# LIBPMEMOBJ-CPP WORKSHOPS

Speaker: Szymon Romik (Intel Data Center Group)

<szymon.romik@intel.com>

May, 2019

# Agenda

## Remind API

## Persistent Memory Programming

- Warmup example (persistent counter)
- Finding bugs related to persistent memory programming
- Converting volatile queue to persitent one
- Hashmap example
- Processing data on persistent memory using map reduce

# Using libpmemobj-cpp

## Introduction and documentation:

- http://pmem.io/pmdk/cpp_obj/

## C++ containers

- http://pmem.io/2018/11/02/cpp-array.html
- http://pmem.io/2019/02/20/cpp-vector.html
- More containers under development

## Libpmemobj manpages:

- http://pmem.io/pmdk/manpages/linux/master/libpmemobj/libpmemobj.7.html

# libpmemobj – what you will need?

**PMEMobjpool** *pmemobj_open(**const char** *path, **const char** *layout);

**void** pmemobj_close(**PMEMobjpool** *pop);

**PMEMoid** pmemobj_root(**PMEMobjpool** *pop, **size_t** size);

**int** pmemobj_tx_add_range(**PMEMoid** oid, **uint64_t** off, **size_t** size);

**int** pmemobj_tx_add_range_direct(**const void** *ptr, **size_t** size);

**PMEMoid** pmemobj_tx_alloc(**size_t** size, **uint64_t** type_num);

**int** pmemobj_tx_free(**PMEMoid** oid);

**void** *pmemobj_direct(**PMEMoid** oid);

TX_BEGIN(**PMEMobjpool** *pop) / TX_END

OID_NULL, OID_IS_NULL(PMEMoid oid)

# libpmemobj-cpp – what you will need?

```
pool<T> pool<T>::open(const std::string &path, const std::string &layout)

peristent_ptr<T> pool<T>::root()

void transaction::run(pool_base& pool, std::function<void()> tx, ...)

peristent_ptr<T> pmem::obj::make_persistent<T>(Args &&... args)

void pmem::obj::delete_persistent<T>(peristent_ptr<T> &ptr)
```

Types: p<T>, peristent_ptr<T>

# Getting started

## Get workshops repo:

```
$ git clone http://github.com/pmemhackathon/2019-05
$ cd
```

## How to compile examples?

- Simply run:

```
$ make
```

- Export your user login in order not to modify commands (you can just copy them)

```
$ export USER=pmdkuser{1..100}
```

# Persistent counter – what you should do

## Change warmup.cpp to print bigger number every time you run it

- Add variable (a counter) to root struct
- Increment the variable inside „inc” method
- Return new value

## Expected result:

```
$ pmempool create obj --layout=warmup -s 100M /mnt/pmem-fsdax0/${USER}/warmup
$ ./warmup /mnt/pmem-fsdax0/${USER}/warmup
1
$ ./warmup /mnt/pmem-fsdax0/${USER}/warmup
2
```

# Pmemcheck – persistent memory error detector

Checks for non-persistent stores

Checks for overwrites

Checks for stores made outside of a transaction

Checks for snapshotting the same object in two different threads

Can be found here: https://github.com/pmem/valgrind

# Pmemcheck – installation and usage

## Installation (Already installed on your machine)

```
$ git clone https://github.com/pmem/valgrind
$ cd valgrind
$ ./autogen.sh
$ ./configure [--prefix=/where/to/install]
$ make install
```

## Usage

```
$ valgrind –-tool=pmemcheck [valgrind options] <your_app> [your_app options]
```

# Find bugs – what you should do

Run:

```
$ pmempool create obj --layout=find_bugs -s 100M /mnt/pmem-fsdax0/${USER}/find_bugs
$ valgrind --tool=pmemcheck ./find_bugs /mnt/pmem-fsdax0/${USER}/find_bugs
```

Fix bugs reported by valgrind and run valgrind again

# Queue – what you should do

Implement a persistent version of queue

It should be based on volatile queue (modify queue.cpp file)

Usage for volatile version:

```
$ ./queue
$ push 1
$ pop
```

Usage for persistent version

```
$ pmempool create obj --layout=queue -s 100M /mnt/pmem-fsdax0/${USER}/queue
$ ./queue /mnt/pmem-fsdax0/${USER}/queue
$ push 1
$ show
```

# Converting queue step-by-step

1. Open a pool using a **path** variable and supply „queue” as layout.
2. Obtain pointer to the root object
3. Change volatile pointers to persistent ones
4. Change memory allocations/deletions
5. Add transactions

# Hashmap

## Implement a hashmap with following interface:

- at(key)
- Insert(key, value). If the key already exists, overwrite old value with new one
- Assume that hashmap user is calling insert method inside transaction

## Do following:

- Use std::hash specialization for pmem::obj::experimental::string (which is already defined)
- Store buckets for a hashmap in pmem::obj::experimental::array container
- For buckets you can use pmem::obj::experimental::vector with elements containing Key and Value

# Hashmap

## To check if it works, compile and run:

```
$ ./simplekv_simple path_to_pool
```

# Data oriented design

The approach is to focus on the data layout, separating and sorting fields according to when they are needed.

