Utilizing GPU Acceleration for Numerical Integration in the DES Experiment

Ioannis Sakiotis (isaki001@odu.edu), Kamesh Arumugam,
Marc Paterno, Desh Ranjan, Balsa Terzic, Mohammad Zubair

INTRODUCTION

- Applications: parameter estimation, simulation of mean dynamics, risk management, ray-tracing
- PAGANI & m-Cubes
- Demonstrated significant performance gains
- Goal: use at scale

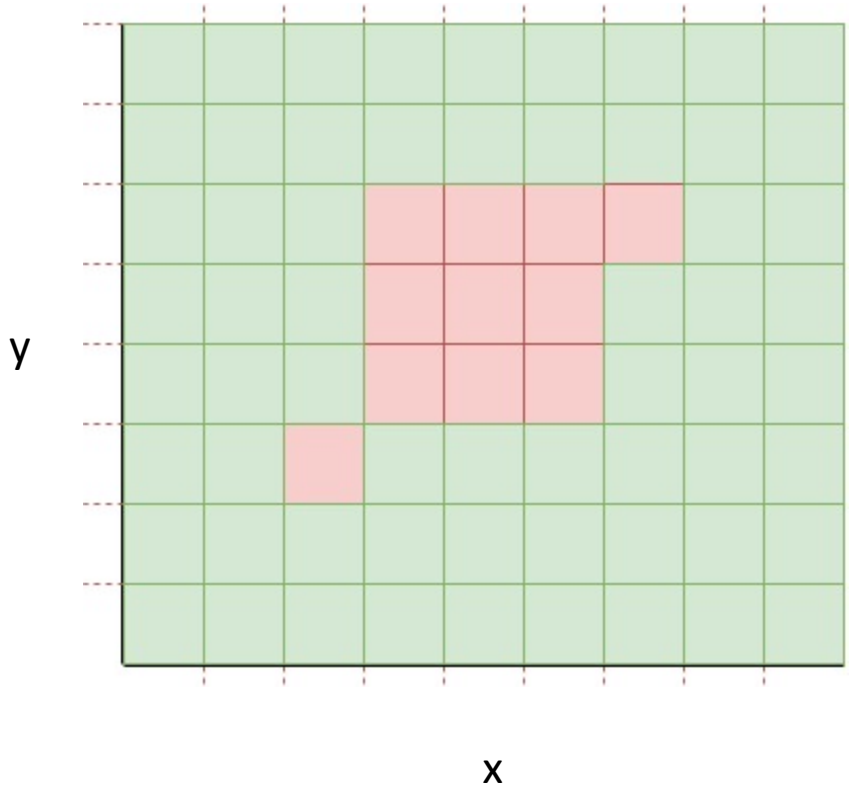
Outline

- Brief description of integration algorithms
- Pre-internship status of integration implementations
- Summer internship work

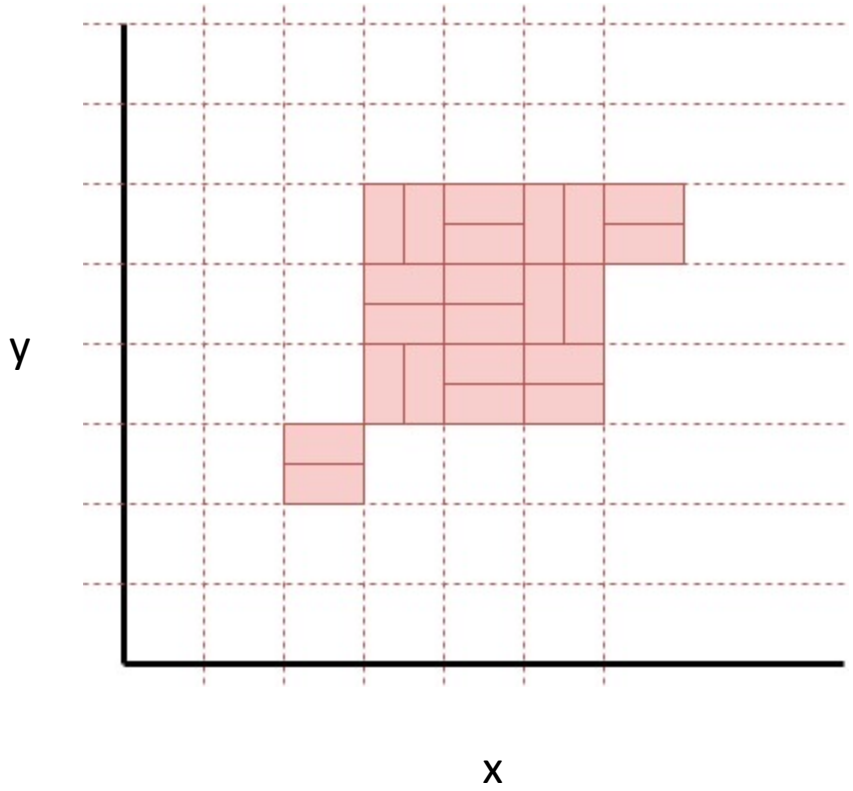
Parallel numerical integration

- Parallelize existing methods
  - VEGAS
  - Cuhre
- Parallel GPU methods
  - m-Cubes
  - PAGANI
- Mathematics identical
- Algorithmic changes
- Non-identical results

