TECHNISCHE UNIVERSITÄT DRESDEN

Dresden Database Systems Group

Internship Memory Management

Kick-Off Meeting, 03.11.2020

Participants

Dresden Database Systems Group

Tutors

Prof. Dr.-Ing. habil. Dirk Habich

- Database Systems on modern hardware
- “Everything”
- dirk.habich@tu-dresden.de

Dipl.-Inf. Johannes Pietrzyk

- Query processing on vectorized HW
- Single-Threaded optimization
- johannes.pietrzyk@tu-dresden.de

Dipl.-Inf. Johannes Fett

- Query processing on GPUs
- Unification of heterogeneous HW
- johannes.fett@tu-dresden.de

Dr.-Ing. Alexander Krause

- Graph-Systems on scale-up systems
- NUMA-aware storage engines
- alexander.krause@tu-dresden.de

Topics

- DRAM + HBM
- NUMA
- GPU

Organizational Matters

- Start: 03.11.2020
- End: 05.02.2021
- Syncs: Tuesday 09:20 (if scheduled)

Memory Management – A solved problem?

Dresden Database Systems Group

On the road so far…

- Memory management takes care of available virtual memory and provides access to it
- Over the time multiple libraries were introduced, which enable easy-to-use memory acquisition and releasing
  - (s)brk, mmap/munmap, **malloc/free**, **new/delete**, …
- Most of the libraries encapsulate it's own "memory management"
  - Keep track of allocated memory
  - Identify free regions to serve memory requests
  - (Avoid fragmentation)
- All of this solutions are usable with "zero-effort"

So what is the problem?

- Most of the libraries are for general purpose, but DB-systems have excessive knowledge about memory access patterns → memory management can be realized more efficient
- New hardware (HBM, NUMA-HW, GPUs, NVRAM) introduces their own libraries accompanied with dedicated functions

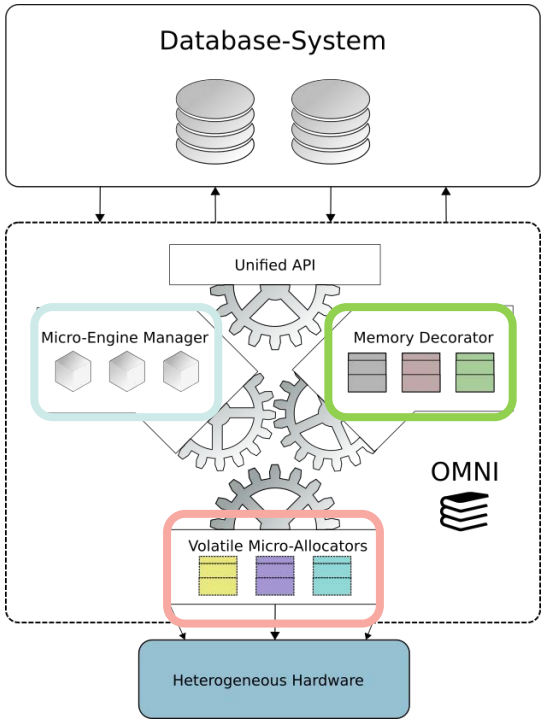
Goal of this internship

Design and implement a unified, easy-to-use library, which enables hardware-oblivious memory management

General Architecture

Dresden Database Systems Group

Concept



ME-Managers

- Actual memory management components
- Stateful entities which live as long as the system lives
- Possible Manager could be:
  - Arena Manager (for columns)
  - Slab Manager (for pointers to columns,...)

Allocators

- actually obtain / release memory
- no management code
- static functions (stateless)
- use dedicated specs to address special needs of specific hardware

Decorators

- Functions (or functors) which are used by the micro-engine managers to modify a specific memory region
- Possible use-cases could be:
  - Alignment of memory
  - Debug information like LOC of allocation
  - Fences
  - ...

General procedure

Dresden Database Systems Group

Deliverables

D1	Prerequisites

- Project Setup
- Build system setup

D2	Allocators Design/Implementation (D/I)

- Design of unified allocator interface
- Implementation of group-specific allocator

D3	Decorators D/I

- Design of unified decorator interface
- Implementation of group-specific decorators

D4	Micro-Engine Memory Manager D/I

- Design of unified memory manager interface
- Implementation of group-specific manager

D5	Wrap Up and Presentation

Schedule

- Every deliverable (except for D5) deadline includes a **short** group-wise presentation followed by discussions
- After the discussion, the next D* is introduced by the tutors
- One week after the start of every D*, a joint discussion is scheduled

Deliverable	Deadline
D1	10.11.2020
D2	
D3	
D4	
D5	

General requirements

Dresden Database Systems Group

Deliverables:

- **Exhaustive** code documentation
- Licence informations in every file
- Meet the code style requirements
- Suitable c-tests for **every** (relevant) procedure
- Suitable micro-benchmarks using google benchmark framework
- Merge-Request after a deliverable is completed

Short-presentations:

- Use DB presentation template
- Send over to your tutor via email **before** the actual assembly

Final presentation:

- The final presentation is a joint work of all students
- All of you decide, who presents the general introduction and interface specifications
- Every group presents their own contribution and selects one who shows their results
- The individual presentations need to include necessary future work and possible improvements
- ~30 min in total

C++ Coding Style

Dresden Database Systems Group

Adhere to general conventions

- Include guards – NO #pragma once
- Opening curly braces on the same line as the function / statement
- Whitespace between Type and asterisk for pointers
- TAB or 4 whitespaces for indendation

```cpp
//filename.h
/*
 * LICENCE here
 */
#ifndef PROJECTNAME_RELATIVEPATHWITHUNDERSCROES_FILENAME_H
#define PROJECTNAME_RELATIVEPATHWITHUNDERSCROES_FILENAME_H

#include <iostream>

class my_class : public you_class {
    //Member functions
    public:
        /**
         * @brief Brief description of method.
         * @details Detailed description of method (WHAT happens inside).
         * Usage:
         * @code
         * Insert code example of usage here
         * @endcode
         * @tparam T Description of template parameter.
         * @param p_out_buffer Parameter1 description.
         * @param p_values_buffer Parameter2 description.
         * @param p_to_add_buffer Parameter3 description.
         * @param p_count_elements Parameter4 description.
         * @return Description of return value.
         */
        template< typename T >
        T * add_data(
            T * const __restrict__ p_out_buffer,
            T * const __restrict__ p_values_buffer,
            T * const __restrict__ p_to_add_buffer,
            std::size_t const p_count_elements
        );
        void say_hello( void ) const {
            std::cout << "Hello from MyClass.\n";
        }
    //Fields
    private:
        /**
         * @brief Brief description of member.
         */
        bool verbose = true;
    //Ctors / Assignment / Dtor
    public:
        MyClass( ) = default;
};
#endif //PROJECTNAME_RELATIVEPATHWITHUNDERSCROES_FILENAME_H
```

D1: Prerequisites

Project Setup

Dresden Database Systems Group

General setup:

- c++17
- Minimum cmake version 3.18
- ctest
- Internal Git project with branch for every group and deliverable
- Licence: Gnu GPL v3

Project requirements:

- automated detection of hardware features
- stand-alone libraries for every hardware
- Integration of hardware specific libraries