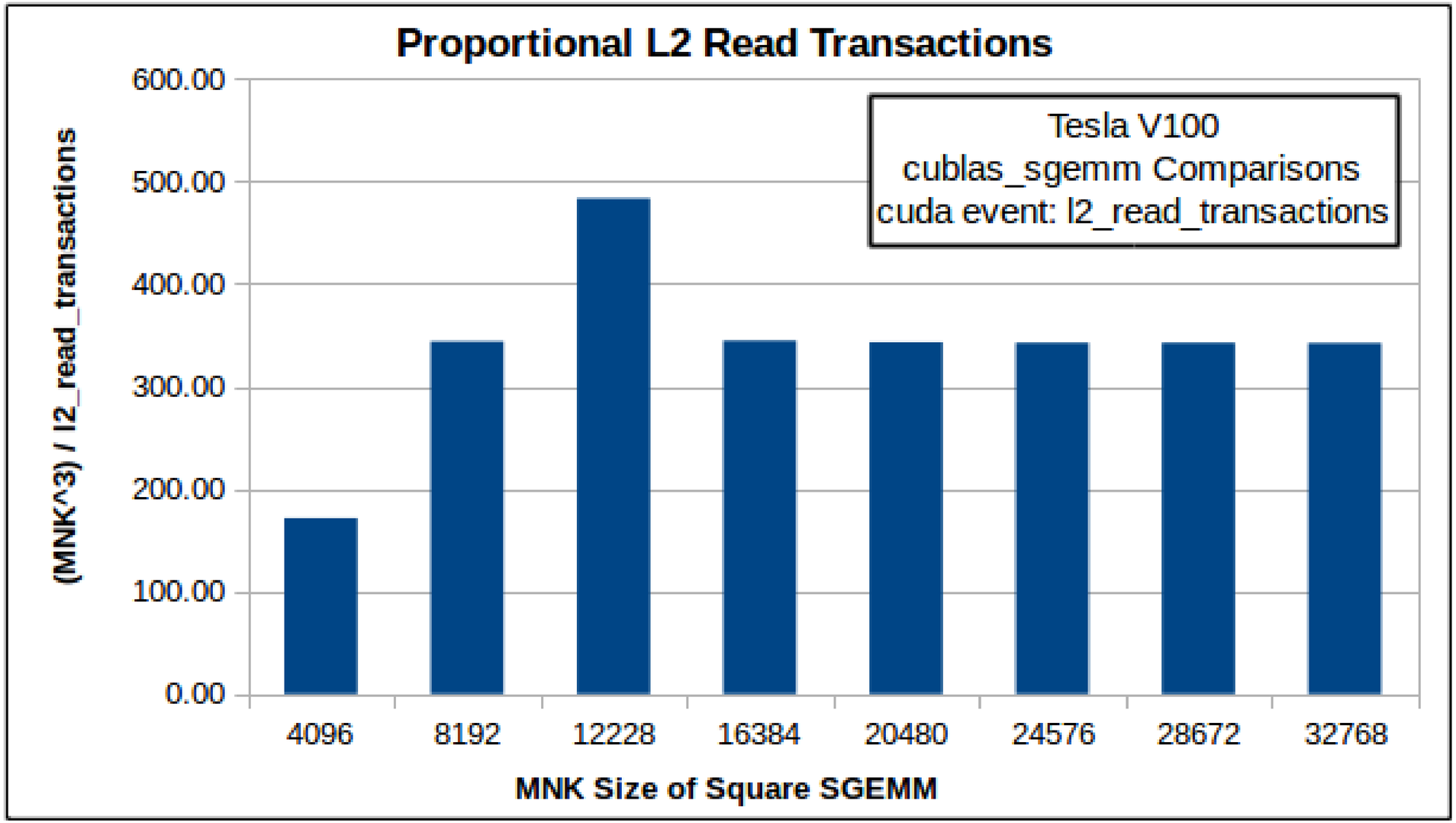
Output – Cuda



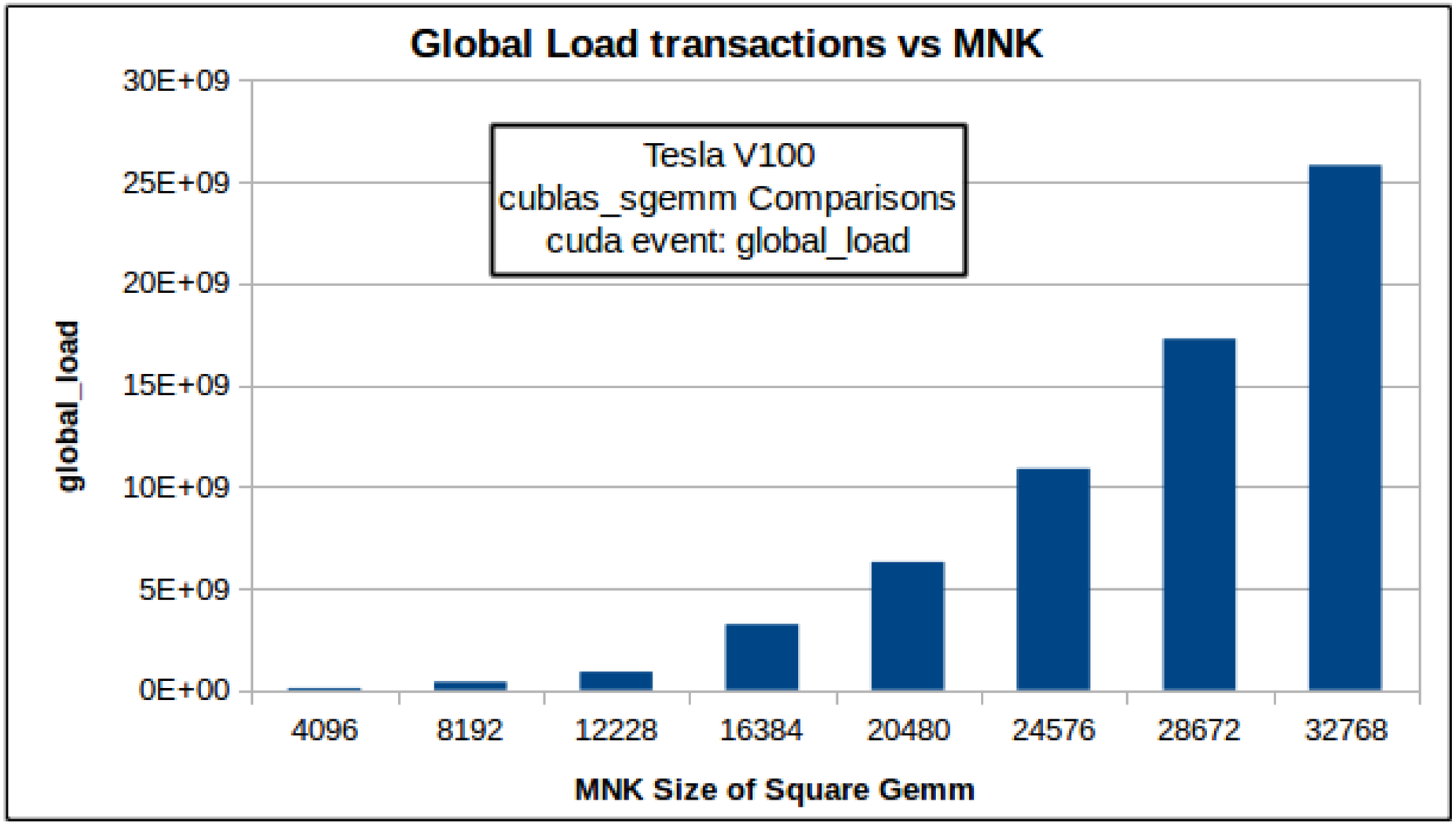
Output – Cuda



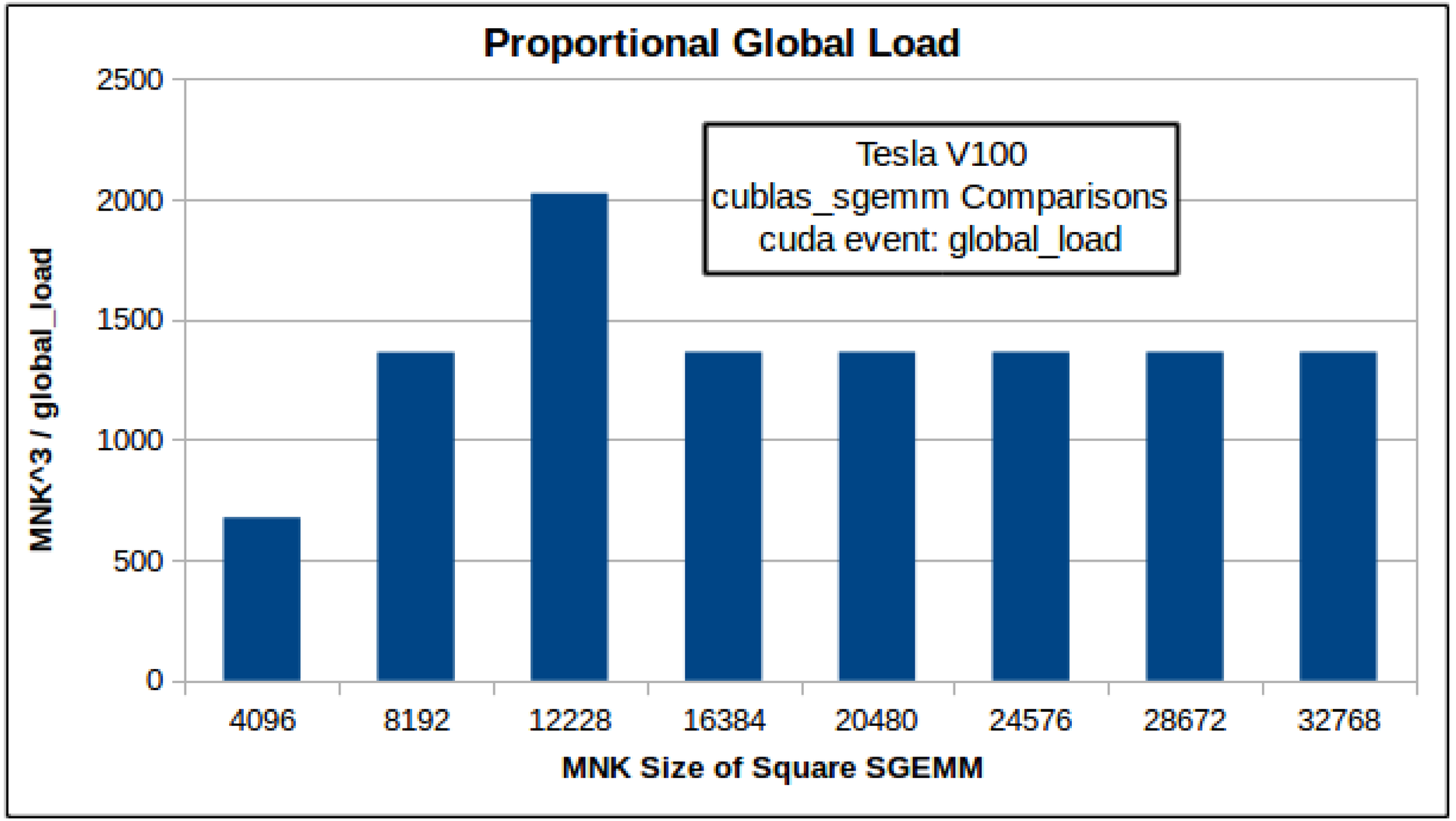