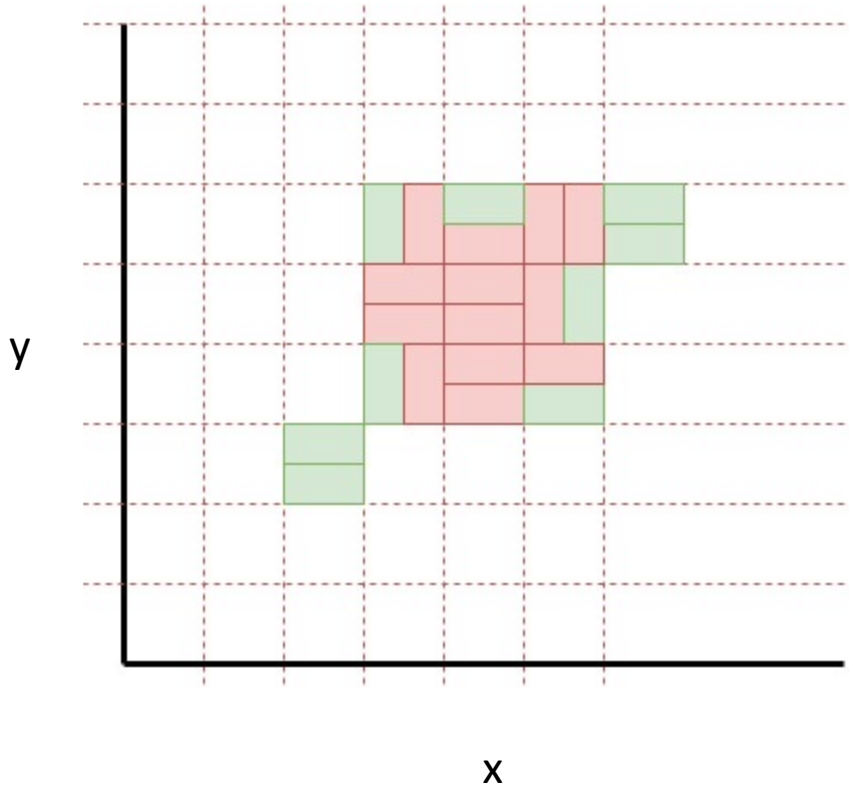
PAGANI



PAGANI

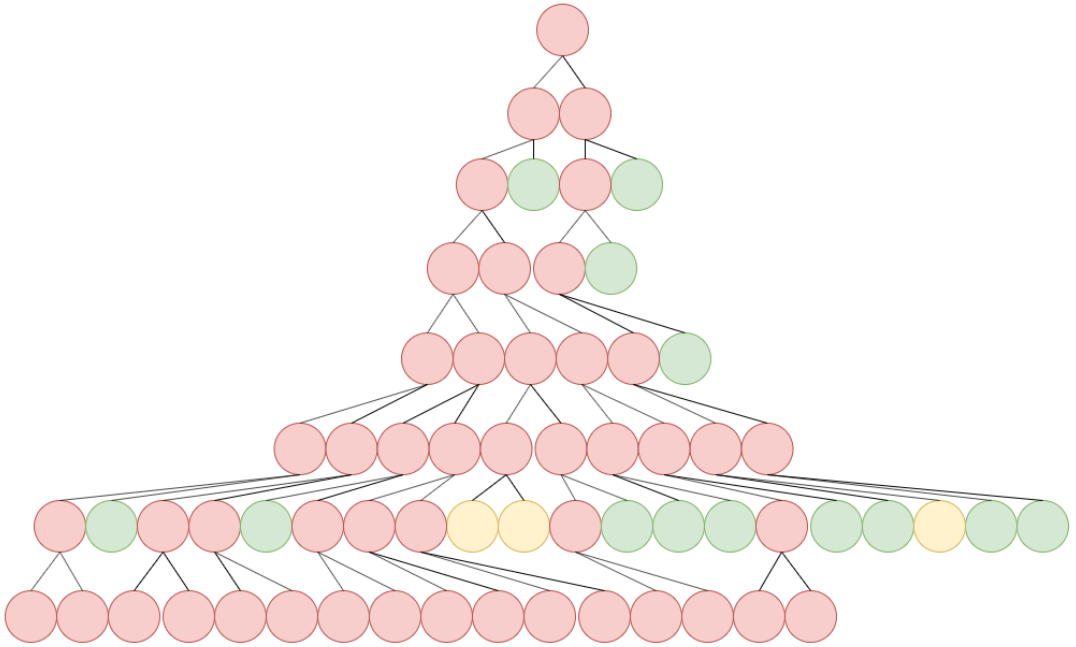


PAGANI

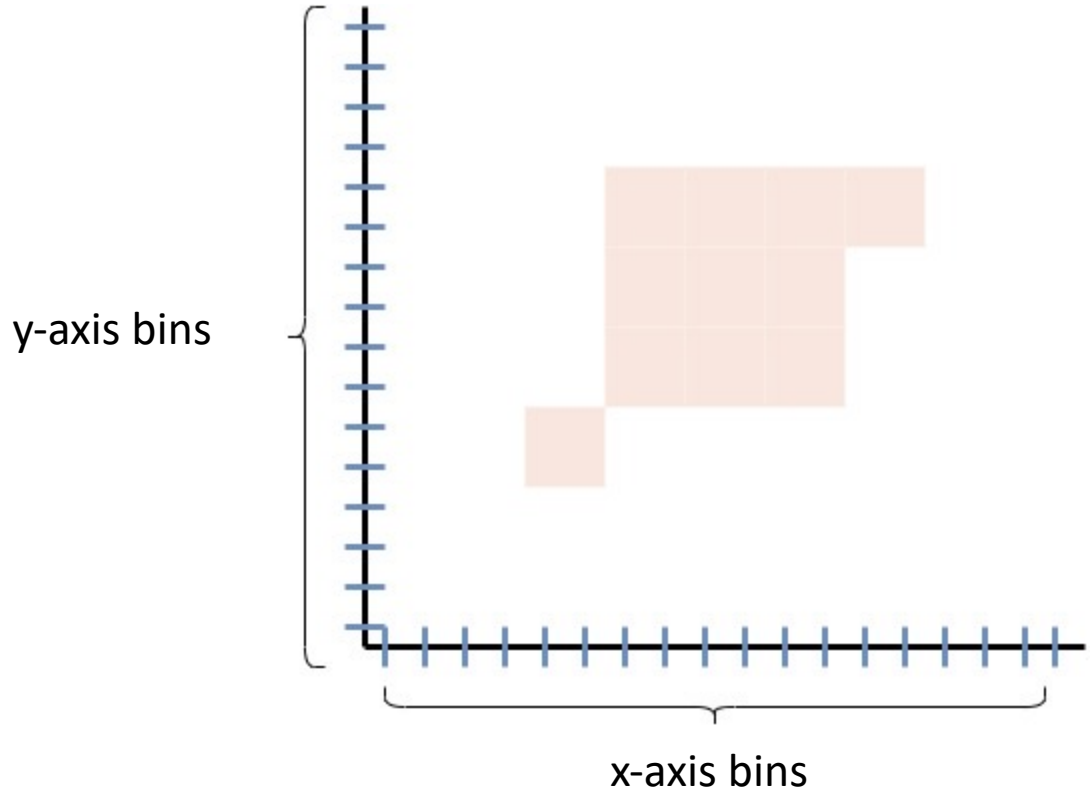


PAGANI

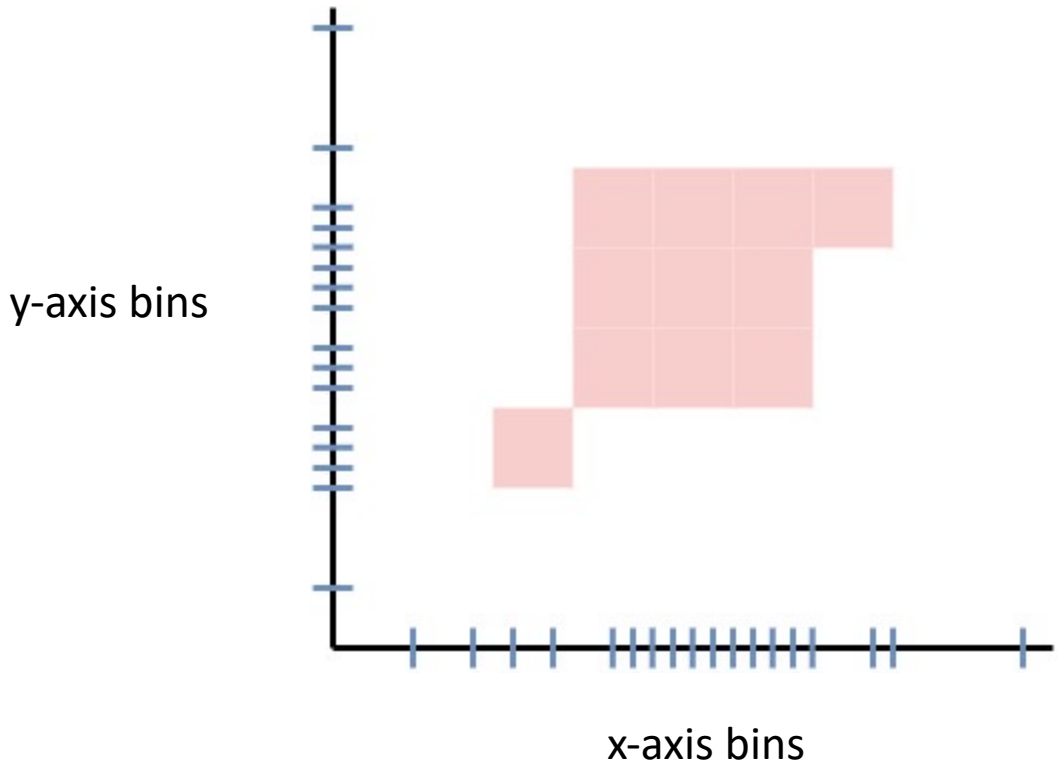
- Quadrature rules
- Error-estimate
- Iterative algorithm
- Generate regions within integration space
- Sub-divide regions (nodes)
- Classify regions
- Green/yellow = accurate enough
- Red = needs sub-division



