

Persistent Memory Workshop

libmemkind, libpmem & libpmemobj-cpp hands-on

Contributors:

Jim Fister (SNIA)

Stephen Bates (Eideticom)

Andy Rudoff (Intel)

Igor Chorażewicz (Intel)

Zhiming Li (Intel)

Peifeng Si (Intel)

<https://github.com/pmemhackathon/2019-07-23>

What Does “Hackathon” Mean To Us?

- Main goal is to show you how to find, configure, and program pmem
 - All slides are in the GitHub repo
 - All shell commands we type are in the GitHub repo
 - You probably don't need to write them down
 - You probably don't even need to type many of them, just cut & paste into the shell
 - Go to <https://github.com/pmemhackathon/2019-07-23> to see today's repo
 - But in a minute, we'll demonstrate cloning the repo to your VM
- Mostly we will show you how to install stuff and get you going
 - After installing samples, try them out, or write your own
 - We'll walk through some for everyone, then will walk around & help you

Agenda

- Logistics
 - How to login to your VM & get it ready
- Persistent Memory Platform Support
 - Platform level support
 - Checking out your kernel
 - Finding and configuring your pmem
- Persistent Memory Programming
 - Installing libraries and tools (pmdk, libmemkind, libpmemobj-cpp, valgrind)
 - Using libmemkind and libpmem
 - Using libpmemobj-cpp:
 - Finding bugs related to persistent memory programming
 - Converting volatile queue to persitent one
 - Hashmap example

Login and make a local clone of the hackathon repo

```
Hostname: devhost.pmemhackathon.io      Port: 31005  
Username: pmdkuser{1...50}
```

```
$ cd  
$ git clone https://github.com/pmemhackathon/2019-07-23  
Cloning into '2019-07-23'...  
remote: Enumerating objects: 14, done.  
remote: Counting objects: 100% (14/14), done.  
remote: Compressing objects: 100% (13/13), done.  
remote: Total 14 (delta 1), reused 14 (delta 1), pack-reused 0  
Unpacking objects: 100% (14/14), done.  
$ cd 2019-07-23  
$ more README.txt
```

Most of the shell commands we type during demos are in this README.txt

Agenda

- Logistics
 - How to login to your VM & get it ready
- **Persistent Memory Platform Support**
 - Platform level support
 - Checking out your kernel
 - Finding and configuring your pmem
- Persistent Memory Programming
 - Installing libraries and tools (pmdk, libmemkind, libpmemobj-cpp, valgrind)
 - Using libmemkind and libpmem
 - Using libpmemobj-cpp:
 - Finding bugs related to persistent memory programming
 - Converting volatile queue to persitent one
 - Hashmap example

Does your System Support Persistent Memory?

- Does my platform support persistent memory?
 - Your vendor determines this. Buy a system meant for it.
 - Don't just buy an NVDIMM and plug it into a random system – you need platform support (like BIOS, ADR, power supply). You want validated configurations.