Output – Cuda



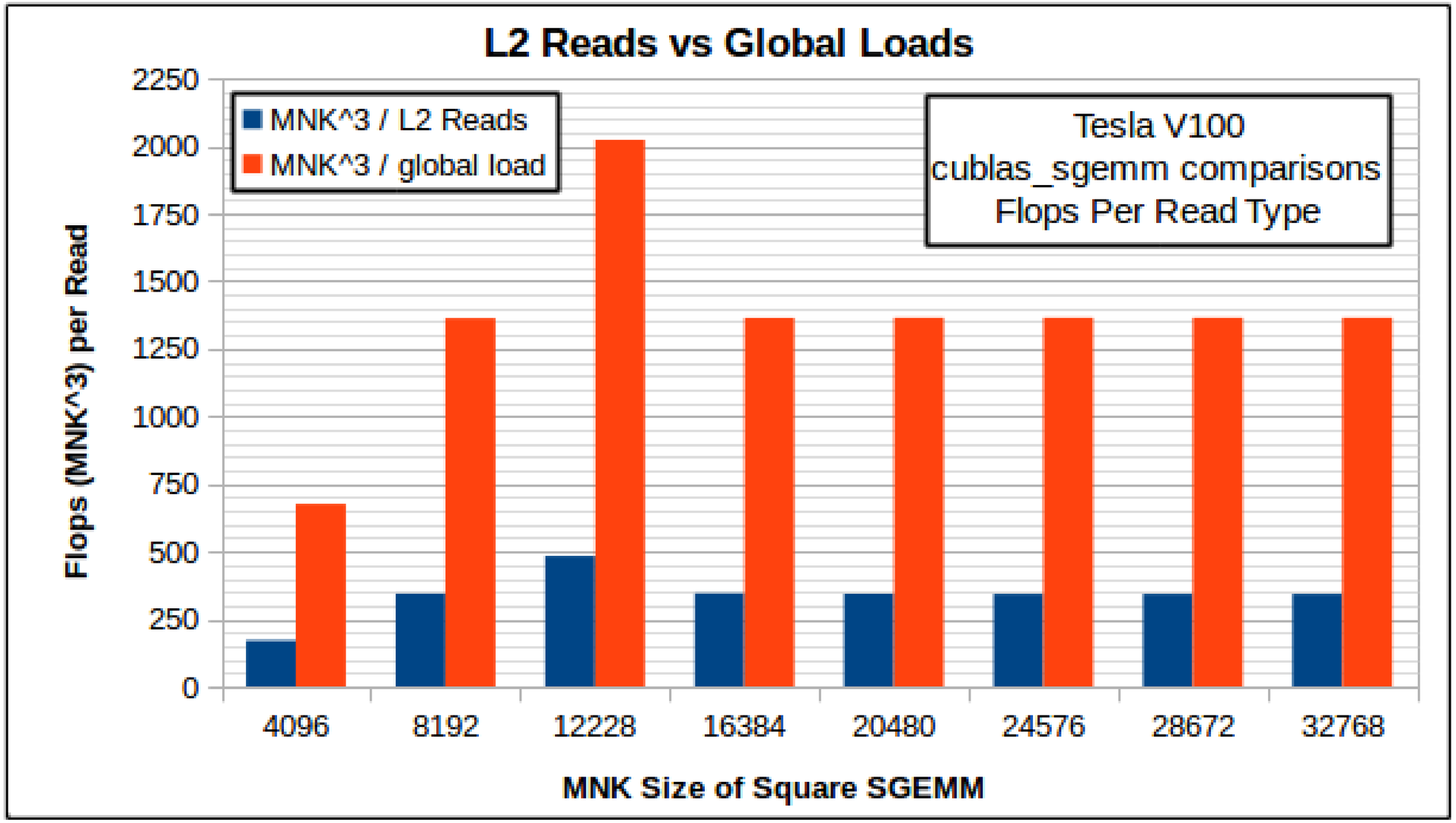
Output – Cuda



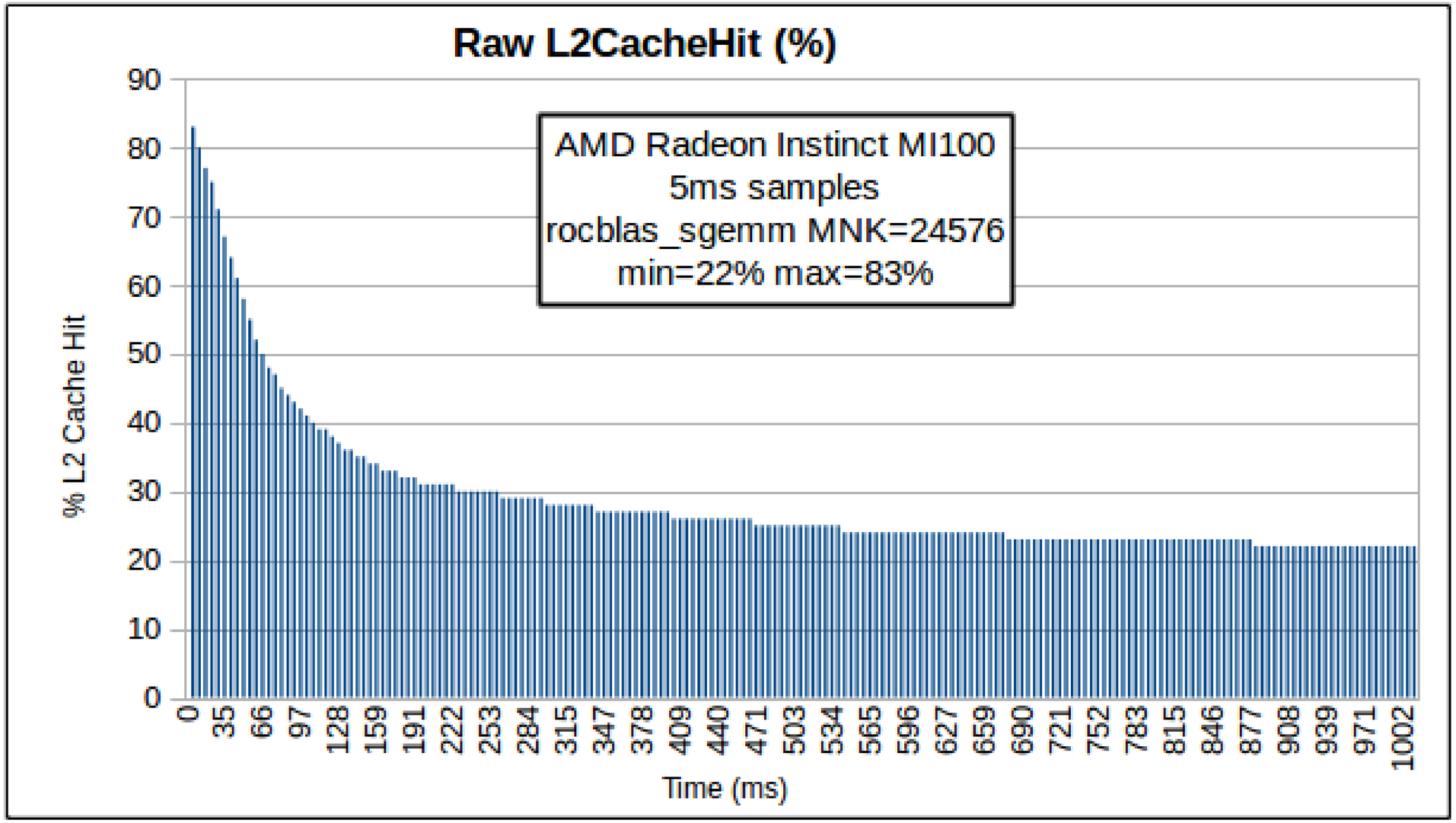
Output – Cuda



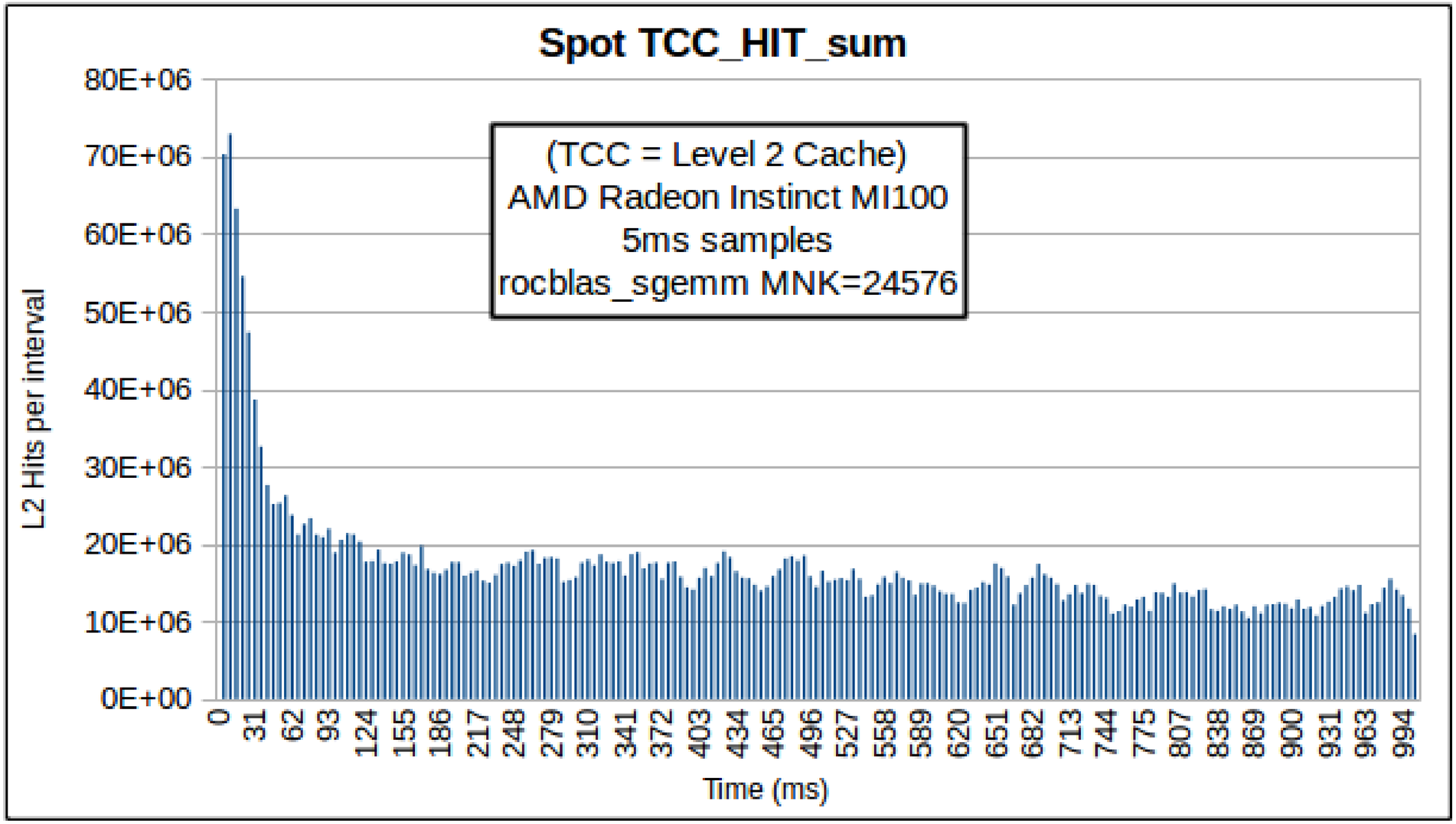