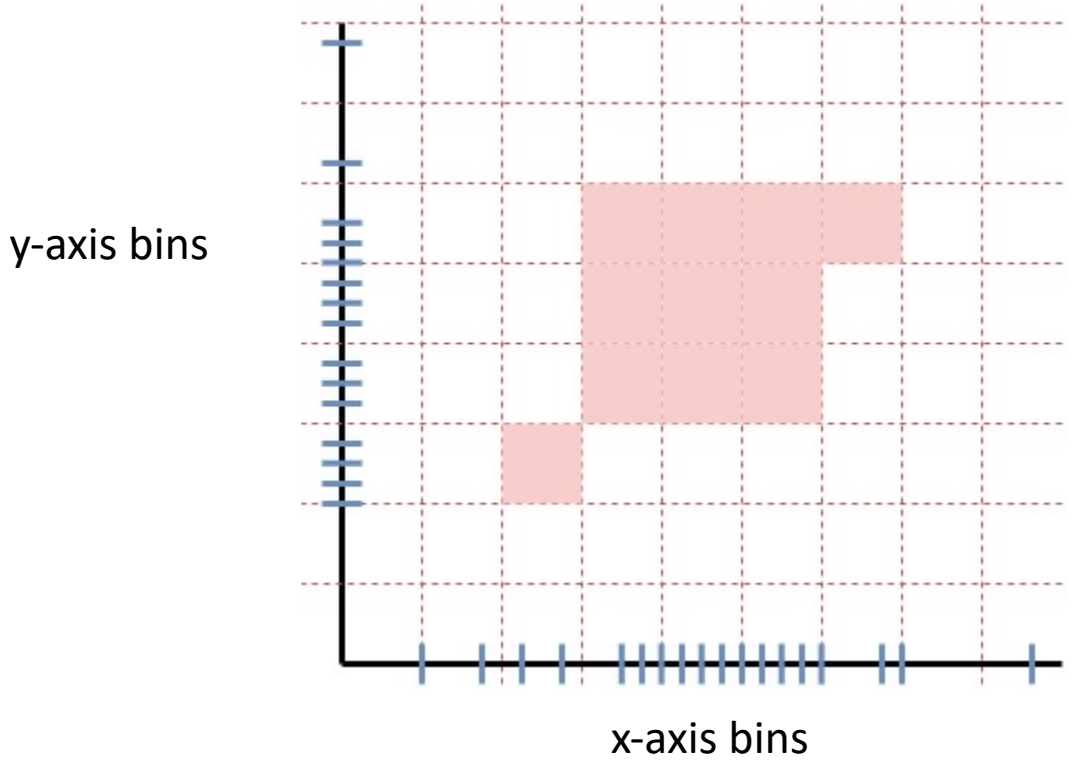
m-Cubes



m-Cubes



m-Cubes



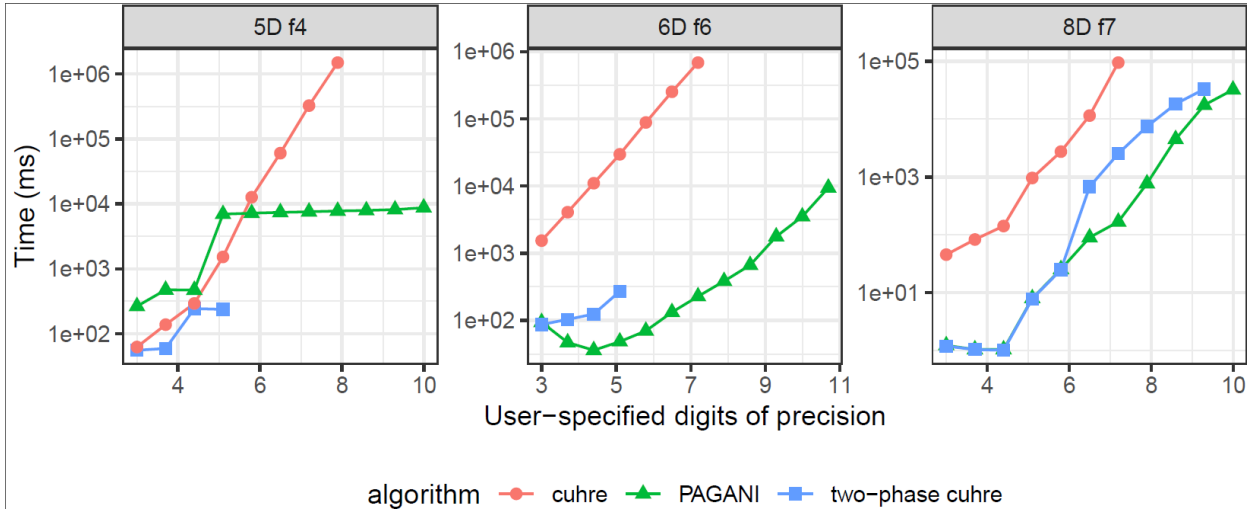
Interface

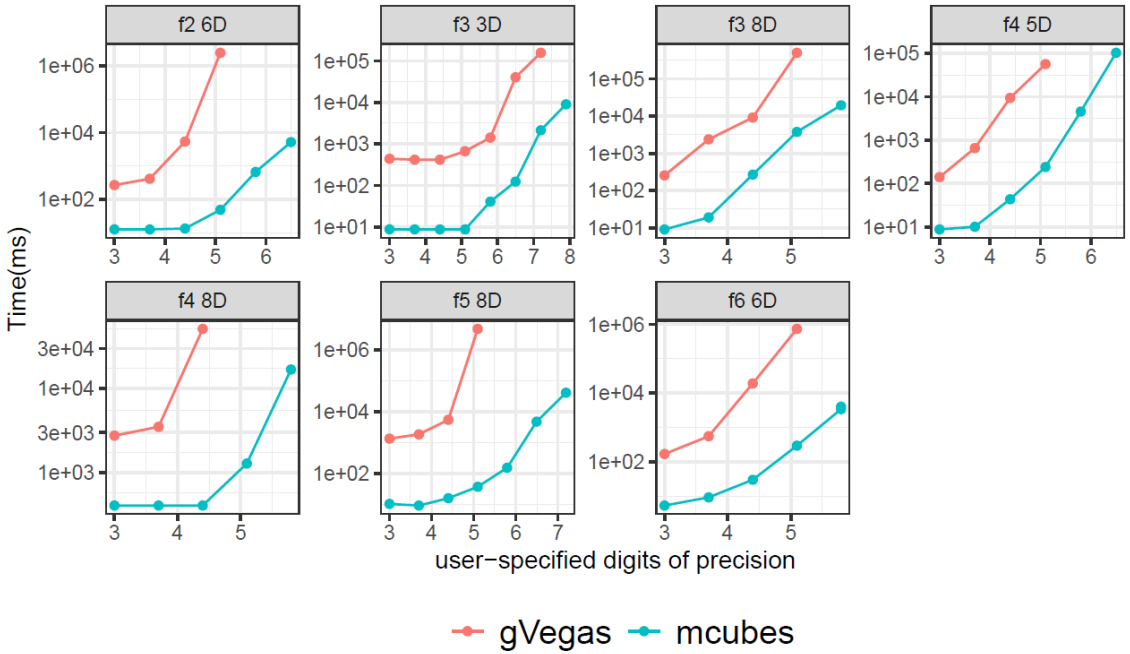
- C++ object
- Function arguments
  - Integrand
  - Relative/absolute error tolerances
  - Integration-space bounds
- Template arguments
  - Integrand type
  - Dimensionality

```
class Gaussian {
public:
  __device__ __host__
 double
 operator()(double x, double y,  double z,  double w)
 {
    double beta = .5;
    return exp(-1.0 *
        (pow(25., 2.) * (pow(x - beta, 2) + pow(y - beta, 2) +
pow(z - beta, 2) + pow(w - beta, 2) )));
 }
};
```

```
Gaussian integrand;
integrate<Gaussian, ndim>(integrand, epsrel, epsabs,
nsamples);
```

Performance





Scaling

- Perlmutter
- 256 MPI ranks
- 1 GPU/rank
- Hundreds samples
- 4 integrands
- 4 volumes
- 69 grid-points = 69 integrations

Scaling

- Perlmutter
- 256 MPI ranks
- 1 GPU/rank
- Hundreds samples
  - 4 integrands
  - 4 volumes
  - 69 grid-points
- Error: Unknown error
- Failure mode triggered after hundreds of integrations

The crash

- Reverted to 1 MPI rank
- Identical crash but took longer to fail
- Allocation in copy-constructor
- Object for 2D interpolation
- Data must be on GPU memory

Detecting memory misuse

- “Compute sanitizer” reported no leaks
- **cudaMemGetInfo**: returns available GPU memory in bytes
- Recorded free memory within copy constructor
- Needed higher resolution memory tracking

Data collection

- **MemTrackSentry** class
- Record free GPU memory upon construction/deletion
- “Leaked” = construction – deletion bytes
- Module execution to generate sample

```
void execute(){

   MemTrackSentry mem_sentry(
            ofstream_var,
            “integrand_label”,
            "module");

   do_integrations();
}
```

