Memkind library

WW12'19

Michal Biesek

michal.biesek@intel.com

Agenda

Memkind:

- Overview
- API
- Detecting kind
- Examples
- Memory fragmentation
- Summary

Memkind overview

Memkind overview

- Based on concept of kind which representing type of memory
- Written in ISO C to be compatible with most of middleware popular in HPC community
- Built on the top of popular allocator – jemalloc (TBB allocator is partially supported by memkind)
- Available on GitHub https://github.com/memkind
- Supported by Linux (available on distro: Fedora, Red Hat…)
- Binaries for latest release are available: http://memkind.github.io/memkind/
- Memkind API is much similar to stdlib API
- Support Intel® Optane™ DC Persistent Memory in App Direct Mode volatile variant

Memkind API Kinds

- Kind is representation for independent memory pool structure (**jemalloc** arenas)
- **Static memory kinds** – predefined kinds which operates on DRAM, offers different characteristics, e.g. :

MEMKIND_DEFAULT – allocation using standard memory and default page size

MEMKIND_HBW – allocation from the closest high bandwidth memory NUMA node at time of allocation

- **Pmem kind** – dynamically kind which supports Intel® Optane™ DC Persistent Memory

Jemalloc overview

Arena - data structure to enable parallelism (each thread has it's own arena)

User objects are divided to two categories: small and large class

Small class - size < 16kb – managed in groups called slabs

Large class - size ≥ 16kb

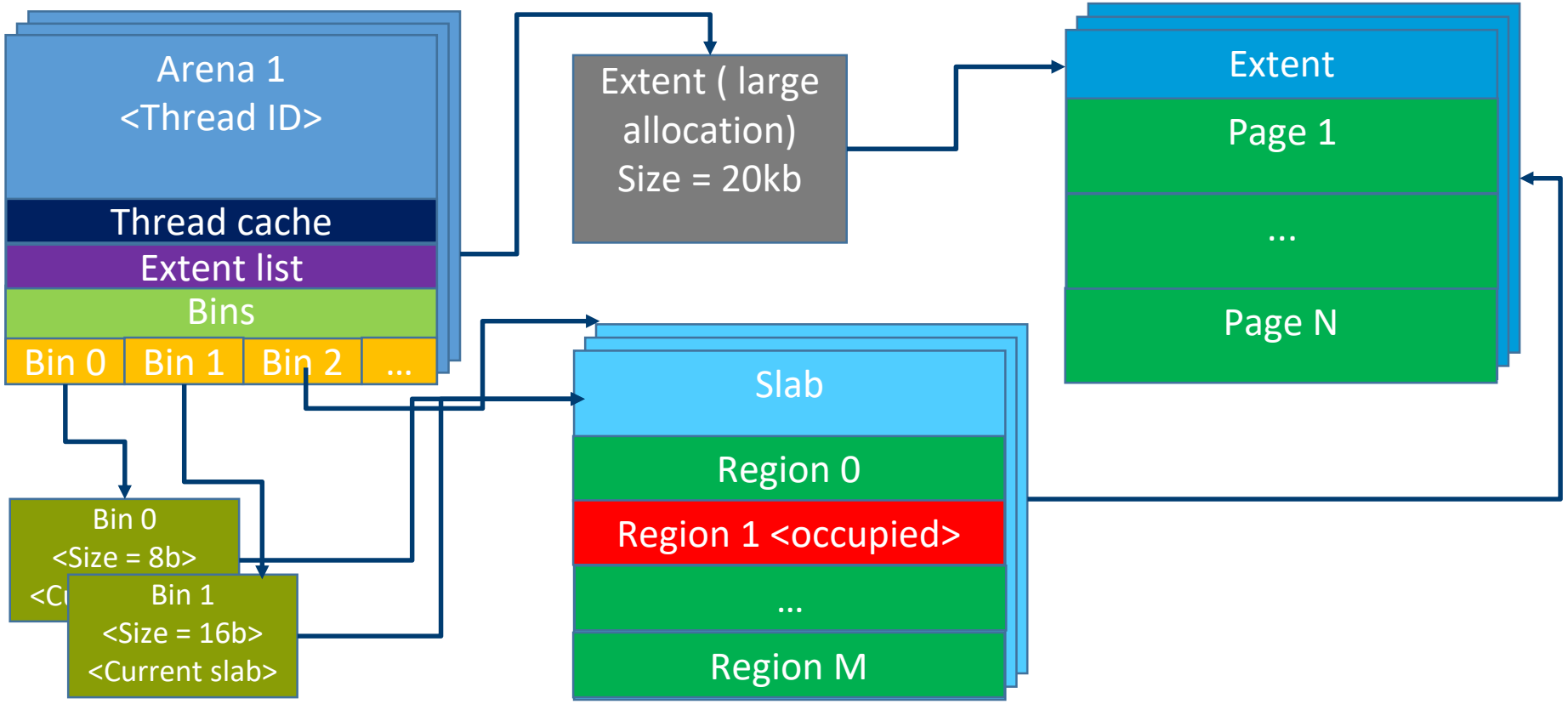
Extents - virtual memory areas that available memory is conceptually divided into (always aligned to multiples of the page size (4kb). Extent can be used to allocate large class or to allocate multiple small class