Output – Cuda



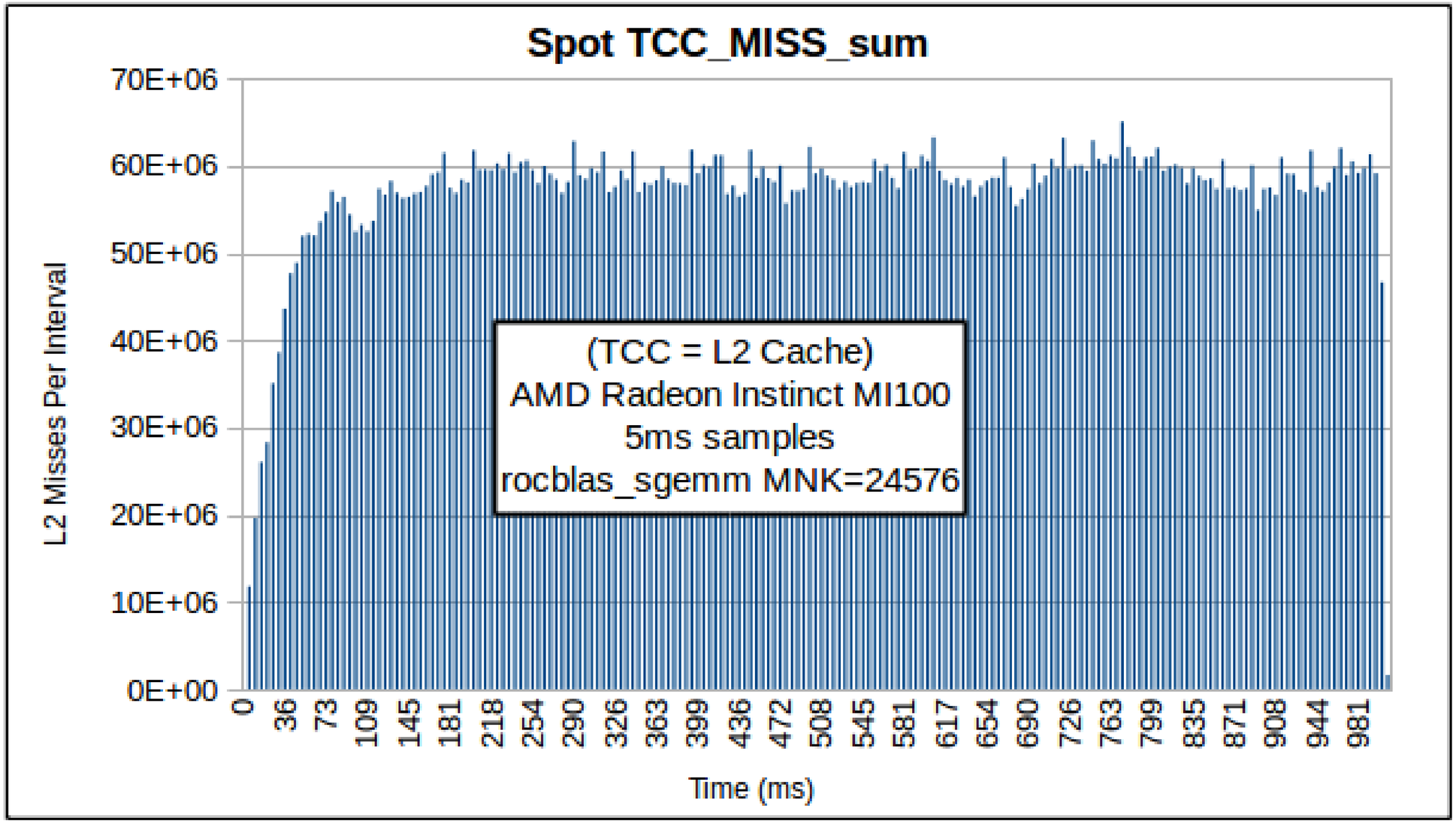
ROCM:::L2CacheHit:device=0



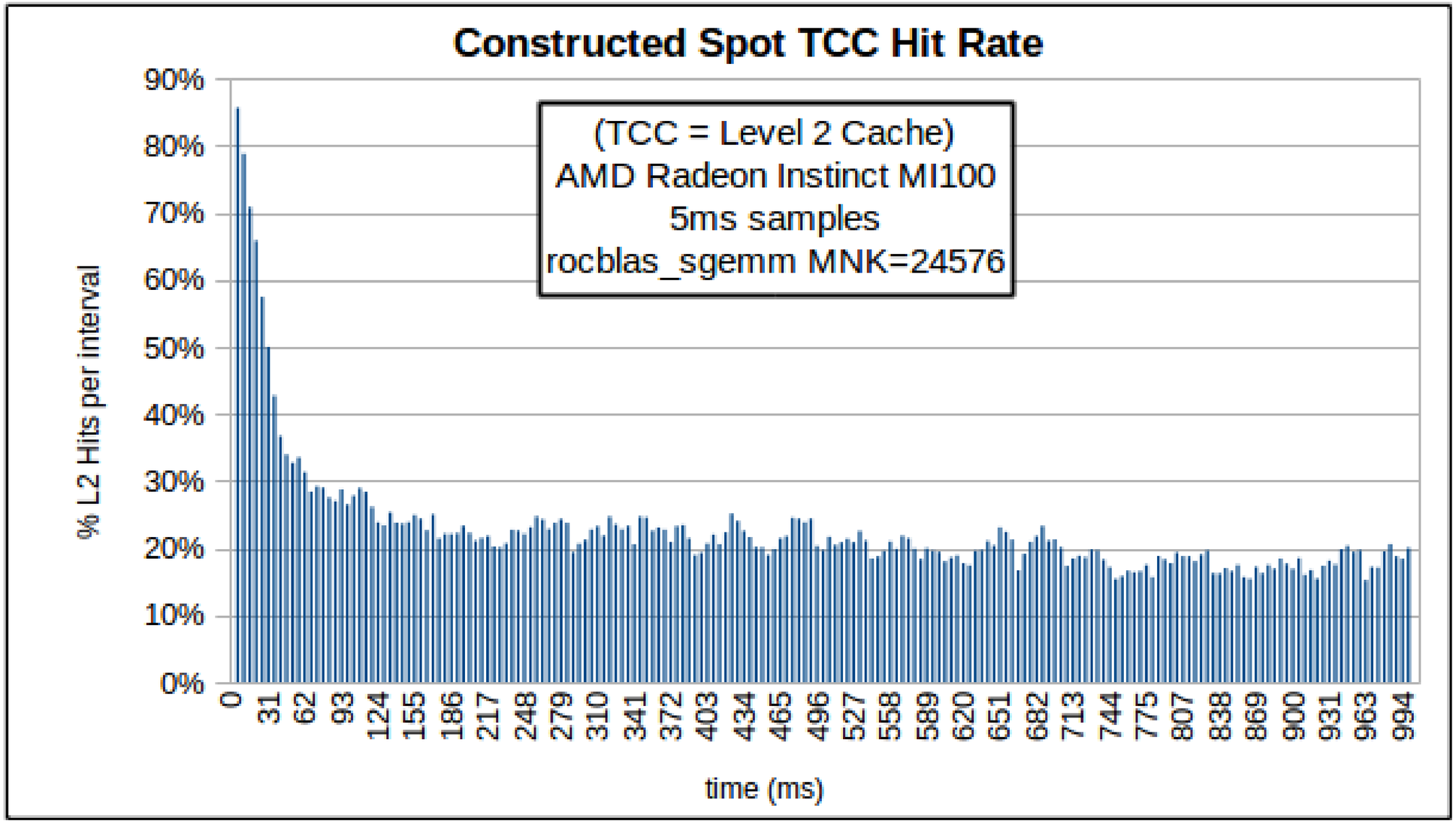