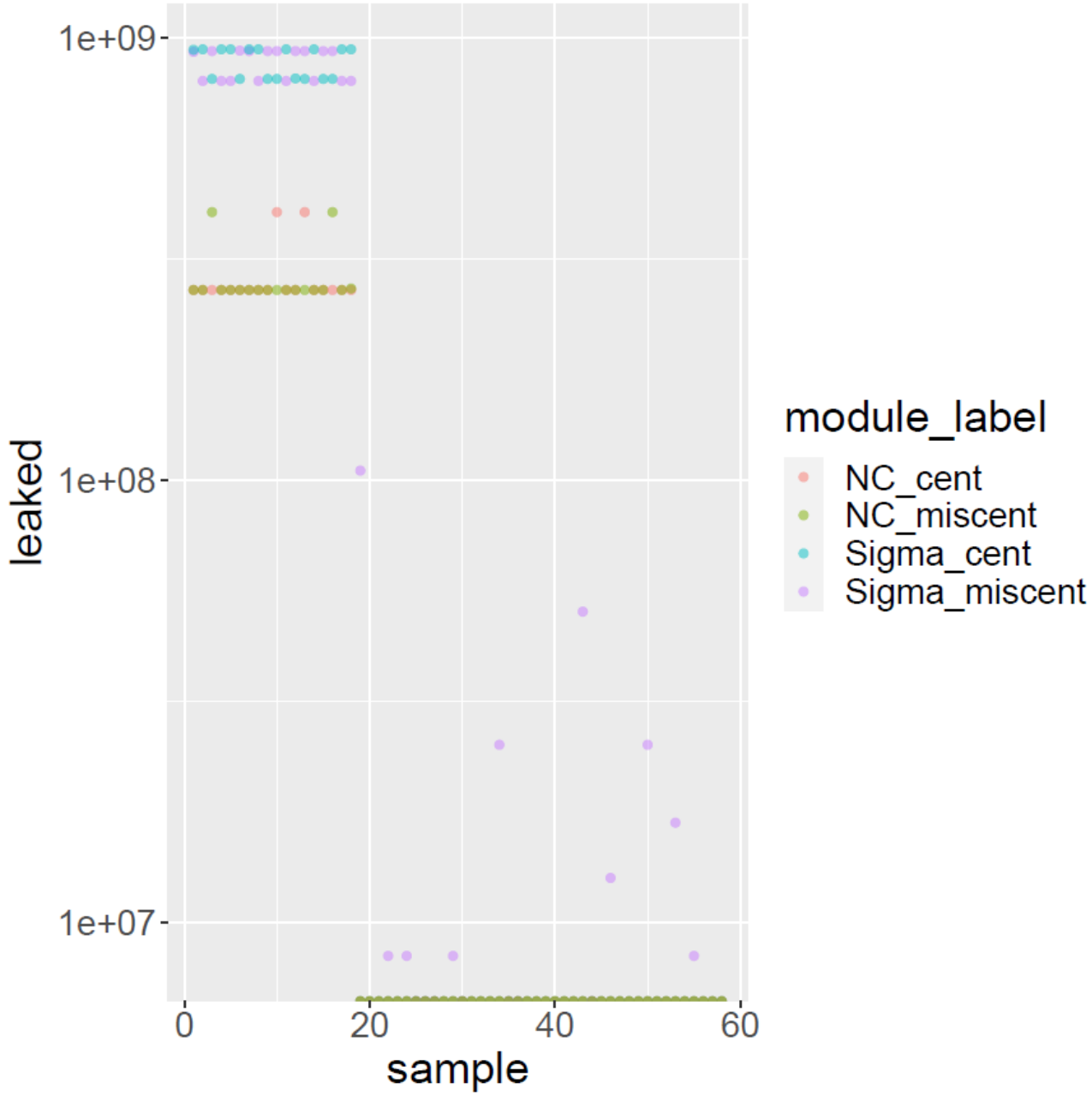
Data collection

- **MemTrackSentry** class
- Record free GPU memory upon construction/deletion
- “Leaked” = construction – destruction bytes
- Module execution to generate sample
- Volume
- Call to integration routine

```
void do_integrations(){

 for(auto& volume: volumes) {
    MemTrackSentry mem_sentry(
          ofstream_var
          “integrand_label”,
          “volume");
    for(auto& point: grid_points) {
          MemTrackSentry mem_sentry(…, “integration”)
          integrate(…)
    }
 }
}
```

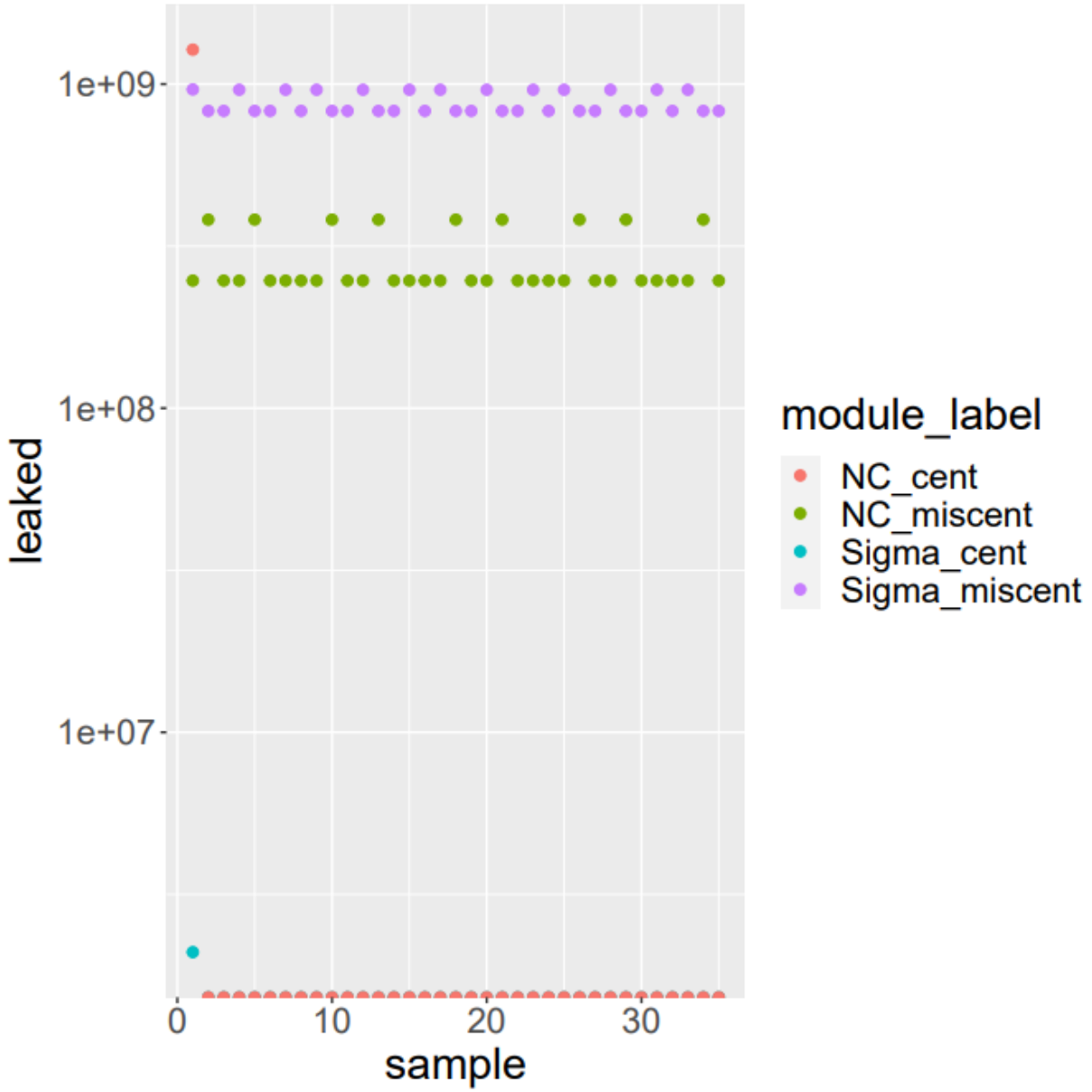
Sample level

- Many leaks
- Validated our hypothesis
- Leaks of different sizes
- Caused by module, integration, integrands?

