Virtual Platform for Architecture Exploration and Performance Analysis of Memory Sub-system

Conference Paper · September 2016

CITATIONS
0

READS
76

2 authors, including:

Mohit Negi
HCL
2 PUBLICATIONS **0** CITATIONS

SEE PROFILE

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Virtual Platform for Architecture Exploration and Performance Analysis of Memory Sub-system

Pragnajit Datta Roy, HCL Technologies Pvt. Ltd., Bangalore, India (*pragnajit.d@hcl.com*)

Mohit Negi, HCL Technologies Pvt. Ltd., Noida, India (*mohit-n@hcl.com*)

***Abstract*—Innovation in device physics has led to proliferation of persistent memory technology which are faster, reliable and occupy smaller form factor. These devices can be fabricated on current 40 nm node using existing fabrication process. Memories like RRAM, 3DXP, memristor etc. has the potential to disrupt $60 billion NAND flash market as they offer speed, reliability and lesser area at the same advanced node process. This necessitate accurate performance benchmarking of these new generation memory in system context under different standard workloads much before the device has been productized.**

This paper explores development of such pre-silicon simulation platform leveraging open source technology (QEMU) and mixed mode (un-timed and AT) simulation methodology.

Keywords—NV-DIMM, PCIe, NVMe, AT, LT, ONFI , DDR4, IOPS, latency, OLTP, QEMU, QBox

I. INTRODUCTION

In today's highly competitive business market, high speed data transmission and processing is paramount in gaining a tactical edge over the competition. Over time, we saw significant improvements in processor performance with the introduction of increased frequencies and increased core-count. Storage sub-system also saw improvements by way of innovation in memory hierarchy and architecture to keep up with processor speed enhancements.

Recent improvements in device physics has spawned innovation in storage technology resulting in not only diminishing cost per gigabyte of storage but also reduction in the form factor and power consumption by 3D stacking of memory array bits. Imagine storing 1TB of data or 500 HD movies in you smartphone. 3D persistent storage technology has shown significant promise and may soon replace traditional NAND Flash storage as they have proven faster, reliable and have higher capacity in comparable form factor.

This paper describes development of Pre-Silicon Virtual Platform (VP) for architecture exploration and performance analysis of such memory sub-system leveraging open source tools and technology. The VP developed is used to carry out performance estimation of a new class of memory technology called 3D RRAM (Resistive Random Access Memory) by building a system from scratch and simulating the memory sub-system on a VP model based on Intel based enterprise class server system.

This paper explores approach to developing the VP. It further explores methodology used to create accurate data points for analysis, workload used and finally the result section highlights the performance achieved and compares the results with spreadsheet based analysis (ideal scenario) of both NV-DIMM and PCIe-NVMe based SSD controller, these two controllers are key to accurately characterize the memory performance with respect to overall system.

Section II starts with brief description of 3D RRAM Technology and then explores modelling technique adapted for developing NV-DIMM based memory sub-system.

Section III gives a brief description of NVMe queuing architecture, PCIe-NVMe protocol and modelling technique adapted to develop PCIe-NVMe based SSD sub-system.

Section IV puts forth i7 core based VP developed for architecture exploration and performance analysis of such storage sub-system using mix of untimed and AT modelling semantics.

Section V briefly describes some workload and performance analysis considerations. We ran some standard benchmarking programs like fio (flexible I/O tester) and iometer on the developed platform.

Section VI, we publish our result based on storage performance metrics like IOPS (Input/ outputs per seconds) and average latencies for different host queue depth, controller and memory configurations. We also briefly compare our results with "spreadsheet only" analysis of the system running read only, sequential Logical Block Address (LBA), workload vectors.

Section VII concludes with brief comment on accuracy of the performance metrics and our future work.

II. 3D RRAM NV-DIMM CONTROLLER

A. *3D RRAM Technology*

New applications like Cloud Analytics, Big Data and (Internet Of Things) IoT etc. necessitates development of new class of storage solutions that are more scalable, higher capacity, give better performance and highly reliable.

Resistive RAM is a new class of memory cells that store bits using different resistance characteristic displayed by switching material sandwiched between two metallic electrodes [1]. The simple structure of RRAM cell, its commonly used materials, process steps and manufacturing tools enable use of existing fabrication process to develop memory chips without any significant initial cost [1].

