INTRODUCTION TO INTEL® 3D XPOINT™ LARGE CAPACITY PERSISTENT MEMORY AND ITS OPEN SOURCE API

Usha Upadhyayula
Data Center Solutions Group

Eduardo Berrocal
Software and Solutions Group
Topics

• Introduction to Persistent Memory
• Introduction to Non-Volatile Memory Libraries (NVML)
• Working with NVML
  • Github* location, Code Samples, Building NVML code
  • Emulating Persistent Memory using DRAM
• Evolving Persistent Memory Usages in HPC
• Persistent Memory Programming with NVML C++ Bindings
  • “Hello World” Persistent Memory
  • Other Code Samples
• Summary
• Q&A
Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright© 2017, Intel Corporation. All rights reserved. Intel, the Intel logo, Atom, Xeon, Xeon Phi, Core, VTune, and Cilk are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804
Intel Persistent Memory

New Type of Memory

• Persistent, Large Capacity & Byte Addressable
  • 6 TB per two-socket system
• DDR4 Socket Compatible
  • Can Co-exist with Conventional DDR4 DRAM DIMMs
  • Demonstrated at SAP Sapphire and Oracle Open World 2017
• Cheaper than DRAM
• DIMM Capacities
  • 128 GB
• Availability
  • 2018
Sources: “Storage as Fast as the rest of the system”. 2016 IEEE 8th International Memory Workshop and measurement, Intel® Optane™ SSD measurements and Intel P3700 measurements, and technology projections.
Introduction to Persistent Memory

• What is Persistent Memory?
  • Byte Addressable
  • Persistent Like Storage
  • Load /Store Access
    • No page caching
  • Cache Coherent
  • Memory-like Performance

• Why Does it Matter now?
  • Adds a new tier between DRAM and Block Storage (SSD/HDD)
  • Larger Capacity, Higher Endurance, Consistent low latency
  • Ability to do in-place persistence
    • No Paging, No Context Switching, No Interrupts, No Kernel Code Running
  • Ability to do DMA & RDMA

• What’s the impact on the Applications?
  • Need Ways to Enable Access to New High Performance Tier
  • May need to re-architect to unlock the new features and performance
What does Re-architecting to Byte-Addressable Persistent Memory Mean

• Applications Need to
  • Ensure Data Persistence
    • Stores are not guaranteed persistence until flushed
      • Need to flush the CPU Caches to Persistent Domain

• Assure Data Consistency
  • Maintain Transactional Consistency of Persistent Data structures
    • Prevent torn updates

• Use Persistent Memory Aware Allocation/Free
  • A leak in persistent memory remains leaked

• Detect and Handle Persistent Memory Programming Errors
Data Persistence – Importance of Flushing

Minimum Required
Power fail protected domain:
Memory subsystem

Custom
Power fail protected domain
indicated by ACPI property:
CPU Cache Hierarchy

Core
L1
L2
L3

DIMM

CPU CACHES

CLWB + fence
-or-
CLFLUSHOPT + fence
-or-
CLFLUSH
-or-
NT stores + fence
-or-
WBINVD (kernel only)

ADR
-or-
WPQ Flush (kernel only)
Data Consistency – Need for Transactions

Flushing to Persistence

open(...);
mmap(...);
strcpy(pmem, "Hello, World!");

pmem_persist(pmem, 14);

Crossing the 8-Byte Store

Result?
1. "\0\0\0\0\0\0\0\0..."
2. "Hello, W\0\0\0\0\0\0..."
3. "\0\0\0\0\0\0\0\0rld\0"
4. "Hello, \0\0\0\0\0\0\0\0"
5. "Hello, World!\0"
Byte Addressable Storage With Memory Mapped Files– Before and After

Before

Avoid Overhead of Paging/Context Switching into Kernel

After
Topics

• Introduction to Persistent Memory
• Introduction to Non-Volatile Memory Libraries (NVML)
• Working with NVML
  • Github* location, code samples, building NVML code
  • Emulating Persistent Memory using DRAM
• Evolving Persistent Memory Usages in HPC
• Persistent Memory Programming with NVML C++ Bindings
  • “Hello World” Persistent Memory
• Summary
• Q&A
The SNIA NVM Programming Model – Exposing Persistent Memory to Applications