ROCM:::TCC_HIT_sum:device=0



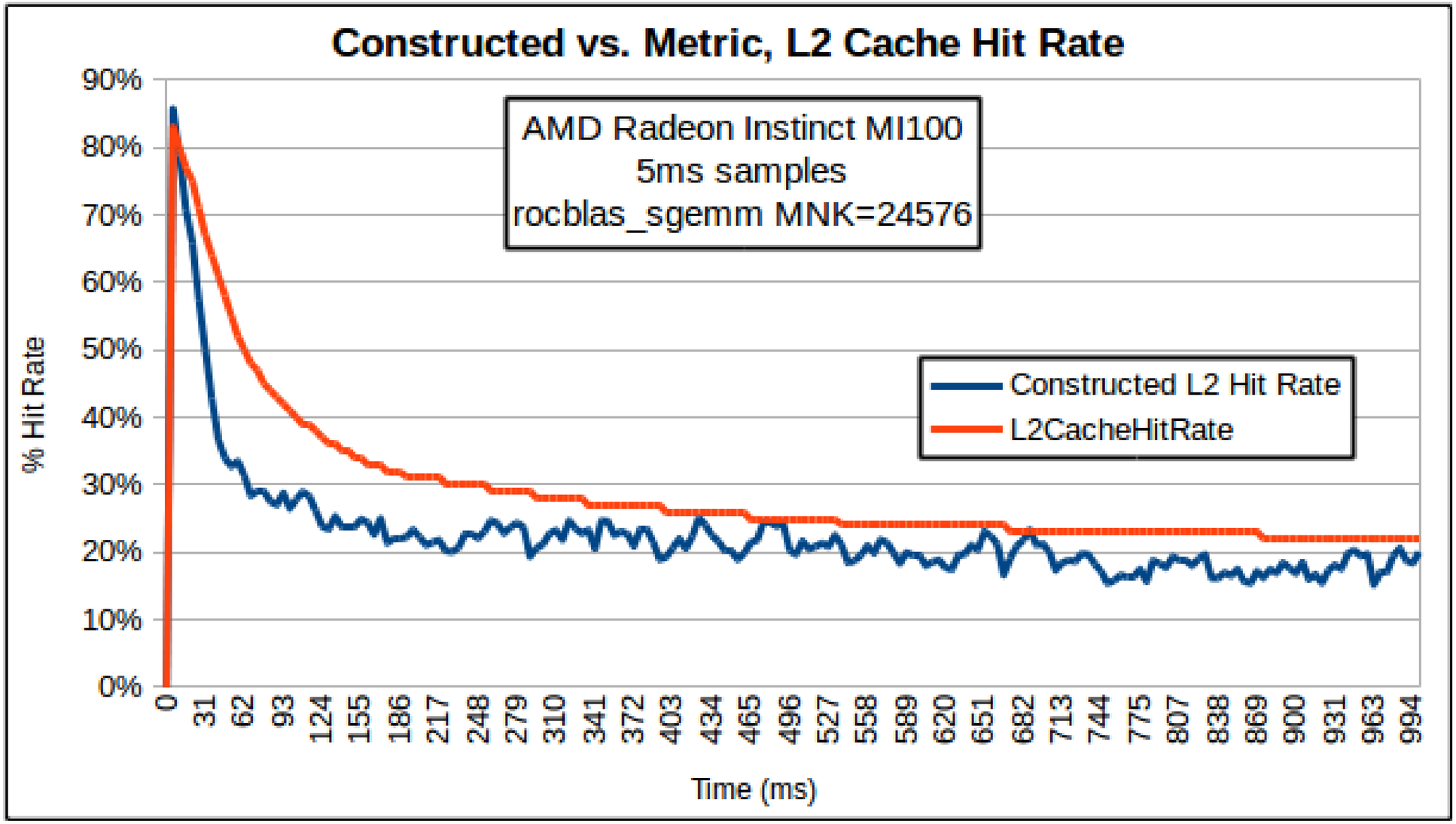
ROCM:::TCC_MISS_sum:device=0



Spot TCC Hit Rate: TCC_HIT / (TCC_HIT+TCC_MISS)