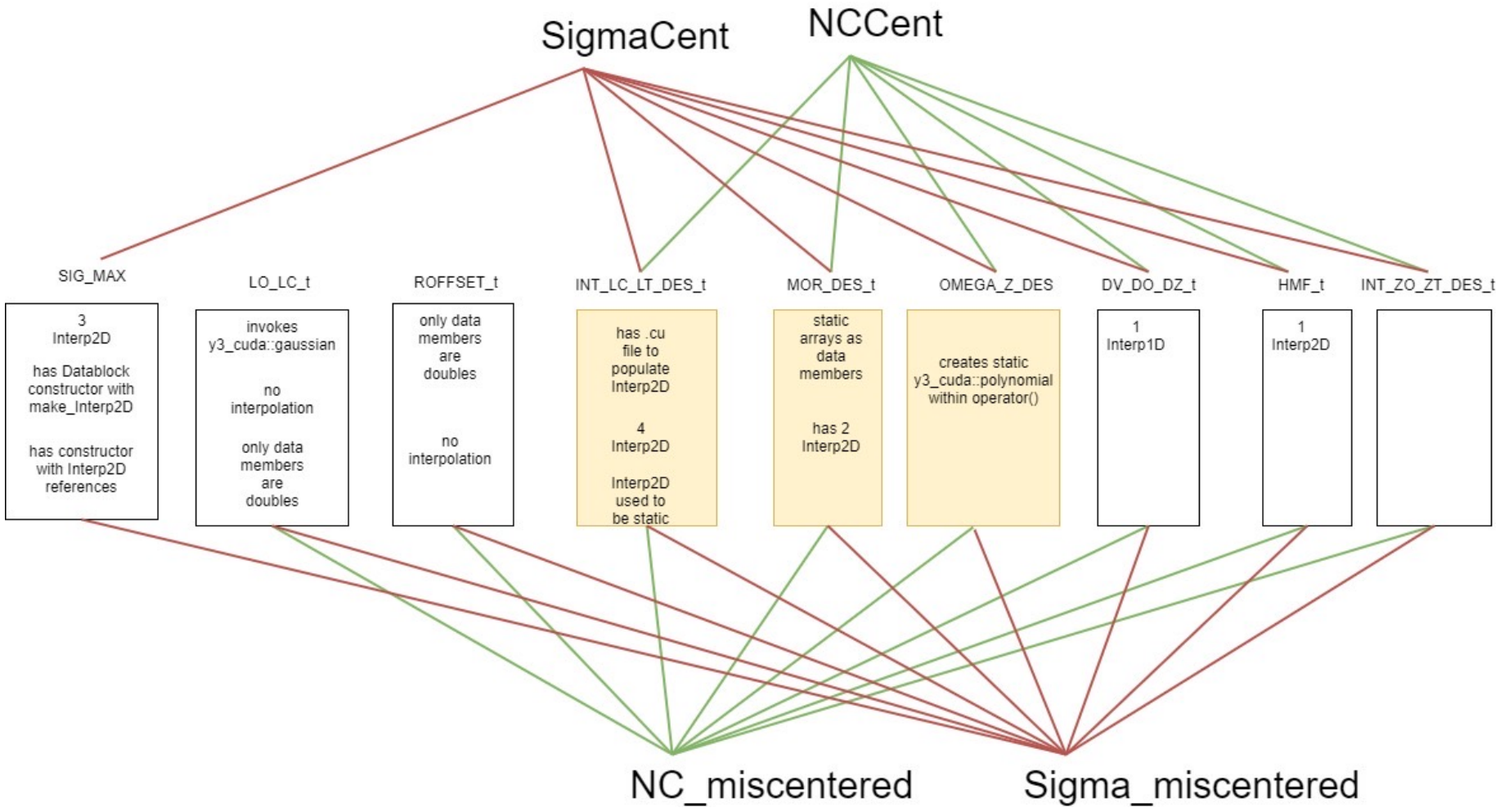


No managed memory

- Crash sooner
- Patterns within modules
- Large initial leak
- No consistency across modules
- NC_cent and Sigma_cent leaked once



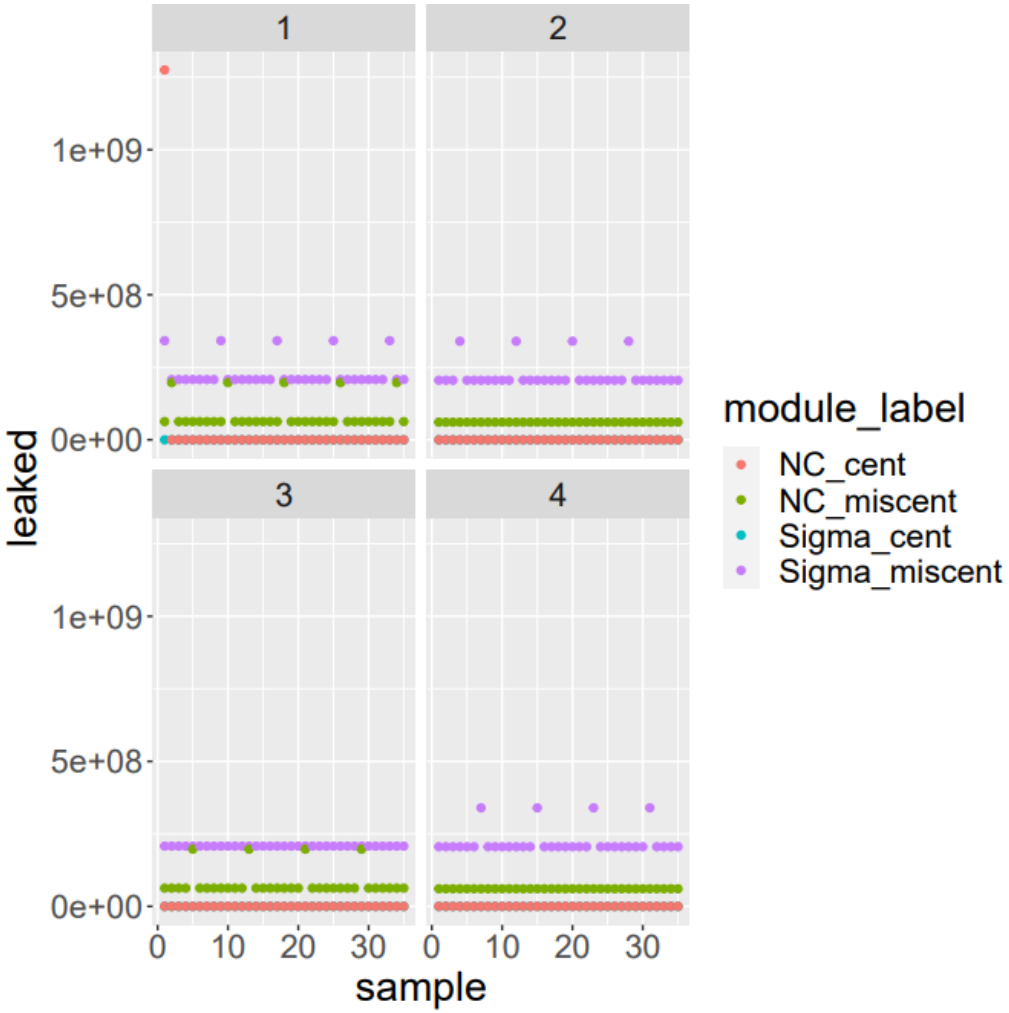
Similarity of integrands



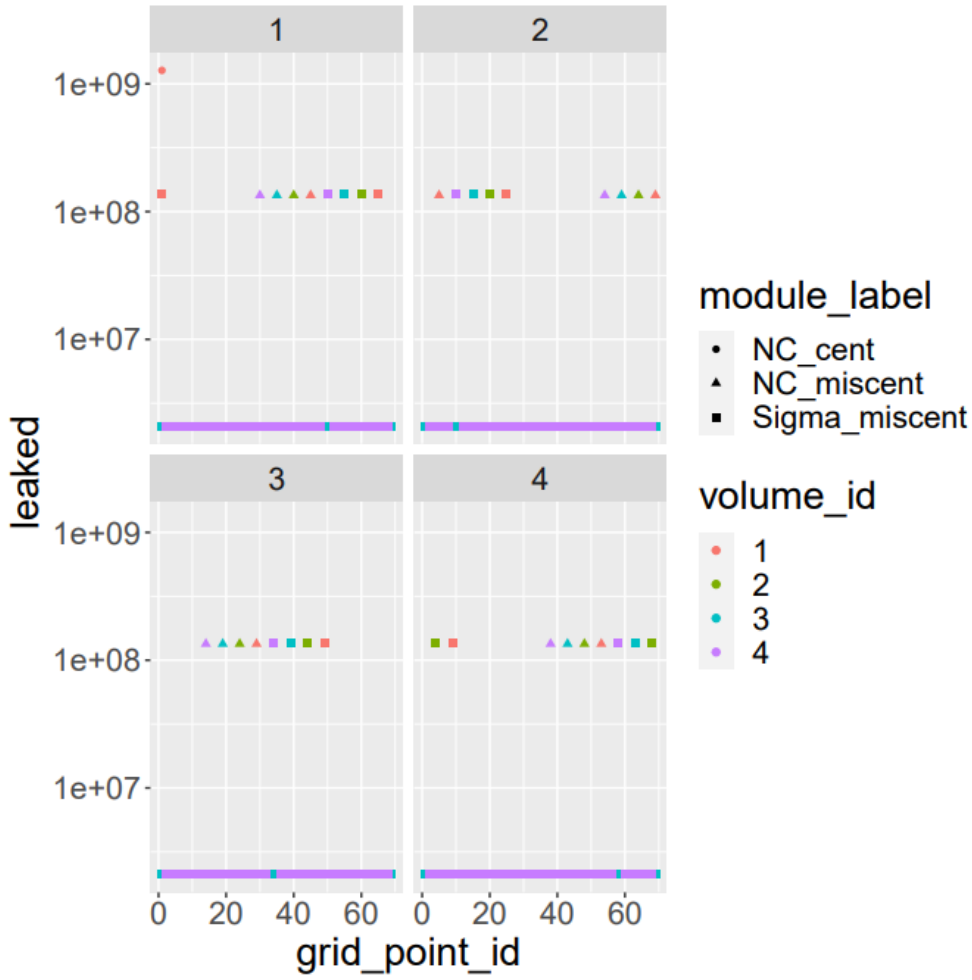
Users of GPU Memory

- PAGANI
  - Quadrature rules
  - Sub-region boundaries
  - Sub-region metadata
- mCUBES
  - Bin boundaries
  - Bin contributions
- Integrands
  - Tabular data
  - Constructed on the CPU
  - Copied to the GPU
- Execute samples
  - 4 integrands/sample
  - 4 volumes per integrand
  - 69 integrations per volume