Management
- Management UI
- Management Library

Block Mode
- Application
  - Standard Raw Device Access
  - Standard File API
- Application
  - File System
  - NVDIMM Driver

Persistent memory
- Application
  - Standard File API
  - Load/Store
  - User Space
  - Kernel Space
  - MMU Mappings
  - pmem-Aware File System

NVDIMMs

SNIA – Storage and Networking Industry Association
NVM – Non-volatile Memory
NVML: A Suite of Open Source Libraries

**Support for volatile memory usage**

- **memkind**

**Low-level support for local persistent memory**
- **libpmem**

**Interface to create arrays of pmem-resident blocks, of same size, atomically updated**
- **libpmemobj**

**Interface for persistent memory allocation, transactions and general facilities**
- **libpmemlog**

**Interface to create a persistent memory resident log file**
- **libpmemblk**

**Link to Open Source**

**Link to Intel® Developer Zone**

**In Development**

**C++, C, Java, Python**

**Application**
- Standard File API
- Libraries
- User Space
- Load/Store

**User Space**

**Kernel Space**

**NVDIMM**

**Low-level support**

**pmem-Aware File System**

**MMU Mappings**
Why NVML

• Simplifies/Facilitates Adoption of Persistent Memory Programming with Higher Level Language Support
  • Built on top of SNIA Programming Model
  • Builds on DAX capabilities on Linux* and Windows*
    • Data Access Granularity at Cache Line
• Provides API to
  ✓ Ensure Data Persistence
  ✓ Data Consistency Assurance
    • Transactional Operations
    • Power Fail Atomicity
  ✓ Allocate/Manage Persistent Memory
• Developer Tools For Debugging and Error Handling
libpmemobj: Core Concepts

✓ Data Consistency Assurance
  • Support for Transactional & Atomic Updates

✓ Implicit Persistence during Transactions
  • Knows When and How to Flush the Data to Persistence Domain

✓ Persistent Memory Allocation & Management API
  • Memory Mapped Region Exposed as Object Pools
    • Thread-safe
  • Allocate/Free Persistent Data Structures
    • Position Independent
  • Root Object
    • Known Location in the Persistent Memory Pool
      • Anchor for Objects in the Pool

✓ Multi-thread Support
Volatile Use of Persistent Memory - libmemkind

- Open Source Memory Allocation Library
  - Provides malloc/free-style API

- Single Allocator Handles Multiple Kinds of Memory

https://github.com/memkind/memkind/blob/master/examples/pmem_example.c
Topics

• Introduction to Persistent Memory
• Introduction to Non-Volatile Memory Libraries (NVML)
• Working with NVML
  • Github* location, code samples, building NVML code
  • Emulating Persistent Memory using DRAM
• Evolving Persistent Memory Usages in HPC
• Persistent Memory Programming with NVML C++ Bindings
  • “Hello World” Persistent Memory
• Summary
• Q&A
NVML Binary Packages

• Download Pre-built Packages:
  • https://github.com/pmem/nvml/releases
  • Works on 64-bit Linux* or Windows* 2016

• Upstreamed into multiple Linux* Distros
  • For example: Fedora* 24
    • dnf install libpmemobj-devel
      • Installs libpmem, libpmem-devel, libpmemobj, libpmemobj-devel
      • C++ header files are not part of this install

• NVML Available as Windows Technical Preview
  • https://github.com/pmem/nvml/releases/tag/1.2%Bwtp1
Building Tech Preview NVML Code from Github*

- git clone
  - https://github.com/pmem/nvml
- cd nvml
- make
- make install (as root)
  - Installs man pages and libraries
  - Install libraries in non-standard locations
    - make install prefix=/xxx/xxx

- Customizing Build Settings
  - Compile with persistent memory aware Valgrind
    - make EXTRA_CFLAGS=-DUSE_VG_PMEMCHECK
Get Started with Persistent Memory Programming without Real Hardware