Spot TCC Hit Rate: TCC_HIT / (TCC_HIT+TCC_MISS)



Questions?

Contacts

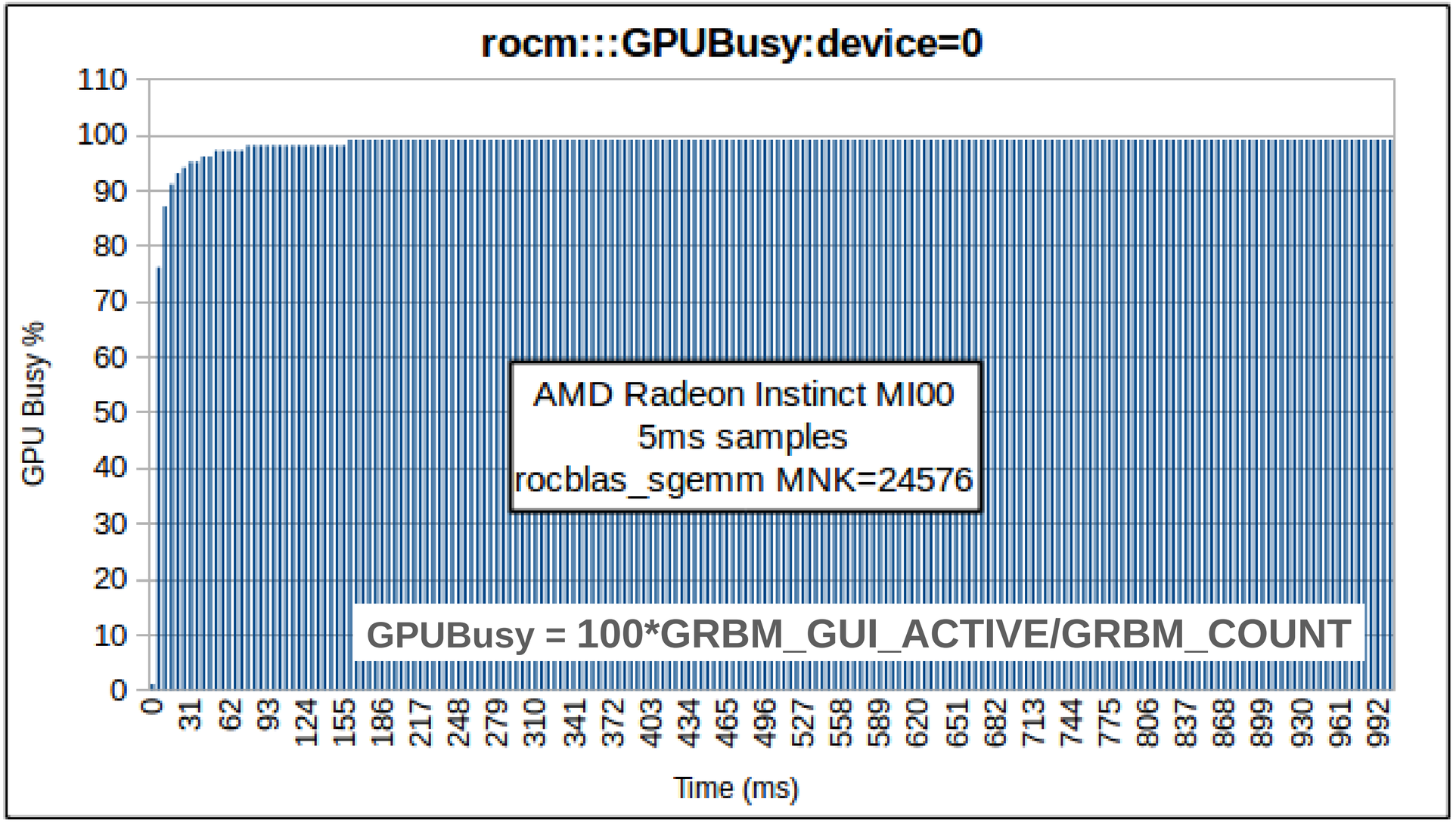
Presenter: Anthony Castaldo TonyCastaldo@icl.utk.edu

