Tool: Pmemcheck persistent memory programming correctness checker

Advantages:

1) pmemcheck helps to detect several pmem errors such as non-persistent stores, memory overwrites, and redundant cache flushes as illustrated below.
2) All the PMDK libraries are instrumented with pmemcheck and it is the ideal tool for PMDK programmers to verify code correctness with no code changes.

Limitations:

1) The persistent memory regions in the program need to be explicitly specified to pmemcheck within VALGRIND_PMC_REGISTER_PMEM_MAPPING and VALGRIND_PMC_REMOVE_PMEM_MAPPING macros.
2) pmemcheck works differently with C and C++ for transactions, and is better suited for C++ since it is doesn't expect the programmer to specify the beginning and end of transaction using VALGRIND_PMC_START_TX_N and VALGRIND_PMC_END_TX_N macros; concurrency issues are also easy to resolve with libpmemobj-cpp since a list of locks can be passed using lambda functions during the creation of a transaction.
3) pmemcheck does not display the exact location of the error and the developer has to rely on the function where the error is detected for debugging.

Examples of errors detected by pmemcheck

1) Non persistent stores

Program: np_stores.c

When data is written to persistent memory but is not explicitly flushed, there is a possibility of data being corrupted since a program crash could result in the loss of data in CPU caches. The example in the program shows memcpy being done without flushing, resulting in an error. To fix the error, pmem_memcpy_persist could be used to ensure that the memcpy operation persists as commented out in the code.

2) Stores not added in transactions

Programs: (a) stores_not_in_tx.c
(b) stores_not_in_tx1.c
(c) stores_not_in_tx2.c

If in a pmem transaction in C, the field being updated is not added to the transaction properly, pmemcheck throws an error. For instance, in program(a), the fields key and val of root are being updated but only val is added to the transaction, resulting in an error. Error is also detected if no member of root or root itself is added to the transaction but members of root are modified within the transaction as shown in program(b). pmemcheck also points out the number of stores made without the variables being added to the transaction. If it is communicated to pmemcheck using the VALGRIND_PMC_ADD_TO_TX macro the addition of the variable being modified to the transaction without explicitly adding it in the transaction using TX_ADD, pmemcheck is able to recognize the difference and it still displays the same errors as seen in program(c). The errors are fixed by uncommenting the proper way to add root to the transaction.

3) **Same memory added to different transactions**

Program: mmy_in_diff_txs.c

If two threads attempt to write to the same object in different transactions, one of the threads might overwrite the value written by the other thread. This could result in a dirty read of the data. When this happens, pmemcheck warns about overlapping regions of memory registered in different transactions, helping detect and resolve possible concurrency errors.

4) **Memory overwrites**

Program: mmy_overwrite.c

If multiple writes happen to a persistent memory location before an earlier write is flushed, memory overwrite happens as shown in the program. This could cause earlier writes to be lost and potential data corruption. pmemcheck warns us about this, however, the flag '--mult-stores=yes' needs to be included in the execution command.

5) **Same memory added to different transactions**

Program: unnecessary_flush.c

Additional flushes which serve no useful purpose as illustrated in the program are also detected with pmemcheck. However, as before, an additional '--flush-check=yes' flag needs to be added to the command.

Application of pmemcheck to HashTable code from Assignment 2

1) **Verification of program correctness**

All the functions in the hashtable implementation where persistent memory was being written or appended into have been added with the Valgrind Pmem check macros and invoked in a single run in a suitable order to check for correctness. No errors were reported by pmemcheck.

2) **Intentional addition of bugs and checking if pmemcheck detects them**

In the hash table initialization function, the newly initialized hashtable entry is checked for persistence before it is pushed into the persistent vector of hashtable entries. pmemcheck detects this and reports the error. pmemcheck also detects the error when hashtable entry is initialized and never made persistent but is instead populated with persistent key, value pairs. Similarly, if hashtables in root are not made persistent and an attempt is made to initialize the hash tables, pmemcheck gives a detailed error report as against the default segmentation fault. It is also found that pmemcheck doesn't raise an error when Valgrind TX_START, TX_END macros are not used in a C++ transaction as it would have for C transactions if TX_ADD were incorrectly used (this could be because transactions in C++ do not rely on explicit TX_ADD, TX_ADD_FIELD instructions).

3) **Verified that adding pmem check macros for objects which are made persistent does not throw any errors**

The commands and output for all of these programs is present in the Run_Results.pdf file.