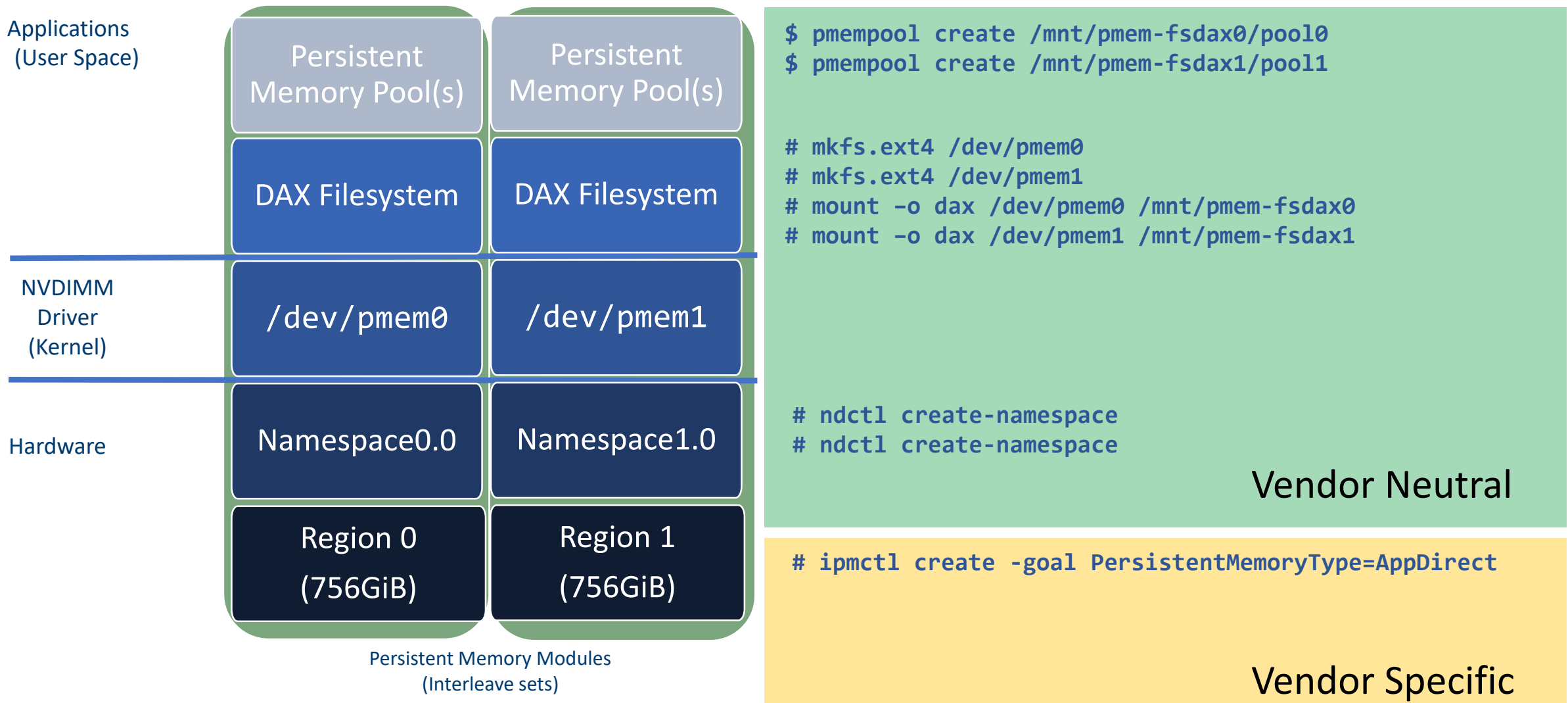
```
ndctl list -BN      # check the "provider" field for ACPI.NFIT
```

- Does my OS support persistent memory?
 - Major OS vendors (and Linux distros) will tell you which version supports it
 - Linux kernel support is enabled in the config file used to build the kernel

```
uname -r           # see kernel currently running
grep -i pmem /boot/config-`uname -r`
grep -i nvdimmm /boot/config-`uname -r`
```

- Are the persistent memory devices mounted?

```
lsblk              # check the mounting name of persistent memory devices
```



Agenda

- Logistics
 - How to login to your VM & get it ready
- Persistent Memory Platform Support
 - Platform level support
 - Checking out your kernel
 - Finding and configuring your pmem
- **Persistent Memory Programming**
 - Installing libraries and tools (pmdk, libmemkind, libpmemobj-cpp, valgrind)
 - Using libmemkind and libpmem
 - Using libpmemobj-cpp:
 - Finding bugs related to persistent memory programming
 - Converting volatile queue to persitent one
 - Hashmap example

Essential Programming Background

- Lots of ways to use pmem with existing programs
 - Storage APIs
 - Libraries or kernels using pmem transparently
 - Memory Mode
- This hackathon doesn't cover the above (too easy!)
 - We assume you want direct access to pmem
 - We show code, but also concepts
 - There are lots of paths you can take, these are just examples

Programming Examples For This Hackathon

- Volatile alloc and free
- Persistent write
- Simple persistent counter
- Converting volatile queue to persistent one
- Implementing a hashmap
- Processing data on persistent memory using map reduce

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>
 - memkind: <https://github.com/memkind/memkind>
- NDCTL: <https://pmem.io/ndctl>
- SNIA NVM Programming Model:
https://www.snia.org/tech_activities/standards/curr_standards/npm
- Getting Started Guides: <https://docs.pmem.io>

Getting started

- Get hackathon repo:

```
$ git clone http://github.com/pmemhackathon/2019-07-23  
$ cd 2019-07-23
```