Repository: https://bitbucket.org/icl/papi.git

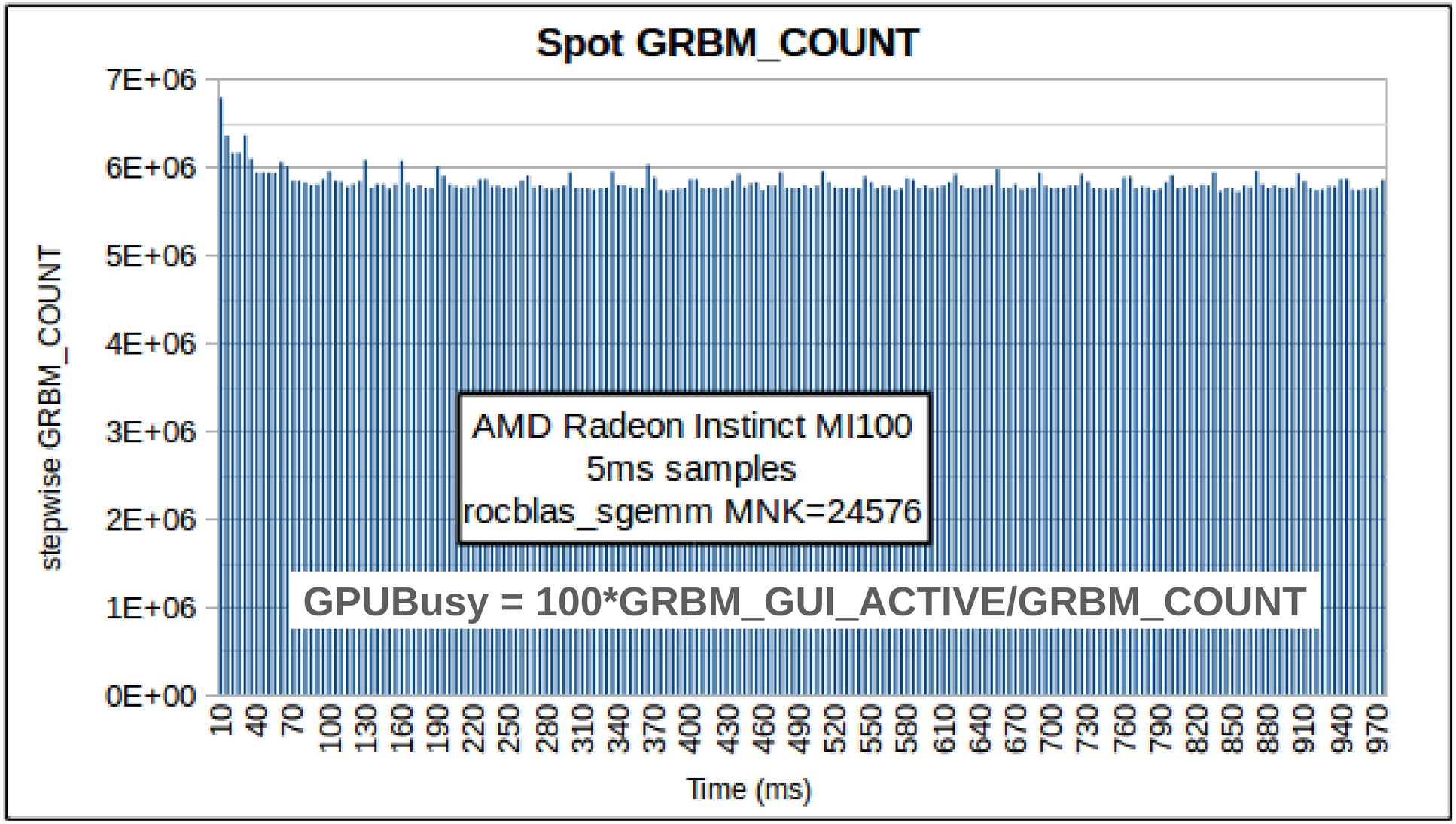
We aim to answer our help line. For assistance with PAPI, email
ptools-perfapi@icl.utk.edu

To read historical Q&A, join the PAPI User Google group by going to
<https://groups.google.com/a/icl.utk.edu/forum/#!forum/ptools-perfapi>

