Investigating memory footprints of data objects

Mateusz Fila

Gaudi Developers Meeting, 17.04.2024

Heterogeneous Frameworks

- R&D project: Heterogeneous Frameworks
- EP-SFT: Mateusz Fila, Benedikt Hegner, Graeme A Stewart

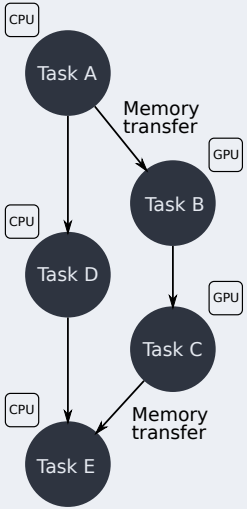
Starting idea:

- Investigate scheduling with accelerators and multi-node setups.
- Start from greenfield
- Mock-up realistic workloads

Why memory footprints?

Beside the execution graph and timings of algorithms a workflow can be characterized by memory sizes of exchanged objects.

- Most of current compute accelerators utilize relatively slow buses
- Communication between nodes is relatively slow
- Cost to bring the data



Object footprint

Different definitions of object memory footprint in use.

Affected by multiple aspects such as:

- type size
- memory layout and alignment
- ownership

Approximate values enough for the use case

```
struct MCHit {
  uint                cellID;
  Vector3             position;
  set<Contribution> contribs;
};

struct Contribution {
  uint      pdg;
  float     deposit;
  Particle* particle;
};
```

Memory footprint measurement

How to obtain information about memory footprints of objects in Gaudi TES?

Collection of ideas, some worked some not:

- Memory profilers
- Serialization
- Allocation statistics
- Specialized counting

Profilers — Massif

Samples the memory usage and creates snapshots

Useful for:

- monitoring overall memory usage
- identifying hot-spots
- detecting unreachable memory leaks

Info

Valgrind heap profiler tool

Limitation

Can’t monitor specific allocations

Profilers — Object introspection

...provides byte accurate memory occupancy information for arbitrarily complex C++ object hierarchies...

Info

New memory profiler by Meta

Limitation

Currently works only with static libraries

Serialization

- Measure the size of buffer
- Utilize serialization mechanism to obtain relevant information

ROOT:

- Serialize each data object to *`TBufferFile`*
- Custom *`TBuffer`* to avoid unnecessary operations
- Serialization of whole TES used in GaudiMP

Limitation

Include serialization format overhead

I failed to extract the information with this method. *`AnyDataWrapper`* serialized without content, error on my side

Process memory usage

Parsing */proc/* implemented in Gaudi with *ProcStats*.

Reports blocks of memory available to a process. Insensitive to:

- allocations fitting in available memory
- dealocations not causing release of memory back to system

Limitation

Provides only coarse, high level information

Allocator monitoring

Replace the allocator with a one that tracks the number of bytes it allocates and deallocates

- adjust the std containers used in a data model
- change custom containers to defer allocation to an allocator

More possibilities for allocations done with *`new`* or *`malloc`*.

Flaw

Intrusive unless a data model already utilizes dynamic allocator

Flaw

Measures whole memory pre-alocated to a container not the part in use (capacity not size)

Malloc monitoring

Not part of the C and C++ standards

glibc defines following functions for monitoring *`malloc`*:

- *`mallinfo mallinfo()`* - deprecated
- *`mallinfo2 mallinfo2()`*
- *`int malloc_info(int options, FILE *fp)`* - output XML
- *`int malloc_stats()`* - print statistics to stdout

Malloc monitoring

Drawbacks:

- Change in memory allocation required
- Original object creation hard to pin-point
- Deep-copy or delete object from TES
- Deep-copy polymorphic objects is tricky
- Deleting relates on correct ownership semantics

Mechanism scheme:

- Single thread, single event slot
- Wait for ‘EndEvent‘
- For each object in TES:
  - get free memory
  - delete object
  - get free memory difference

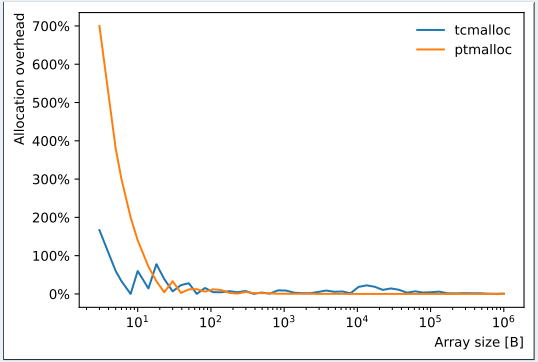
Flaw

Assumes object ownership

Malloc monitoring results

Relative success:

- glibc malloc (ptmalloc) not reporting freed but awaiting dealocations
- wrote malloc wrapper to include such allocations
- tcmalloc reports freed memory as expected
- wrote malloc wrapper to monitor requested memory (no memory overheads)



Example output

id	class	size [B]
/Event/	DataObject	0
/Event/SomeInt	AnyDataWrapper<int>	8
/Event/SomeDouble	AnyDataWrapper<double>	8
/Event/SomeVec4	AnyDataWrapper<MicroVector4>	32
/Event/SomeVec3	AnyDataWrapper<MicroVector3>	24
/Event/SomeFloat	AnyDataWrapper<float>	8

Specializations

Assumptions done by previous approaches might be not fulfilled by every data model

Provide code to calculate memory footprint for each class

Flaw

- Maintenance effort
- Only supported classes can be used

EDM4hep specific

EDM4hep data model used in key4hep:

- object data stored in PODs
- objects stored in collections
- collections owned by *`Frame`*
- only collections stored in TES

Collections can't be destroyed directly from TES

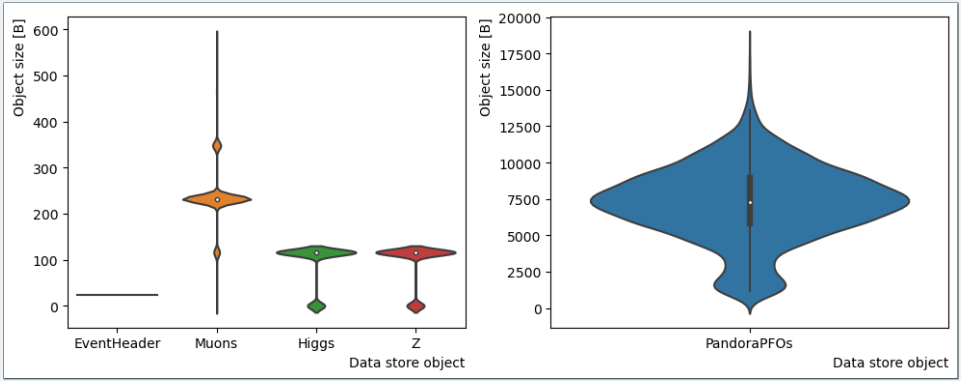
Measuring with malloc will lead do segmentation faults

Since all the allowed types are known, collection *`coll`* memory footprint can be calculated directly:

`coll.size()*sizeof(coll::value_type)`

EDM4hep results

Example memory footprint distributions



}