In general – allows for better utilization of CPU cache

For persistent memory – allows for optimized snapshotting

## Try to optimize your hashmap layout

# Hashmap

## Modify your hashmap to make insert() method atomic for user (use transaction inside insert implementation)

- libpmemobj function:

  PMEMoid pmemobj_pool_by_ptr(const void* addr)

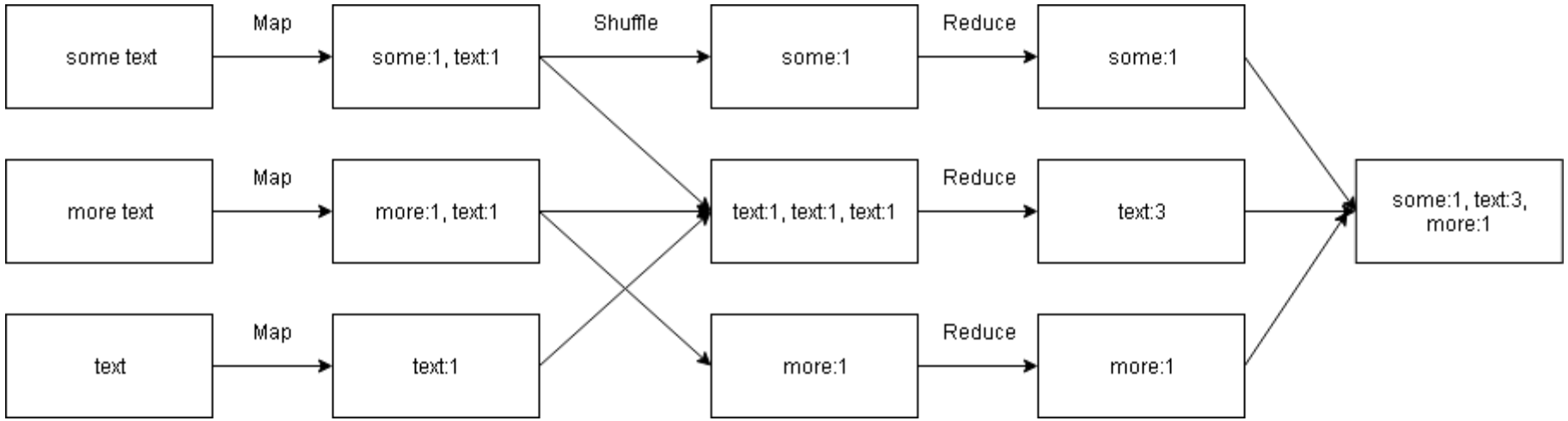
  might be useful

# Map reduce

Programming model for processing and generating big data sets

Consists of Map, Reduce and Shuffle steps

- Map – performs filtering, transformation or sorting
- Shuffle – redistributes data based on the output keys produced by map step
- Reduce – summary operation (reducing list of values)



# Map reduce example

This example uses MapReduce to count words in text files

MapReduce is implemented using:

- std::transform - https://en.cppreference.com/w/cpp/algorithm/transform
- std::accumulate - https://en.cppreference.com/w/cpp/algorithm/accumulate

usage (also in README.txt):

```
$ simplekv_word_count pool file1.txt file2.txt ...
```

# PMDK Resources

## PMDK Resources:

- Home: https://pmem.io
- PMDK: https://pmem.io/pmdk
- PMDK Source Code : https://github.com/pmem/PMDK
- Google Group: https://groups.google.com/forum/#!forum/pmem
- Intel Developer Zone: https://software.intel.com/persistent-memory
- libpmemobj-cpp: https://github.com/pmem/libpmemobj-cpp
- valgrind: https://github.com/pmem/valgrind
- NDCTL: https://pmem.io/ndctl
- Getting Started Guides: https://docs.pmem.io

# More Developer Resources

## Getting Started

- Intel IDZ persistent memory- https://software.intel.com/en-us/persistent-memory
- Entry into overall architecture - http://pmem.io/2014/08/27/crawl-walk-run.html
- Emulate persistent memory - http://pmem.io/2016/02/22/pm-emulation.html

## Linux Resources

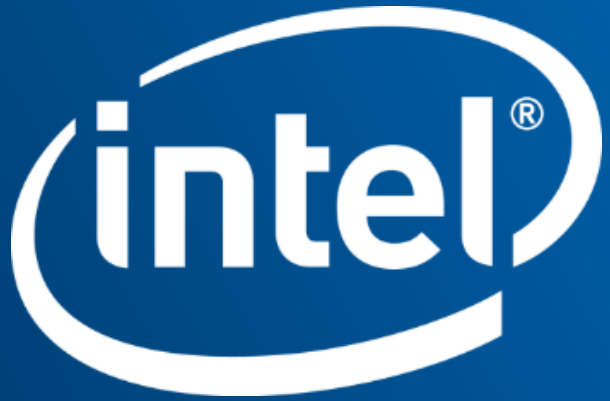
- Linux Community Pmem Wiki - https://nvdimm.wiki.kernel.org/
- Pmem enabling in SUSE Linux Enterprise 12 SP2 - https://www.suse.com/communities/blog/nvdimm-enabling-suse-linux-enterprise-12-service-pack-2/

## Windows Resources

- Using Byte-Addressable Storage in Windows Server 2016 - https://channel9.msdn.com/Events/Build/2016/P470
- Accelerating SQL Server 2016 using Pmem - https://channel9.msdn.com/Shows/Data-Exposed/SQL-Server-2016-and-Windows-Server-2016-SCM--FAST

## Other Resources

- SNIA Persistent Memory Summit 2018 - https://www.snia.org/pm-summit
- Intel manageability tools for Pmem - https://01.org/ixpdimm-sw/


intel