Volumes

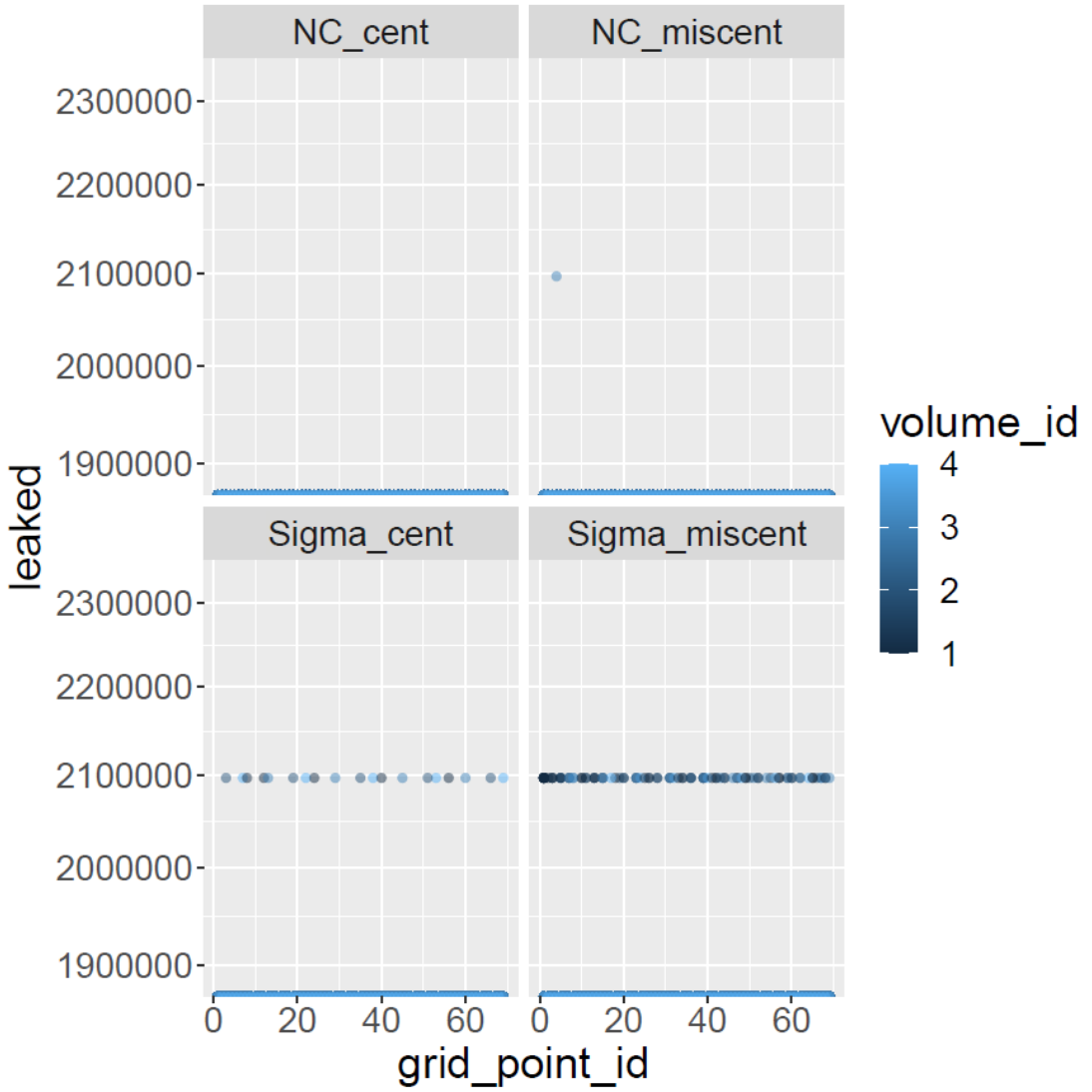


Grid-points



Pipeline without invoking integrate

- No crash
- Smaller leaks
- Integration has a leak
- Additional leak
- Toy program with only integrand copying to gpu memory



Identifying the issue

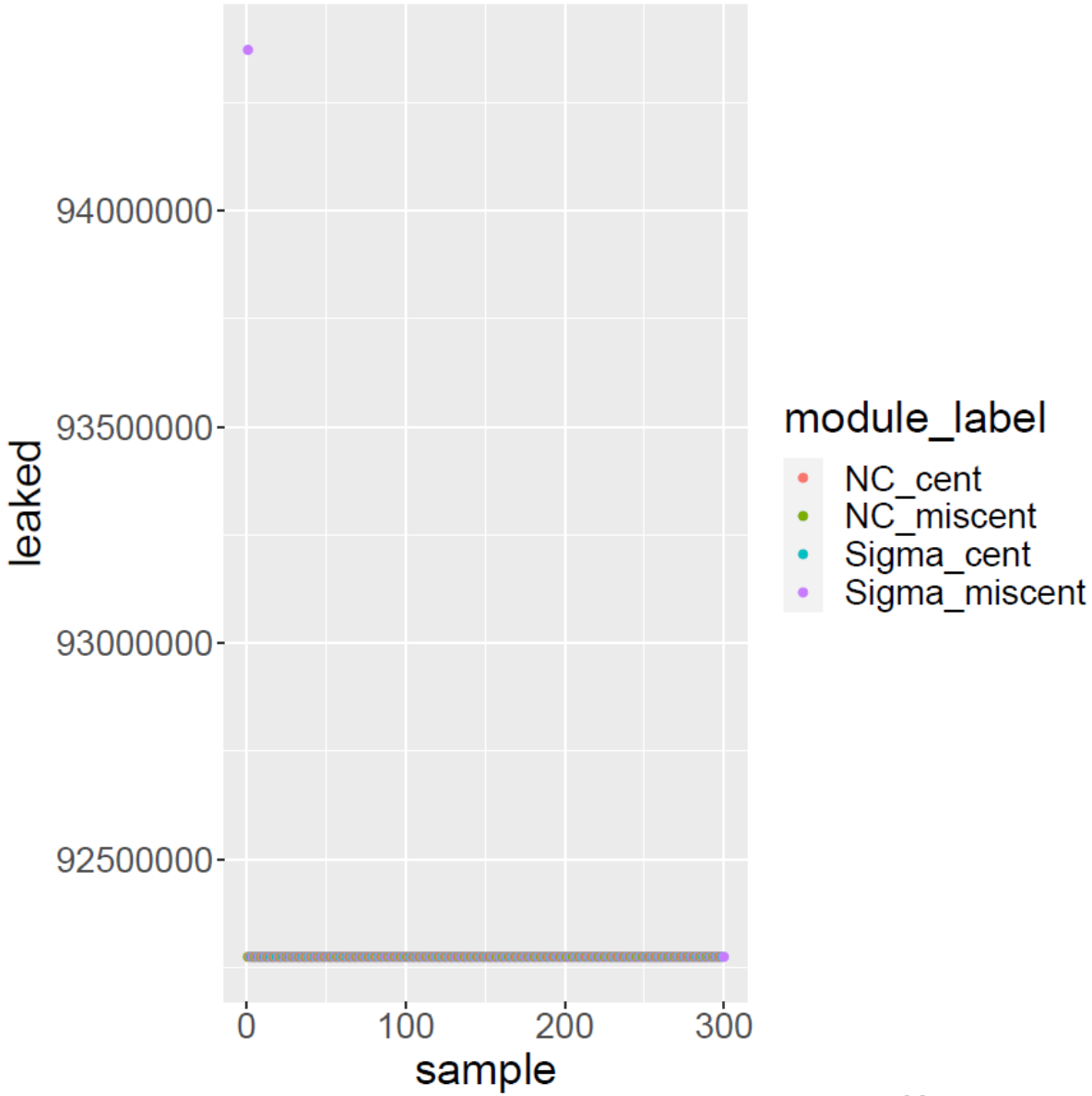
```
template <typename F>
cuhreResult<T>
integrate(F& integrand, ...){
        F* d_integrand = quad::cuda_copy_to_managed(integrand);
        gpu_integrate(d_integrand, ...);
        cudaFree(d_integrand);
}

template <class T>
T*
cuda_copy_to_managed(T const& integrand_on_host){
   T* buffer = cuda_malloc_managed<T>();
   new (buffer) T(integrand_on_host);
   return buffer;
}
```