Slab - when the extent used to allocate small_class. Slab maintain a bitmap to track which regions are in use

Bin - management used in slab, each bin correspond to size class

Thread cache - a cache unique to each thread for allocating small memory

Jemalloc 5.0 simplified overview



Memkind – jemalloc interaction

- Memkind 1.8 is based on customized jemalloc 5.0
- Jemalloc provide **extent management hooks** functionality as form of interface to heap memory management allocator
- Heap of kinds from memkind is internally managed by jemalloc
- Jemalloc provides **mallctl** interface for introspecting memory allocator, setting modifiable parameters and triggering actions

Memkind – jemalloc interaction

Memkind API	Jemalloc API
memkind_create_pmem	arena.create
memkind_destroy_kind	arena.destroy
memkind_detect_kind	arenas.lookup
memkind_create_pmem_with_config	arena.dirty_decay_ms

Memkind API

Memkind API kind configuration interface

```
struct memkind_config *memkind_config_new()

void memkind_config_delete(struct memkind_config *cfg )

void memkind_config_set_path(struct memkind_config *cfg, const char *pmem_dir );

void memkind_config_set_size(struct memkind_config *cfg, size_t pmem_size );

void memkind_config_set_memory_usage_policy(struct memkind_config *cfg,
memkind_mem_usage_policy policy )
```

Memkind API heap management

```
void *memkind_malloc(memkind_t kind, size_t size )

void *memkind_calloc(memkind_t kind, size_t num, size_t size )

void *memkind_realloc(memkind_t kind, void *ptr, size_t size )

void memkind_free(memkind_t kind, void *ptr )

size_t memkind_malloc_usable_size(memkind_t kind, void *ptr )

memkind_t memkind_detect_kind(void *ptr )
```

Memkind API PMEM

```
int memkind_create_pmem(const char  *dir , size_t  max_size , memkind_t  *kind )

int memkind_create_pmem_with_config(struct memkind_config  *cfg , memkind_t  *kind )

int memkind_destroy_kind(memkind_t  kind )
```

Note:

The pmem space limitation is configurable option could be limited by user by providing max_size or by space available on filesystem max_size = 0.

Memkind detecting kind

Memkind API detecting kind functionality

Besides function memkind_t memkind_detect_kind(void *ptr)

Passing NULL value as kind to functions like:

- memkind_malloc_usable_size(NULL, ptr)
- memkind_free(NULL, ptr)
- memkind_realloc(NULL, size, ptr)

Results in detecting kind inside memkind library. In other words this is this “One API to rule them all” allows to allocate to specific destination, with no need to remember from where deallocate or from where reallocate.