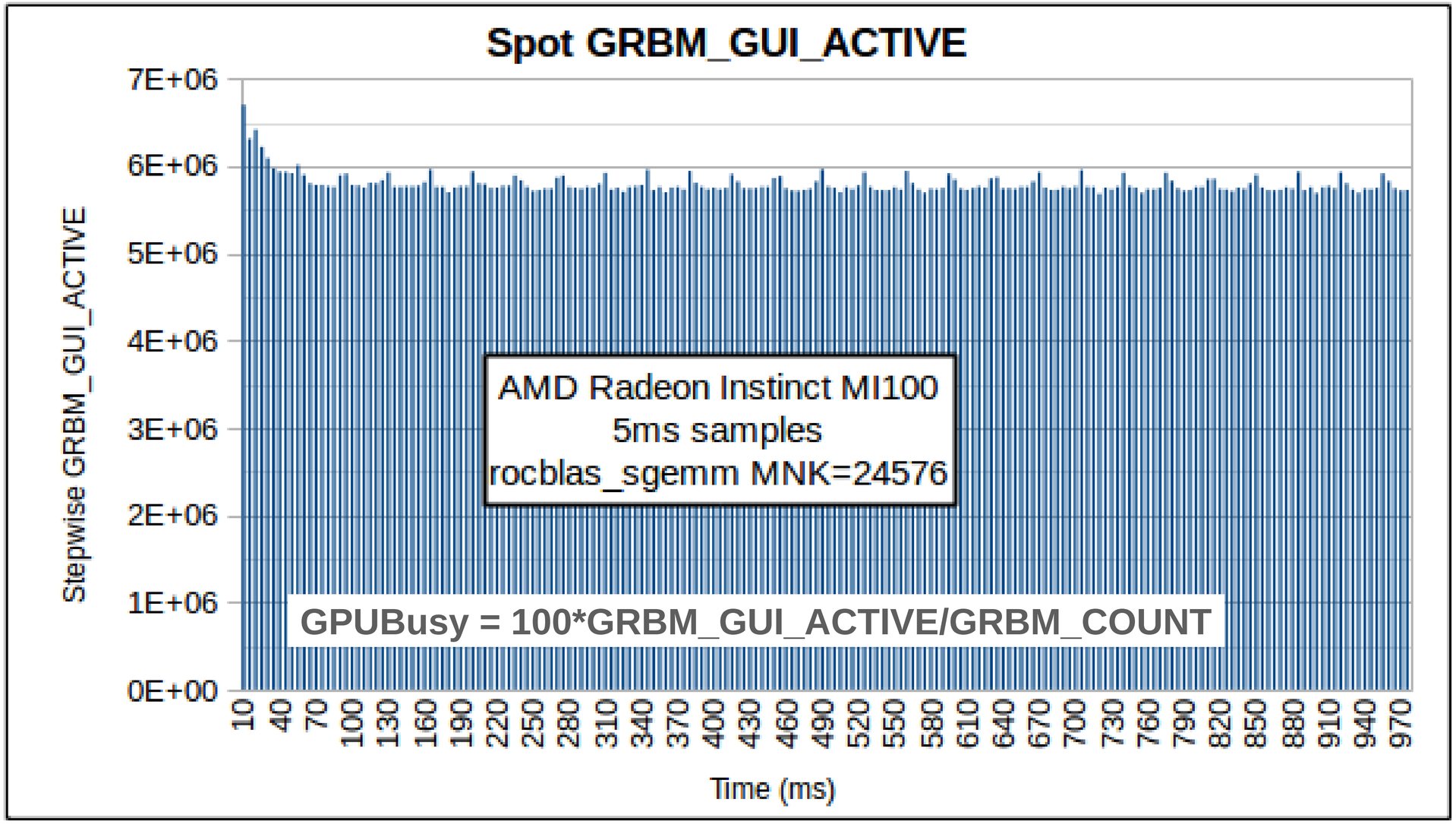
ROCM:::GPUBusy:device=0



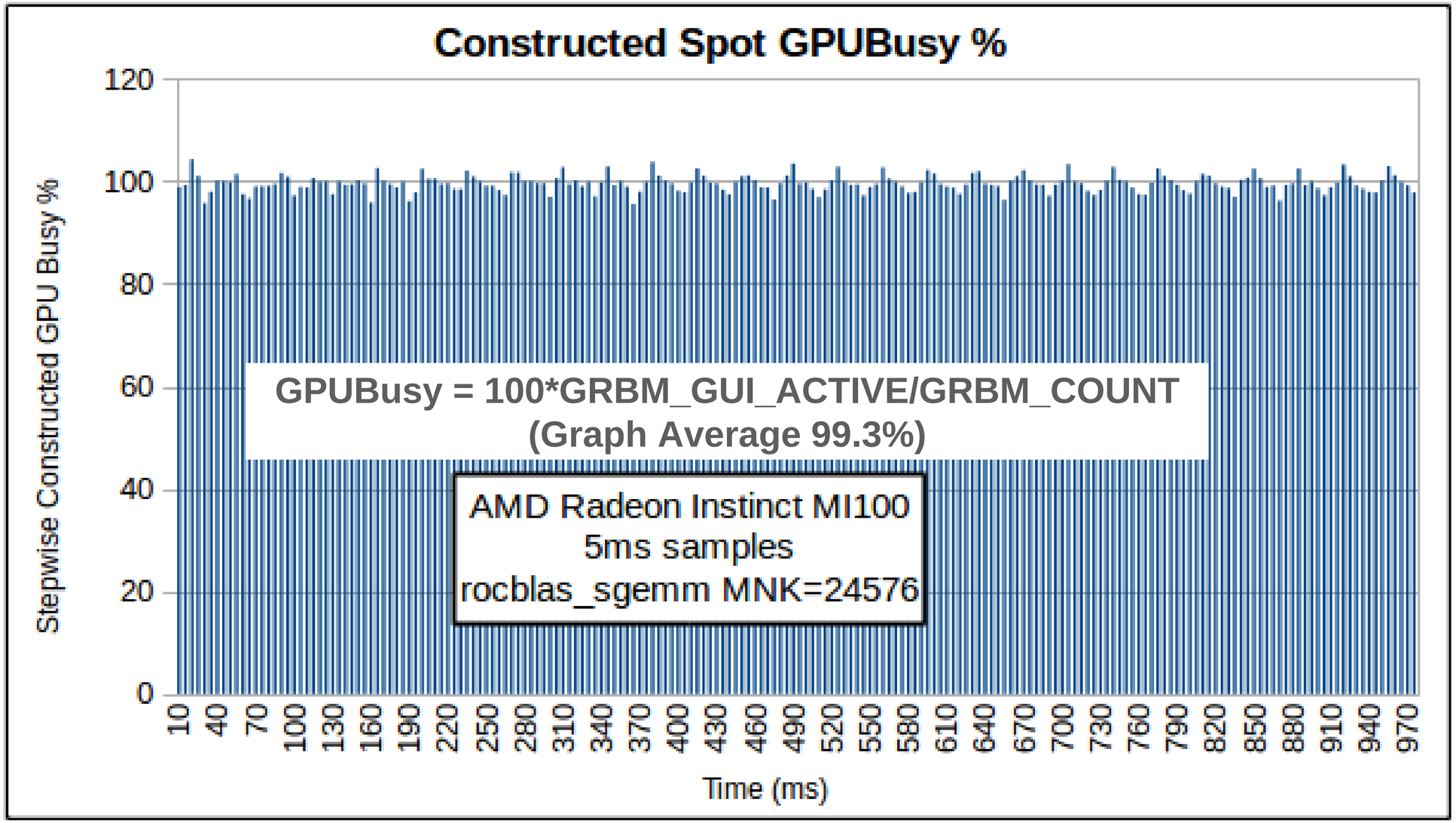
ROCM:::GRBM_COUNT:device=0



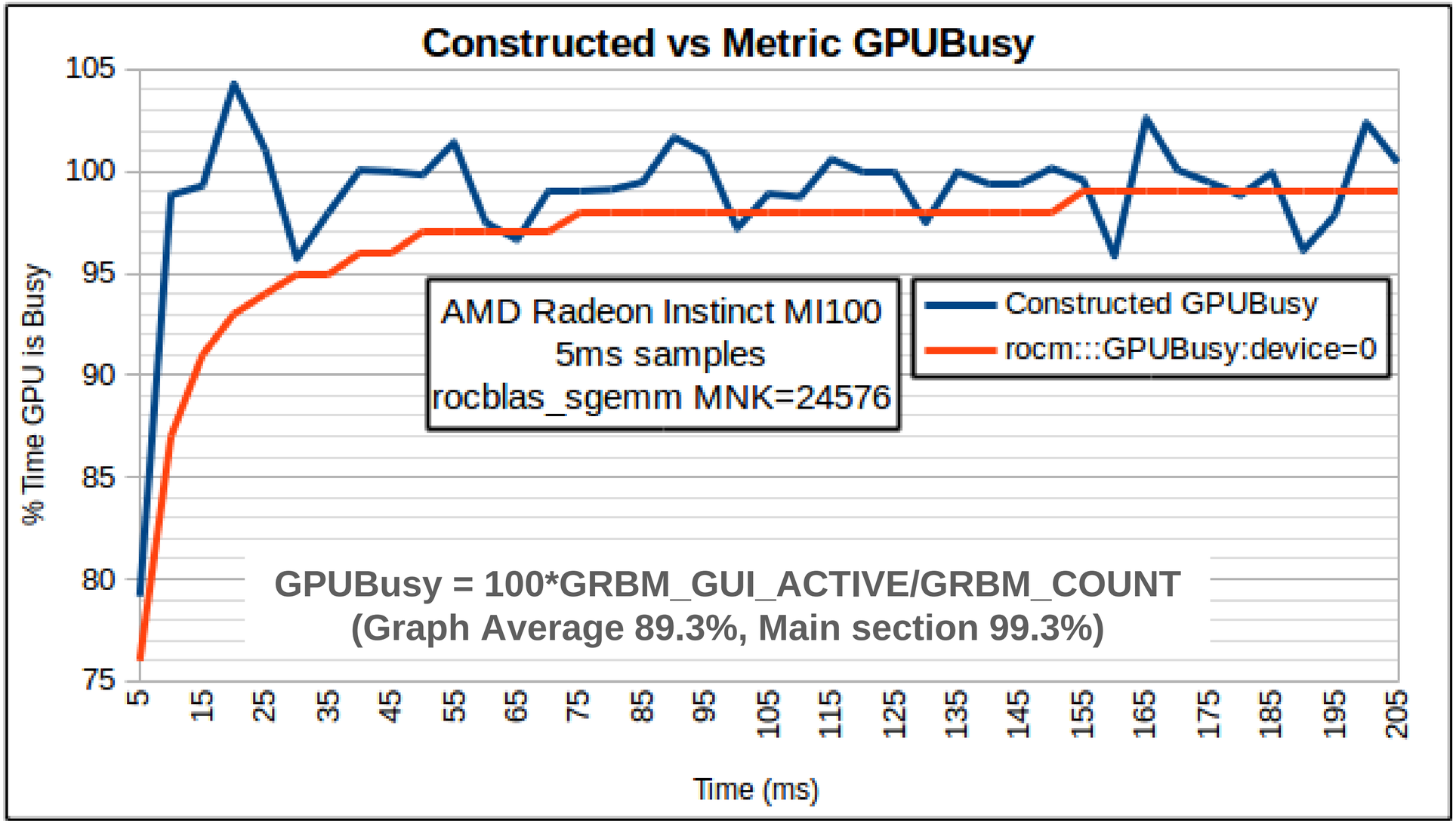
ROCM:::GRBM_GUI_ACTIVE:device=0



