NVM Direct

Version 0.3

The advent of persistent memory in enterprise computing introduces new problems related to managing consistent application state. To solve these problems Oracle is developing NVM Direct, an open source C library and a set of C language extensions for applications utilizing NVM. This library and the extensions implement an application interface (API) that is applicable to a wide range of applications.

This repository contains the shared library portion of the API and an example application. It is mostly complete, but there are a few pieces missing. It has not been rigorously tested yet. This preliminary version is being released so people can understand the intent of the API, and get a feel for what it can do. This is not production quality.

There will be a GitHub repository containing the C extensions precompiler source code when it is completed. For now the Linux executables for the precompiler can be downloaded from the Oracle Technology Network at the NVM Direct page. The precompiler consists of two executables: clang and usidmap. The precompiler itself is a modified version of clang which takes the preprocessed version of code using the C extensions, and produces C source which gcc can compile. The precompiler also produces a JSON file describing all the persistent structs and callback functions. The JSON file is processed by usidmap to generate the code to register all the USID's for an application. The makefiles to use these executables are included in the eclipse DebugX build configurations for all projects. Extract the two executables and ensure they are available through your $PATH as clang and usidmap. This should work on any modern x64 Linux distribution.

The library is coded both with and without the C extensions. If `NVM_EXT` is defined then the version with extensions will be compiled. This will be set if the DebugX build configuration is used. The two versions provide a comparison of how the extensions will look when used in a real application.

The library has been developed using Eclipse Luna. There are three Eclipse projects in the repository for the NVM Direct code. There are two build configurations for each project. The configurations ending in X compile the code with the C extensions. Of course that will not work without the precompiler.

There are two Eclipse projects for an example application. They are written with the C extensions and require the precompiler.

The bin project just contains a shell script to compile a C file that uses the NVM Direct C extensions. It is only intended to work with the makefiles generated by Eclipse Luna for gcc. However it should be easy to modify for other environments.

Note that most of the DebugX build configurations use a custom shell script to run usidmap and to post process its output. This is necessary because of some problems with usidmap. A future version will not

require the post processing. The test_dmf project does not run usidmap because it does not define any USID's.

Eclipse Projects:

- `test_nvmd`: This is a silly NVM application that tries to get most of the library code called. It forks itself and kills its child to exercise recovery.
- `nvmd`: This is the portable shared library. It contains the bulk of the code.
- `nvmds`: This is a service layer that allows the library to be customized for different environments. The current service layer is for testing the library on Linux. It does not actually work with persistent memory. It needs to integrate with Intel's libpmem to actually work with NVM.
- `DirectMappedFile`: This is an example shared library written with the NVM Direct API. It uses NVM Direct to create a region file that supports atomic block I/O to NVM using memcpy. It supports an arbitrary number of threads simultaneously doing reads and writes. The size of the file can be changed concurrently with I/O. This is similar to the functionality provided by Block Translation Table. See dmf.h and dmf.c for more information.
- `test_dmf`: This is an application to test DirectMappedFile. It gets multiple threads simultaneously reading/writing/resizing a file.
- `bin`: This just contains a shell script for compiling sources that use the C extensions for NVM Direct.

Documents:

There is some documentation in the root of the repository. This is intended to aid in understanding the code.

- `LICENSE`: This contains the Universal Permissive License that this code is released under.
- `NVM_Direct_API.pdf`: This is an 100 page document describing the API and the reasoning behind the design. It includes the extended C syntax and all the publicly available library entry points. It is not as detailed as man pages, but it is a start.
- `NVM_Direct_Presentation.pdf`: This is a slide deck that has been used for presentations on the API. It is pretty basic.
- `NVM_Direct_Notes.pdf`: This is the same presentation, but with the notes for each slide. The notes give information which could not be presented due to time constraints.

To Do:

The API is not yet an industrial strength software package ready for use in production code. This alpha release has all the software interfaces that a production quality release will have, but lacks some of the external support that is really required. Some of the algorithms can also be improved.

- The error reporting is inadequate. Real developers will need much more information about asserts caused by improper library calls and NVM corruptions found by the library. An error reporting

system needs to be developed.

- The documentation does not go down to the level of detail a developer expects in a man page. The .h files are the best documentation. The man pages need to be written.
- There are Doxygen annotated comments throughout the code, but there is no overall Doxygen documentation. Doxygen documentation needs to be generated and made complete.
- There is a small test program used for initial debugging. However it is far from an adequate test of the entire API in all its corner cases. A full test suite needs to be developed.
- The code contains a few TODO comments that create Eclipse tasks. These still need to be addressed. They do not significantly impact functionality.
- The library does not actually work with persistent memory. The service layer uses a disk file system and does not actually do the necessary processor cache flushing and persistent memory barriers needed for real NVM. The service layer needs to be integrated with libpmem.
- The precompiler that implements the language extensions is not complete yet, and will not be open sourced until it is in a better state. The library cannot be compiled with `NVM_EXT` defined without the precompiler. Development versions of the precompiler have successfully compiled the code with extensions, but not all of the extended features are available. The most significant issue is the automatic construction of the initialized nvm_type structs that describe the persistent structs. They have been hand crafted in usidmap.c. It is difficult to keep the C declarations and usidmap.c in sync.
- Debugger support is needed for the language extensions. Following self-relative pointers in gdb is cumbersome at best. The debugger should be able to recognize USID's and report structs properly formatted. Getting formatted dumps of transaction undo could be very useful for resolving recovery problems.
- Eclipse and other IDE's contain C parsers that do not recognize the NVM extensions for C. This makes some Eclipse features unusable with code that contains extensions.
- There are some configuration parameters that the service layer passes to the generic layer of the library. Currently these are hard wired as numeric values in the service layer code. There needs to be a mechanism to allow per application configuration of the parameters.
- Since the code has never been run on real NVM there are no performance numbers. Benchmarks need to be developed and run on real hardware to evaluate the library.
- The code has not been instrumented to report performance statistics. This is needed for application tuning, and bottleneck discovery.