1 Coding with NVDIMM

1.1 Creating pmempool

Before NVDIMM can be used, the user must create whats called a memory pool on the NVDIMM. The NVDIMM has several modes, in order to be able to create a memory pool the mode must be set to fsdax. On a server this must be done by the system administrator. To see what mode the NVDIMM is in can be done by the command ndctl-list. A program called pmempool must also be installed on the server, it is this program that will create the memory pool. The command used for creating the memory pool for this thesis is

```
1 pmempool create --layout my_layout --size=50G obj pool.obj
```

The layout is a string stored in the memory pool. When a program access a memory pool it need to send a string that match the string in the memory pool in order to use it. The user can specify the size of the memory pool, if size is not specified the pmempool will create a pool with the lowest size allowed. There are three different types of memory pool to choose from, they are obj, log and blk. Which type of memory pool to use depends on which type of library is used in the program. In this thesis the libpmemobj library was used and that is why obj was used in the creating of memory pool. For log and blk are for the libraries libpmemlog and libpmemblk. The last part of the command line is the name and file address of the memory pool.

1.2 Different types of libraries

1.2.1 Libpmemobj

Libpmemobj[19] allows objects to be stored in persistent memory without being torn by interruptions. The objects in question are not class objects one finds in C++, but instead they are variable-sized blocks of data that fall under the term object storage. The object has an object ID that is independent when it comes to location. The changes or updates to these objects are atomic because the library have transactions to make this happen. This library can be used for multithreading and are have been optimized for scaling when it comes to multithreading. The main author also mention that the C++ version of this library is the cleanest and least prone to error compared to all the other libraries[10]. He therefore recommends that programmers should start

using this library if they are new to persistent memory programming.

1.2.2 libpmemblk and libpmemlog

These libraries are made for specific cases. libpmemblk[17] is used for handling large arrays of persistent memory blocks. The blocks must be larger than 512 bytes in order to work. This library is useful if the program is made to manage a block cache. Libpmemlog[18] is used to append log files. If the program logs a lot of data, it might be better to use libpmemlog in order to avoid going through the traditional file system where most of the time would be spent waiting. Both libraries are built upon the libpmem library, but unlike libpmem their updates can't be torn because of interruptions.

1.3 Using a library

The way of using the a NVDIMM library is a lot similar to using ordinary arrays. Once the programmer have chosen what NVDIMM library to use and included the library in the code the memory pool must be opened. The first thing the code need is the path to the memory pool and a layout, which is a string that identifies the pool that the user can choose what it will be. This can either be a command line argument the user gives when starting the program or it can be hard coded into the code. This is what has been done at listing 1 at line 5 and 11. These two strings are used as arguments when initiating the pool at line 14-15. The initiation is also followed up with an if-sentence at line 16-19 to check that the memory pool has been successfully created. If it has not the program will print out an error message and exit the program.

Next is to create a NVDIMM array, this is what happens at line 21. The NVDIMM array pointer is a void pointer that is casted to a double pointer. The array gets initiated at line 22. The method used is called POBJ_ALLOC, this method are similar to malloc for DRAM. The method have six arguments, the first argument is what memory pool the array will be assigned to. The second argument is the the address of the pointer. Third argument is the type of the elements in the array. The fourth argument is the length of the array, that is the size of type multiplied with the number of elements in array. The last two arguments are set to NULL.

When writing to an NVDIMM array the programmer must use the method called D_RW. It only have one argument which is the name of the array. It is followed up with a pair square brackets that contains the index of the element the programmer wants to write to, an example can be found at line 25.

D_RO is the name of the method one must use to read an element from an NVDIMM array. This method also have one argument which is the name of the array and the square brackets contains the index of the element that will be read. Line 29 is an example of how to add the value of an element in a NVDIMM array to a variable.

In order to deallocate the a NVDIMM array one must use the method POBJ_FREE and the only argument needed is the address of the NVDIMM pointer, an example can be found in line 34. If the programmer forgets to free up the NVDIMM array there will be a permanent memory leak that will last even after the program have stopped running. In order to get rid of the memory leak one must delete the memory pool and create a new one.

Lastly one must close the memory pool before the program is terminated. This is done with pmemobj_close, it only have the pointer for the memory pool as argument. Line 35 shows how to close the memory pool.

Listing 1: Example of coding with NVDIMM

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <libpmemobj.h>
4
5  POBJ_LAYOUT_BEGIN(array);
6  POBJ_LAYOUT_TOID(array, double);
7  POBJ_LAYOUT_END(array);
8
9  #define ARRAY_LENGTH 1000
10 #define LAYOUT_NAME "my_layout"
11 int main(int argc, char *argv[])
12 {
13   double average = 0.0;
14   int i;
15   //The path for the memory pool.
```

```
16    const char path[] = "/mnt/pmem0-xfs/pool.obj";
17
18    /* create the pmemobj pool or open it if it already
         exists */
19    PMEMobjpool *pop;
20    pop = pmemobj_open(path, LAYOUT_NAME);
21    if (pop == NULL) {
22      perror(path);
23      exit(1);
24    }
25    //Creation of NVDIMM array.
26    TOID(double) nvm_array;
27    POBJ_ALLOC(pop, &nvm_array, double, sizeof(double) *
         ARRAY_LENGTH, NULL, NULL);
28    //Writing to the array.
29    for(i=0;i<ARRAY_LENGTH;i++){
30      D_RW(nvm_array)[i] = i;
31    }
32    //Reading from the NVDIMM array.
33    for(i=0;i<ARRAY_LENGTH;i++){
34      average += D_RO(nvm_array)[i];
35    }
36    average = average / ARRAY_LENGTH;
37    printf("%f\n", average);
38
39    POBJ_FREE(&nvm_array);
40    pmemobj_close(pop);
41    return 0;
42  }
```