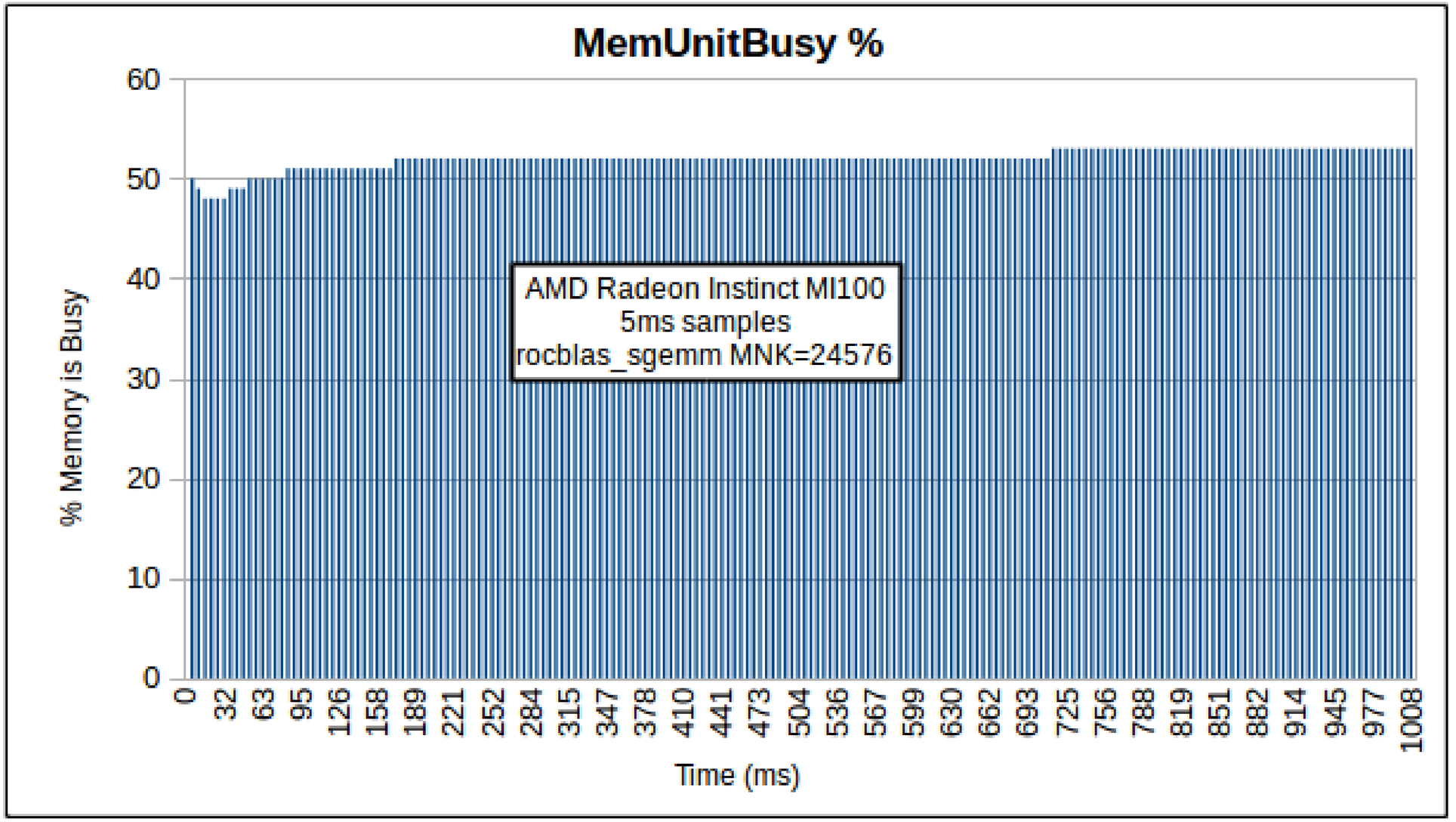
Constructed Spot GPU_Busy



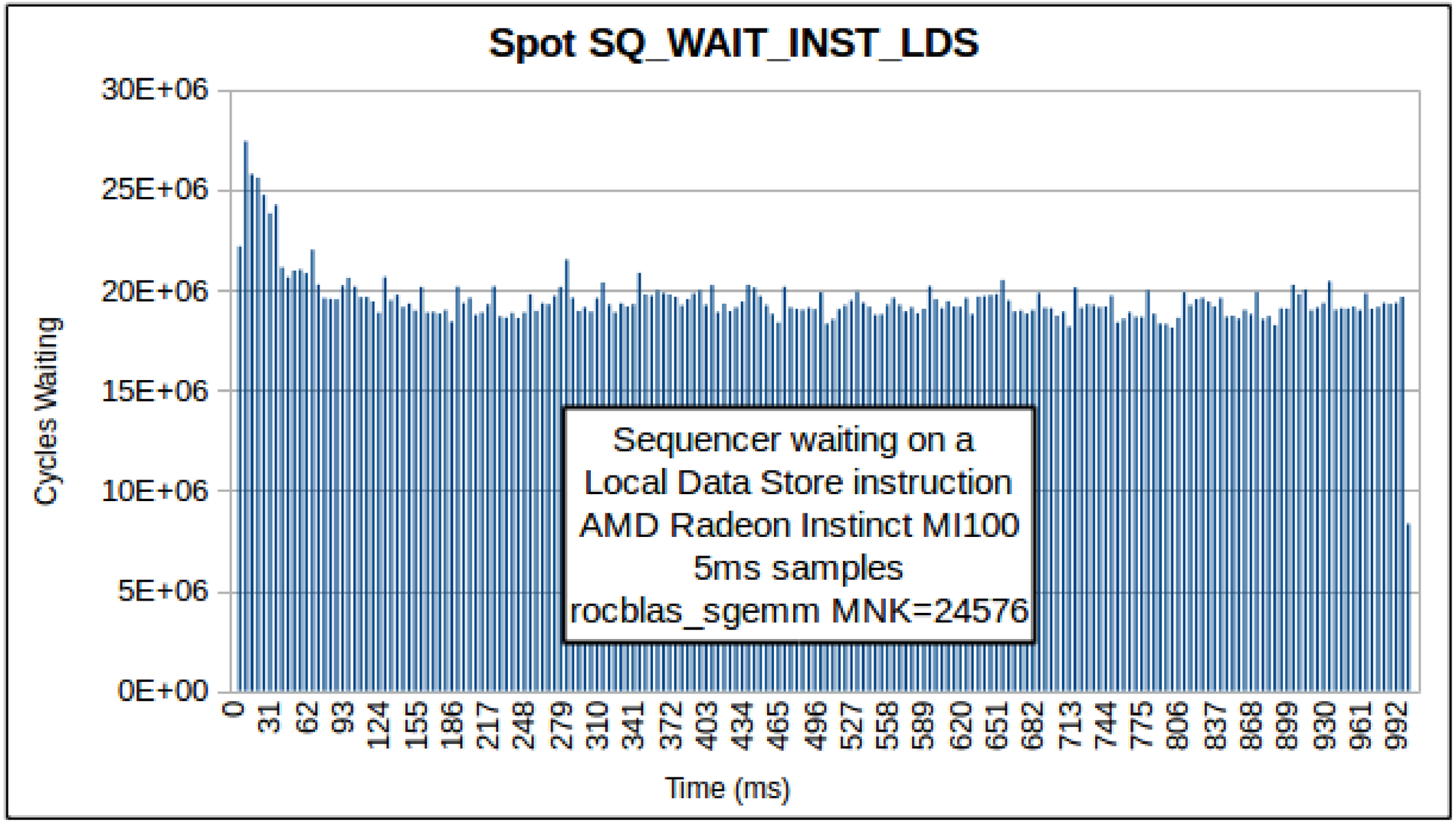
Constructed Spot GPU_Busy



ROCM:::MEM_UNIT_BUSY:device=0



ROCM:::SQ_WAIT_INST_LDS:device=0



ROCM:::SQ_INSTS_VALU:device=0