- How to compile examples?
 - Simply run:

```
$ make
```

Libmemkind

Volatile allocate and free

- Use pmem as volatile memory
- Libmemkind provides APIs to manage the space on pmem
- Run:

```
$ ./volatile /mnt/pmem-fsdax0/${USER}/  
Create memkind on /mnt/pmem-fsdax0/pmdkuser1  
Allocate 32 bytes for pmem_str0  
Allocate 31 bytes for pmem_str1  
Free pmem_str0  
Free pmem_str1  
Destroy memkind
```

Libpmem

Persistent write

- Libpmem provides primitive APIs to write data on pmem persistently
- Copy a file to pmem and check its content
- Run:

```
$ ./simple_copy simple_copy.c /mnt/pmem-fsdx0/${USER}/simple_copy.c
```


libpmemobj-cpp

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>

Persistent counter – what you should do

- Make sure you have the newest version of hackathon repo:
- 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-fsdx0/${USER}/warmup
$ ./warmup /mnt/pmem-fsdx0/${USER}/warmup
1
$ ./warmup /mnt/pmem-fsdx0/${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

```
$ 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-fsdx0/${USER}/queue  
$ ./queue_pmemobj_cpp /mnt/pmem-fsdx0/${USER}/queue  
$ push 1  
$ show
```

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
5. Add transactions

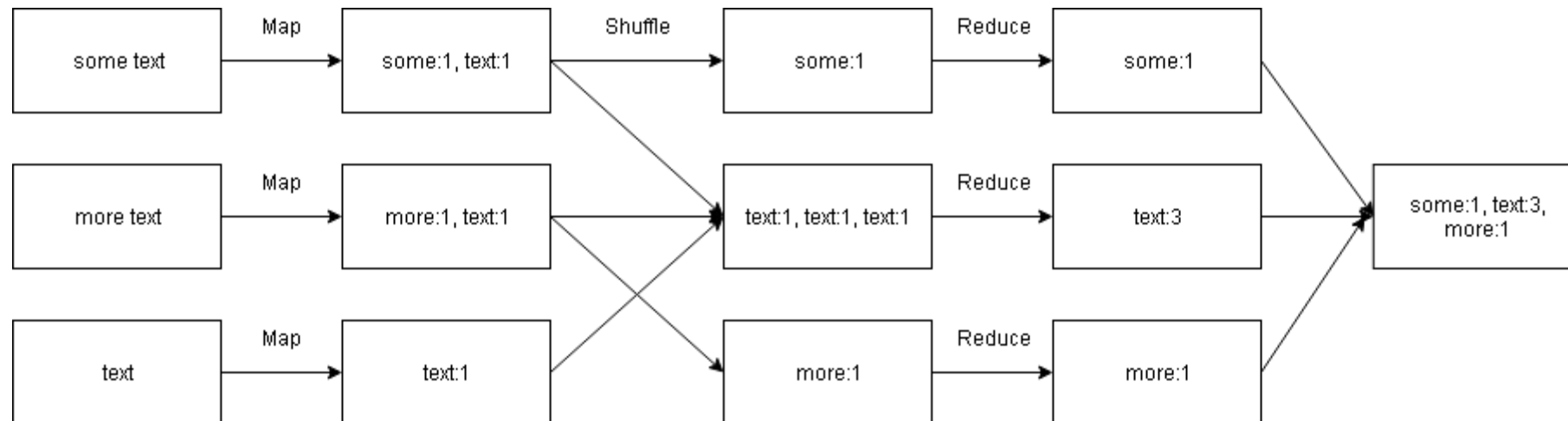
HashMap

- Implement a hashmap with following interface:
 - at(key)
 - Insert(key, value)
- To check if it works, compile and run:

```
$ pmempool create obj --layout=simplekv -s 100M /mnt/pmem-fsdx0/${USER}/simplekv-simple  
$ ./simplekv_simple /mnt/pmem-fsdx0/${USER}/simplekv-simple
```

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):

```
$ pmempool create obj --layout=simplekv -s 100M /mnt/pmem-fsdax0/${USER}/simplekv-words  
$ ./simplekv_word_count /mnt/pmem-fsdax0/${USER}/simplekv-words words1.txt words2.txt
```

More Developer Resources

- Find the PMDK (Persistent Memory Development Kit) at <http://pmem.io/pmdk/>
- 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/>