- PM-aware Applications can be Written/Tested
- Using SSD or Hard Disk
- Any Linux* or Windows* distro that supports Memory Mapped Files (with No DAX Support)
- EXT4, XFS, NTFS
  - Support for DAX
- Emulating Persistent Memory
  - [http://pmem.io/2016/02/22/pm-emulation.html](http://pmem.io/2016/02/22/pm-emulation.html)

Start with Emulation; Validate & Tune on Platforms with Real DIMMs
Emulating Persistent Memory Using DRAM (Linux*)

Reserve PMEM through Kernel Configuration(kernel >4.2)

- Enable BIOS to treat Memory Marked as e820- type 12 as PMEM
- Enable Direct Access (DAX support)
- Identify Usable regions
- Specify Kernel Parameter (Grub file)
  - memmap=<size of pmem>!<offset in DRAM>
- On reboot
  - pmem region is created
  - Kernel offers this space as PMEM to the PMEM driver
  - OS sees this DRAM region as PMEM & Creates
    - /dev/pmem0, /dev/pmem1 ..

Make File system

- EXT4 and XFS modified to support PMEM
  - sudo mkfs.ext4 /dev/pmem0 or sudo mkfs.xfs /dev/pmem0

Mount it on /dev/pmemx with DAX option

- Ex: sudo mount -o dax /dev/pmem0 /mnt/pmem/
Topics

• Introduction to Persistent Memory
• Introduction to Non-Volatile Memory Libraries (NVML)
• Working with NVML
  • Github* location, code samples, building NVML code
  • Emulating Persistent Memory using DRAM
• Evolving Persistent Memory Usages in HPC
• Persistent Memory Programming with NVML C++ Bindings
  • “Hello World” Persistent Memory
• Summary
• Q&A
Evolving Persistent Memory Usages in HPC

- System’s point of view
  - Augmented capacity (great for memory-bound codes)
    - **6TB** for a 2 socket system
    - Speed close to that of DRAM
    - Cheaper than DRAM
  - Use case:
    - **ISO3DFD**: solve 3D acoustic isotropic wave equation
    - Used for Oil & Gas Exploration
    - Check-point hundreds of MBs each time step; reuse it in future time steps
Evolving Persistent Memory Usages in HPC

• Middleware’s point of view
  • Adding capabilities – such asFault Tolerance – transparently to users
    • Example: PMEM-aware MPI (one sided and MPI-IO)
      • One sided extension: Define persistent “windows” for automatic check pointing
      • MPI-IO: 2 modes: PMEM_IO_AWARE_FS, PMEM_IO_DISTRIBUTED_CACHE
    • Improving hierarchical check-pointing by using PMEM as “level 0” (super fast check-pointing)
      • 1 check-point for every time step
Evolving Persistent Memory Usages in HPC

• Application’s point of view
  • Expanding application’s features and interactivity
    • Enables **Interactive In-Situ Visualization** with Large Memory capacities (e.g. Para View*)
      • Multiple time-steps can be kept in memory eliminating round trips into File system
      • Improve restart times on Application crash/power fails
Topics

• Introduction to Persistent Memory
• Introduction to Non-Volatile Memory Libraries (NVML)
• Working with NVML
  • Github location, code samples, building NVML code
  • Emulating Persistent Memory using DRAM
• Evolving Persistent Memory Usages in HPC
• Persistent Memory Programming with NVML C++ Bindings
  • “Hello World” Persistent Memory
• Summary
• Q&A
Persistent Memory Programming with C++ Bindings
C++ Bindings for libpmemobj

• Goal
  • Simplify Programming with Persistent Memory Objects
    • Using C++ familiar constructs for Persistent Memory Allocator & Transactions

• C++ Version Support:
  • C++ 11 compliant compiler or new
  • GCC (4.8.1) and Clang (3.3), Visual Studio* 2012

• Implementation
  • All classes/functions in a single C++ namespace
    • nvml::obj
  • Header file: /include/libpmemobj
Main Features of C++ Bindings