Identifying the issue

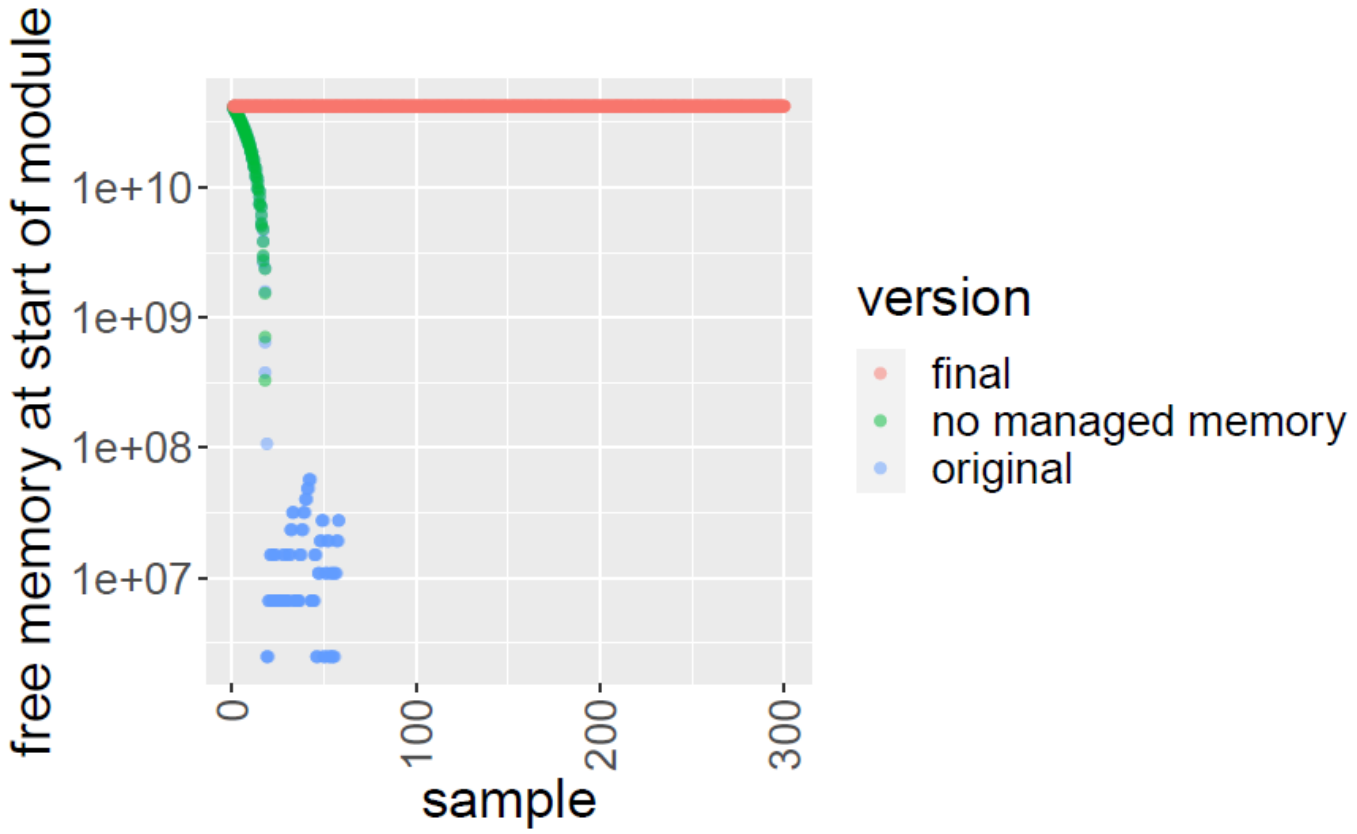
```
template <typename F>
cuhreResult<T>
integrate(F& integrand, ...){
        F* d_integrand = quad::cuda_copy_to_managed(integrand);
        integrate(d_integrand, ...);
        d_integrand->~F();
        cudaFree(d_integrand);
}
```

Not realizing the issue is fixed

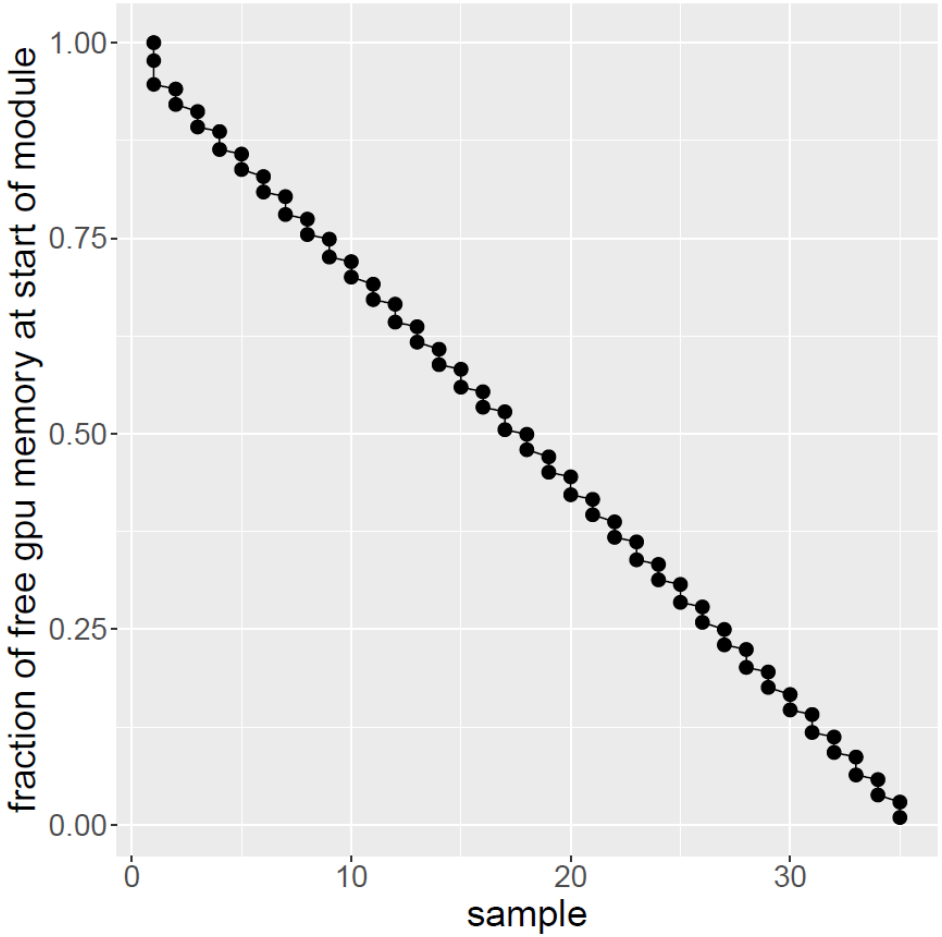
- Repeatedly edited integrands and tested for memory impact
- Executed only PAGANI/m-Cubes
- Amount of reported leaking would exhaust memory