Note:

Depending of usage and workload it could result with non-trivial overhead.

Memkind examples

Example - allocation to DRAM and PMEM

- Showing using libmemkind to allocating to DRAM and PMEM
- Freeing memory by explicitly pass correct value of kind
- Example shows usage of functions like:
- memkind_create_pmem(path, size, kind)
- memkind_malloc (kind, ptr)
- memkind_free(kind, ptr)
- memkind_destroy_kind(kind)

Example - allocation to DRAM and PMEM

```
int main(int argc, char *argv[])
{
        struct memkind *pmem_kind = NULL;
        memkind_create_pmem("/mnt/pmem", 0, &pmem_kind);
        char * ptr_default = (char *)memkind_malloc(MEMKIND_DEFAULT, size);   //allocate in DRAM
        char * ptr_pmem = (char *)memkind_malloc(pmem_kind, size);           //allocate in PMEM
        memkind_free(MEMKIND_DEFAULT, ptr_default);
        memkind_free(pmem_kind, ptr_pmem);
        memkind_destroy_kind(pmem_kind);
        return 0;
}
```

Example - detecting kind

- Showing using libmemkind to create kind with custom configuration
- Freeing memory with usage of internal look up for correct kind inside libmemkind
- Example shows usage of functions like:
- memkind_config_new()
- memkind_config_set...(cfg,param)
- memkind_config_delete(cfg)
- memkind_free(NULL, ptr)

Example – detecting kind

```
void *alloc_buffer_g[10000]; //global allocation buffer
int main(int argc, char *argv[])
{
        struct memkind_config *pmem_cfg = memkind_config_new();
        memkind_config_set_path(pmem_cfg, "/mnt/pmem/");
        memkind_config_set_size(pmem_cfg, 1024 * 1024 * 64);
        memkind_config_set_memory_usage_policy(pmem_cfg,
        MEMKIND_MEM_USAGE_POLICY_CONSERVATIVE);
        memkind_create_pmem_with_config(pmem_cfg, &pmem_kind);
        //...allocation in global buffer from different kinds
        DeallocateGlobalbuffer ();
        memkind_config_delete(pmem_cfg);
        memkind_destroy_kind(pmem_kind);
        return 0;
}
```

```
static void DeallocateGlobalbuffer()
{
        for(int i = 0; i < 10000; i++)
        {
                memkind_free(NULL, alloc_buffer_g[i]);
        }
}
```

Memkind memory fragmentation

Memkind memory usage policy

- Memory fragmentation problem results in decreasing capacity of memory kind
- API to reduce memory consumption in form of memory usage policy:

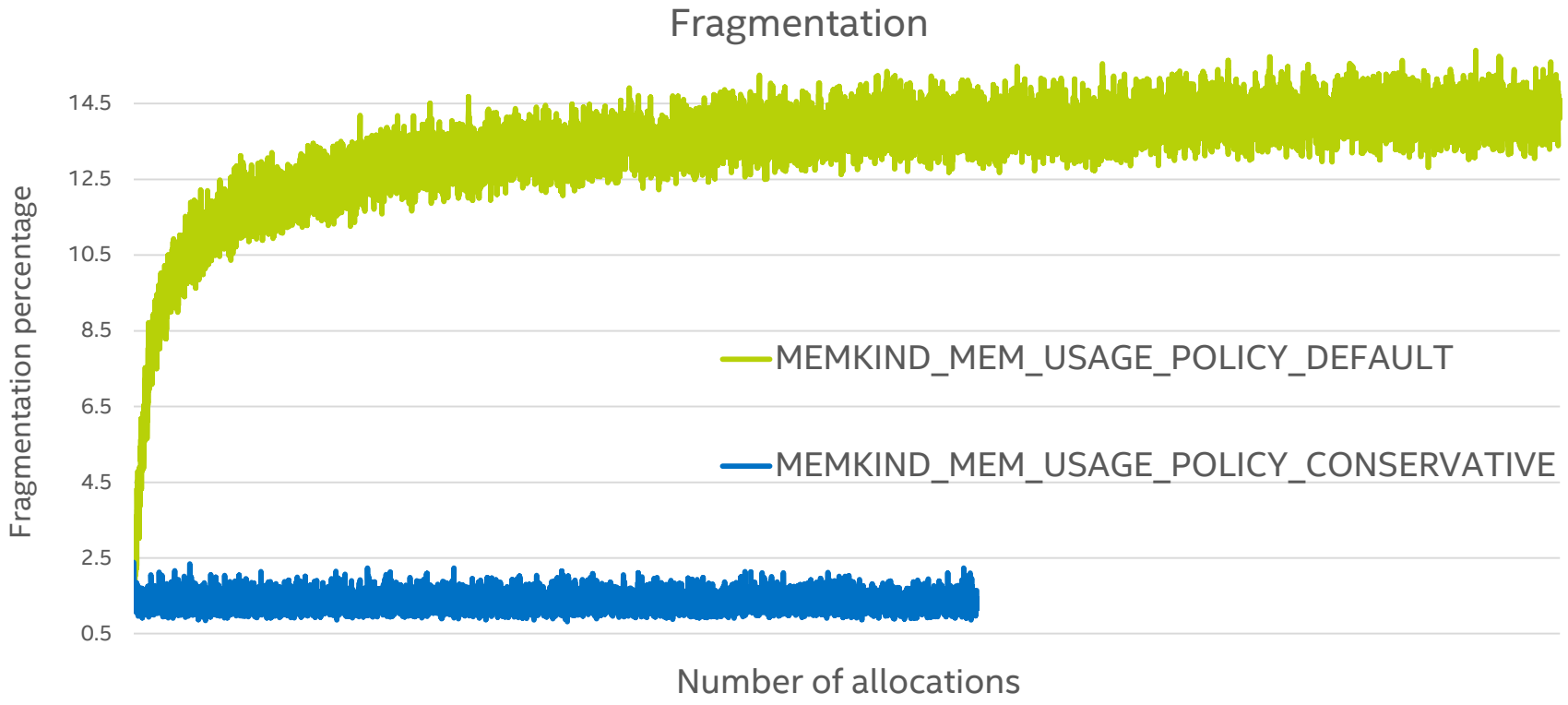
MEMKIND_MEM_USAGE_POLICY_DEFAULT – default memory usage policy

MEMKIND_MEM_USAGE_POLICY_CONSERVATIVE – prioritize memory usage at cost of performance

Note:

Setting memory usage policy is always associated with trade off between memory usage and CPU utilization.

Fragmentation



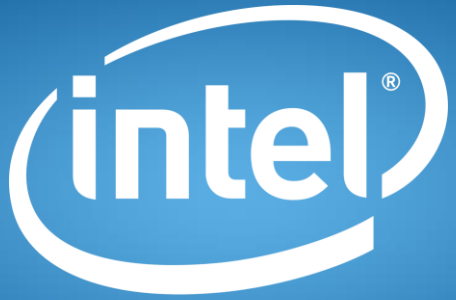
Memkind and multithreaded applications

- Another form of tuning the application for reducing memory overhead is to reduce number of arenas associated to the kind.
- Default number of arenas associated to kind is equal 4 times numbers of CPU
- Setting environment variable MEMKIND_ARENA_NUM_PER_KIND to lower value results with decrease memory fragmentation

Memkind summary

Memkind summary advantages

- Provide a unified API to management PMEM and DRAM
- User friendly functions like **free/realloc/usable_size** result to easier integration with already existing application
- API allows to customize characteristic of allocations depending of application usage



experience
what's inside™