- Support for Transactions
  - C++ Wrappers for
    - Scoped Transactions
      - Both Manual and Automatic
    - Closure-like Transactions
  - Support for Automatic Snapshotting of Basic Data Types in a Transaction
    - p<> Property: Wrapper for basic types
    - Indicates persistent variable
- Persistent Pointer: C++ Wrapper with properties of std::shared_ptr
  - Designed to handle complex classes
  - Support for Allocating/Deallocating functions
    - Transactions & Atomic API
  - Does not work with polymorphic types
Sample Code using Persistent Memory C++ Classes
#include <stdio.h>
#include <fcntl.h>
#include <iostream>
#include <libpmemobj++/p.hpp>
#include <libpmemobj++/persistent_ptr.hpp>
#include <libpmemobj++/pool.hpp>

using namespace nvml::obj;
using namespace std;

struct root{
    p<int> pint;
    p<double> pdouble;
};

int main (int argc, char* argv[])
{
    /* open pool */
    auto pop = pool<root>::open("/mnt/mem/poolfile","layout");

    /* get root object */
    auto root_obj = pop.get_root();

    /* update the data */
    root_obj->pint = 5;
    root_obj->pdouble = 4;
    cout << " persistent variables:" << root_obj->pint << " " << root_obj->pdouble << endl;

    /* close pool */
    pop.close();
    return 0;
}
Hello World with Persistent Data Structures

```cpp
#include <stdio.h>
#include <fcntl.h>
#include <iostream>
#include <pmemobj++/p.hpp>
#include <pmemobj++/persistent_ptr.hpp>
#include <pmemobj++/pool.hpp>
#include <pmemobj++/make_persistent.hpp>
#include <pmemobj++/transaction.hpp>

using namespace nvml::obj;
using namespace std;

struct root { /* Structure with transaction aware wrappers for basic types */
    p<int> pint;
    p<int> pint2;
    p<int> pint3;
};

int main (int argc, char* argv[])
{
    /* open pool */
    auto pop = pool<root>::open("poolfile","layout"); /* add check (raises exception) if file exists or not */

    /* get root object */
    auto root_obj = pop.get_root();

    /* update the data transactionally */
    transaction::exec_tx (pop,&{
        root_obj->pint = 5;
        root_obj->pint2 = 6;
        root_obj->pint3 = root_obj->pint+root_obj->pint2;
    });
    cout << " persistent variables:" << root_obj->pint << " " << root_obj->pint2 << " " << root_obj->pint3 << endl;
    pop.close();
    return 0;
}
```
Transactions with Basic Types using p <> Property

```cpp
using namespace nvml::obj;
struct root {
    p<int> pint;
    p<int> pint2;
    p<int> pint3;
};

// Do the transactional allocation using C++ Lambda function
transaction::exec_tx (pop, [&] {
    root_obj->pint = 5;
    root_obj->pint2 = 6;
    root_obj->pint3 = root_obj->pint + root_obj->pint2;
});
```

- All operations in this block are transactional. Allocations and assignments are done atomically.
Sample 3: Transactions with Complex Types using persistent_ptr

```cpp
class complex_type {
private:
  p<double> counter;
  p<int> pint;

public:
  complex_type () { counter = 0.0; pint = 0; }
  complex_type (double c, int p) : counter (c), pint (p);
  void set_pint (int var) { pint = var; }
  int get_pint () { return pint; }

  void set_event () {
    counter = counter + 1;
  }
  double get_counter () { return counter; }
};

transaction::exec_tx (pop,&){
  /* transactionally allocate and construct the object */
  root_obj->comp = make_persistent<complex_type>();
  for(int i = 0; i <= 5; i++) {
    root_obj->comp->set_pint(i);
    if (root_obj->comp->get_pint() == 2){
      root_obj->comp->set_event();
    }
  }
  /* transactionally delete the object */
  delete_persistent<complex_type>(root_obj->comp);
};
```