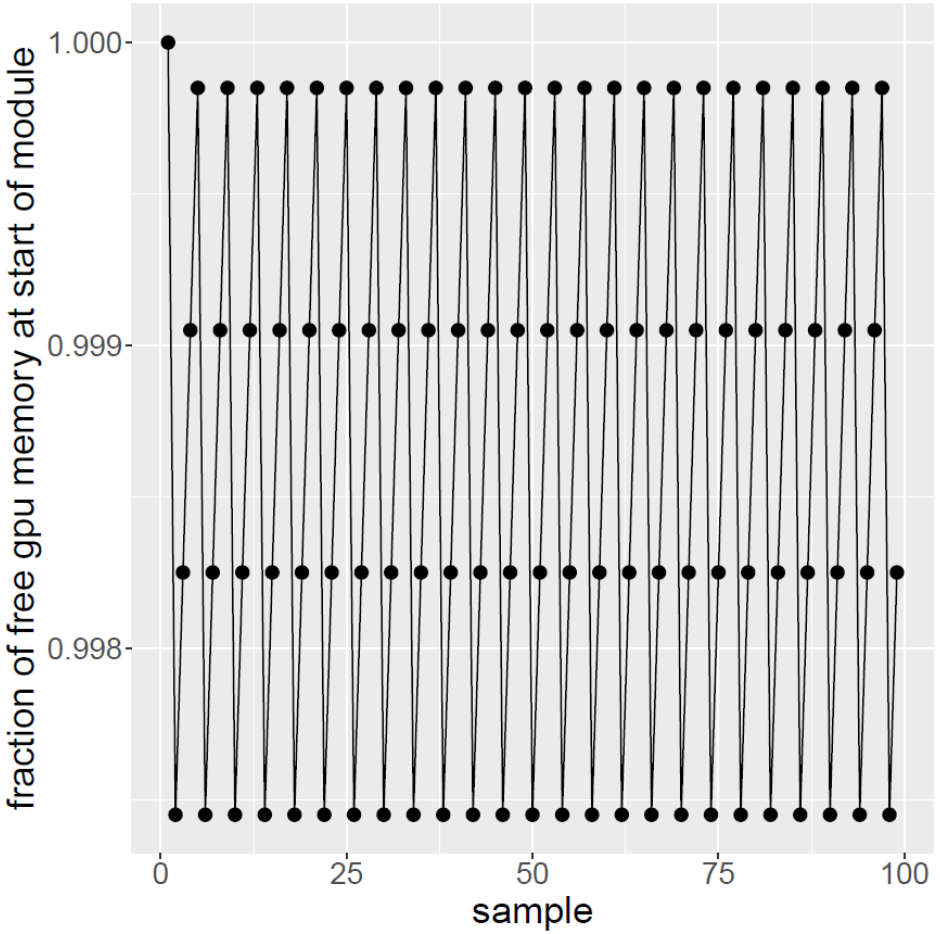


Looking at cudaMemGetInfo at the start



Before and after



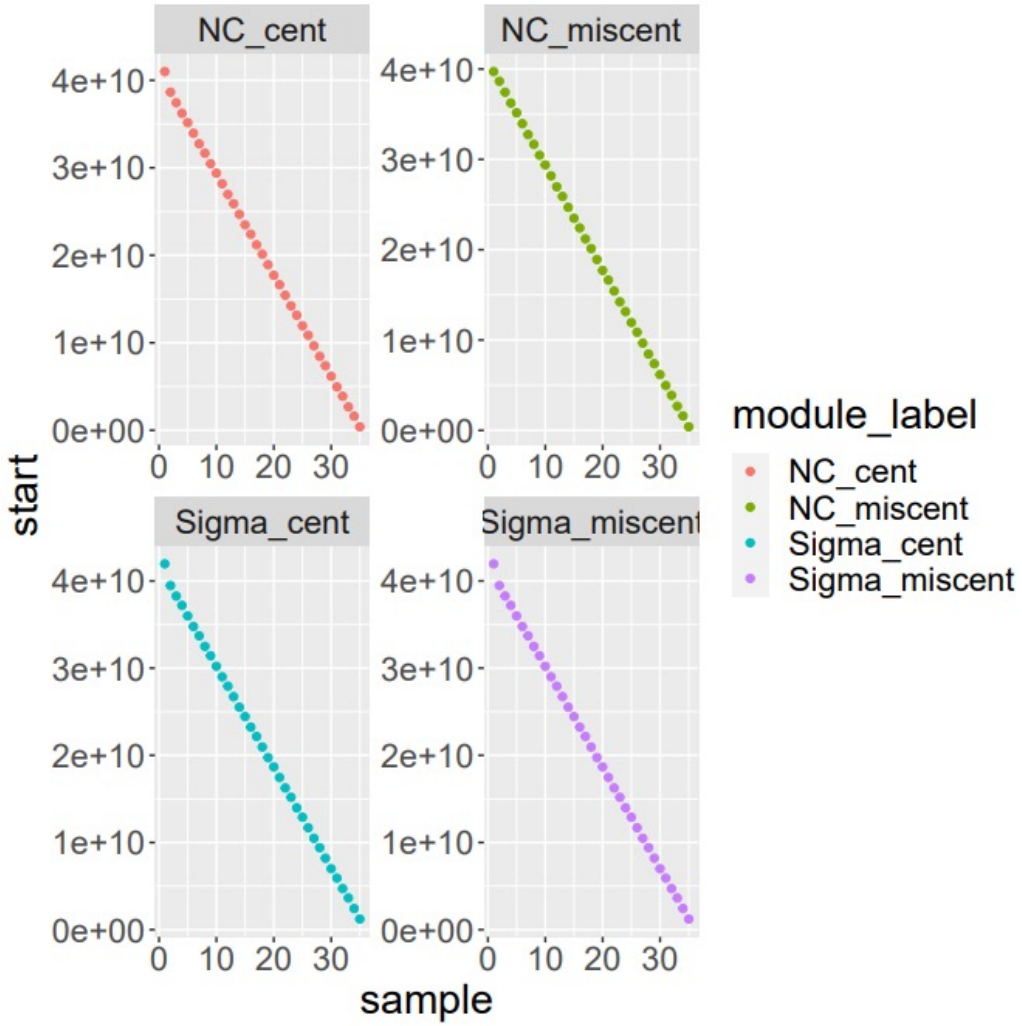


Conclusion

- Insight on cudaMemGetInfo
- Fixed integrands
- Removed unneeded managed memory
- Fixed memory leak
- Successful run with 256 MPI-ranks

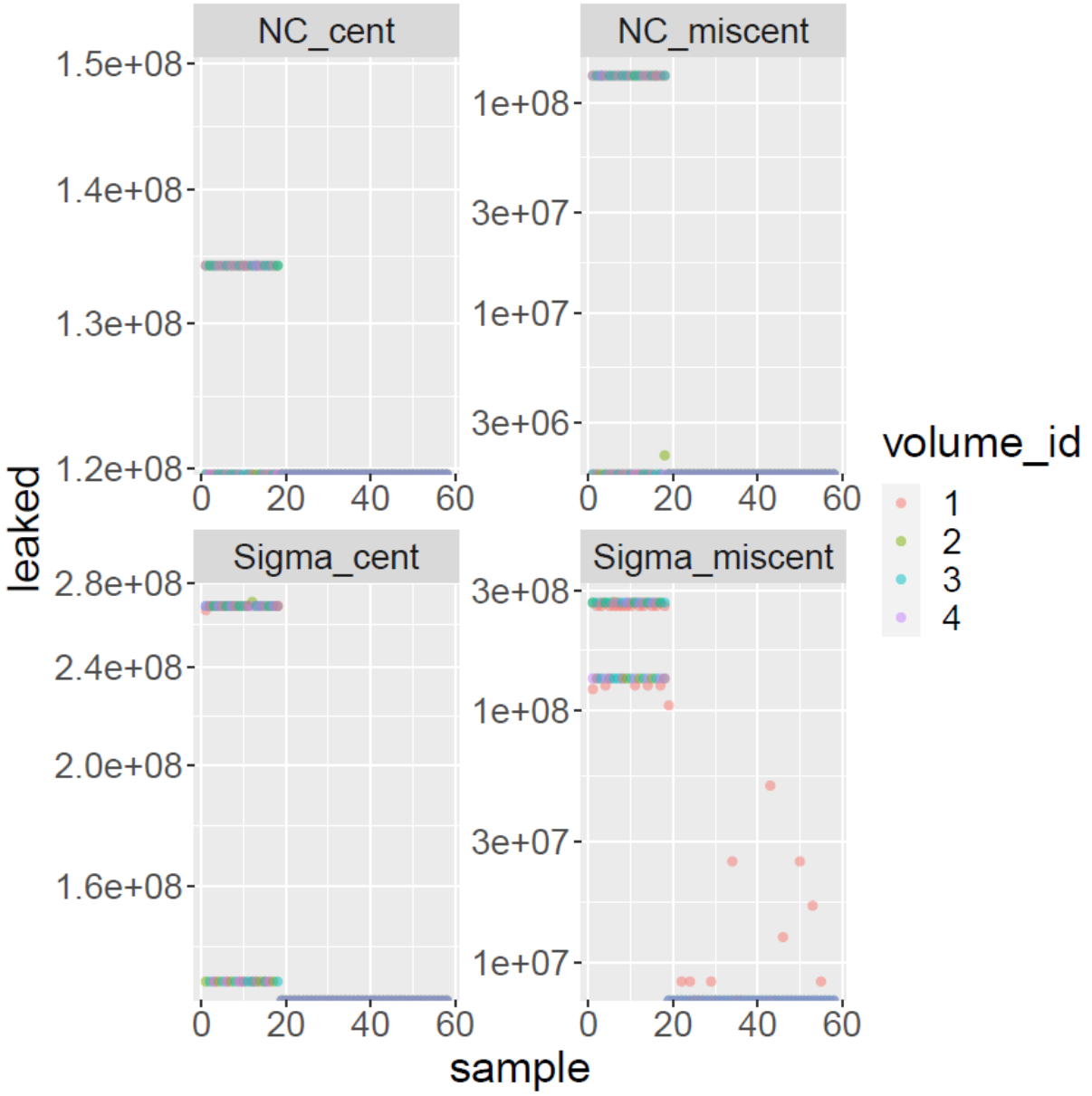
QUESTIONS

Free memory at module start



Managed-Volume level

- Plot not identical to module level
- Volume has no impact on integrand-related allocations



Managed Grid-point level

- Integration level
- Single integration calls
- Multiple leaks?
- Integration must have a leak
- Why not all grid-points?