- All operations in this block are transactional
- ```make_persistent()``` : Allocates memory and constructs objects in-place
Multithreading Support
Synchronization Primitives

- Synchronization Primitives Reside in Persistent Memory
  - In the event of a crash these primitives are automatically released
- Mutex
  - Usage same as std::mutex
  - Supports Shared and Timed Mutex
  - Sample usage:
    - [http://pmem.io/nvml/cpp_obj/master/cpp_html/classnvml_1_1obj_1_1mutex.html#a787e04aea62ea7499c0e9a9da3d756b](http://pmem.io/nvml/cpp_obj/master/cpp_html/classnvml_1_1obj_1_1mutex.html#a787e04aea62ea7499c0e9a9da3d756b)
- Condition variables
  - Usage same as std::condition_variable
  - Sample Usage:
    - [http://pmem.io/nvml/cpp_obj/master/cpp_html/classnvml_1_1obj_1_1condition_variable.html](http://pmem.io/nvml/cpp_obj/master/cpp_html/classnvml_1_1obj_1_1condition_variable.html)
using namespace nvml::obj;
struct compound_type {
  p<int> pint;
  p<double> pdouble;
  void set_int_variable (int val) {
    pint = val;
}
  shared_mutex smtx;
};

/* pool root structure */
struct root {
  persistent_ptr<compound_type> comp;
};

/* create a pmemobj pool */
auto pop = pool<root>::create("poolfile", "layout", PMEMOBJ_MIN_POOL);
auto proot = pop.get_root();

/* do the transaction: object allocation and assignment */
transaction::exec_tx(pop, [&] {
  proot->comp = make_persistent<compound_type>();
  proot->comp->set_int_variable(12);
  proot->comp->pdouble = 2.0;
}, proot->comp->smtx);

• All operations in this block are transactional and atomic
• make_persistent() : Allocates memory and constructs objects in-place
Putting it all Together – Map-Reduce Example

• Programming model introduced by Google* in 2004
• Inspired by functional programming
  • Primitives map and reduce in Lisp
• Typical “hello world” example or MR is word counting.

* Overview of MapReduce. This figure is a modified version of Figure1 from: MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Dean and Sanjay Ghemawat
Putting it all Together – Map-Reduce Example

- MR model achieves FT by storing its intermediate results in files residing in a local file system.
  - Attached HDD or SSD
  - **Problem:** multiple orders of magnitude difference in speed against RAM
- **Solution 1:** Mount your local file system on top of a PMEM device
  - Improved performance but...
  - **Problem:** software still needs to be designed with a volatile-versus-persistent memory mentality
- **Solution 2 (NVML):** Program directly against PMEM
  - To achieve FT, specify what data structures should be permanent
Putting it all Together – Map-Reduce Example

- Data Structures
Putting it all Together – Map-Reduce Example

Placeholder to Show Code on Github* & run the sample remotely
Topics

• Introduction to Persistent Memory
• Introduction to Non-Volatile Memory Libraries (NVML)
• Working with NVML
  • Github* location, code samples, building NVML code
  • Emulating Persistent Memory using DRAM
• Evolving Persistent Memory Usages in HPC
• Persistent Memory Programming with NVML C++ Bindings
  • “Hello World” Persistent Memory
  • Other Code Samples
• Summary
• Q&A
Summary

- NVML Resources:
  - Intel® Developer Zone: https://software.intel.com/persistent-memory
  - Webinar
    - Persistent Memory Programming using NVML
      - https://software.seek.intel.com/PersistentMemory_Programming_Reg
  - pmem.io - http://pmem.io/
  - Google* group: https://groups.google.com/forum/#!forum/pmem
  - Source: https://github.com/pmem/nvml
  - Man pages for the libraries:

- Additional Resources
  - An overview of persistent memory programming
    - http://www.snia.org/PM
  - SNIA* Standards Portfolio
    - NVM Programming Model v1.2a – Draft for public review
BACKUP
Importance of Flushing: Application with Memory Mapped Files

• Why is flushing required?
  • With Memory-mapped files:
    • Stores are not guaranteed persistence until flush API is called
    • Stores are visible before they are persistent

• Do standard flushing APIs work with pmem?
  • Yes, standard APIs work as expected
    • msync() on Linux
    • FlushFileBuffers() on Windows
    • The kernel will use instructions like CLWB as necessary

• Can Applications just flush with CLWB from user space
  • Only when supported by the kernel/file system
  • Libraries like NVML determine when it is safe