Compared to traditional NAND Flash memory, RRAM is much faster (RRAM Read time is around 1 us and write time is 2 us), bit-alterable and requires lower voltage, enabling its use in both embedded and SSD applications. The versatility of RRAM technology permits very low latency memory access and offers best area efficiency in a 3D Cross Point array (3D RRAM) configuration.

B. *NV-DIMM Controller*

From system perspective, RRAM solid state memory device can be fabricated in different form factor, such as SSD or NV-DIMM based memory chips.

Figure 1 below shows a simplified view of the inherent performance difference between equal capacities NV-DIMM, PCIe-NVMe based SSDs, and HDDs.

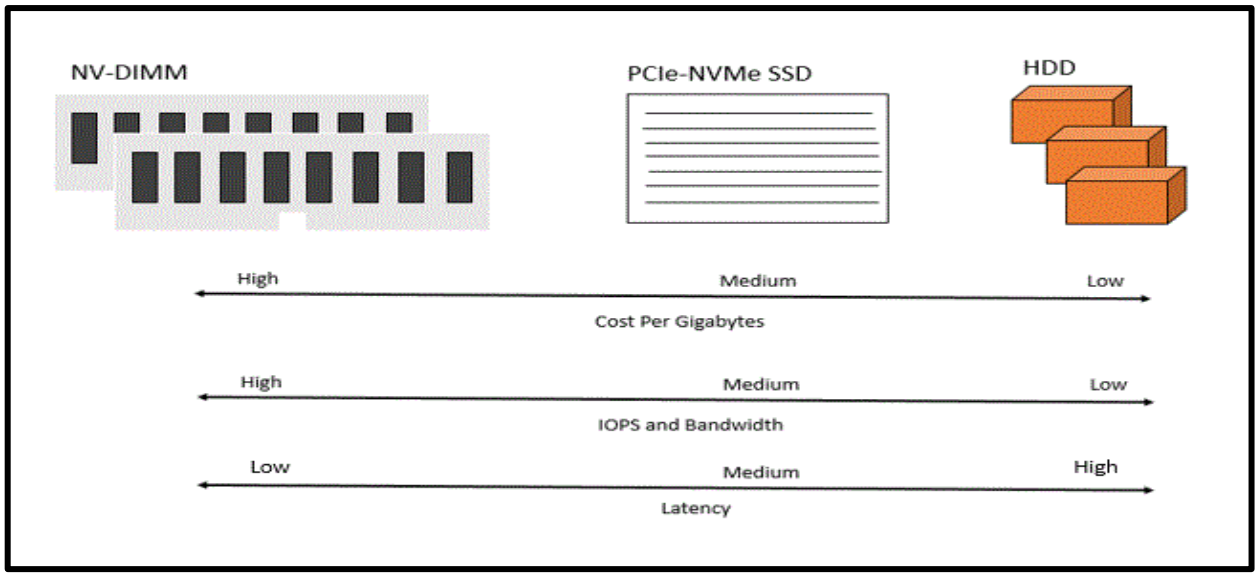

Figure 1: Performance comparison between DIMM, SSD and HDD

This new class of memory needs custom driver as it is row addressable without elaborate refresh and read-modify and write cycle that are characteristic of DRAM and NAND flash memories respectively.

C. *Block driver consideration*

NV-DIMM is treated as a block device node by the host driver. We have developed Linux block driver that is initialized as an RRAM device node. The submission queue (SQ) of the driver is a circular buffer. The block layer submits commands to be dispatched into the SQ in a separate thread. The SQ dispatches commands to the NV-DIMM Controller's command and data buffer over CPU/DDR4 bus fabric via DMA. Owing to high spatial

locality of data accesses, the block layer assembles the data transaction request into fixed size blocks (IO Size) with each byte in the IO block having contiguous locations in memory. The IO Size can vary from 512 bytes, 1K up to 8K. The block driver dispatches commands payload with starting LBA and number of sub-blocks as payload field.

As we will see later, performance analysis is done by varying IO size with host queue depth and measure IOPS, average latency and bus or bank utilization.

The driver also polls the completion queue of the controller to keep track and retire completed commands. This architecture is highly scalable and can be used in multicore architecture. Each core can have their own queue to process data and the controller is designed to support data access request from multiple cores.

D. NV-DIMM TLM2.0 simulation consideration

As shown in figure 2, the DDR4 bus interface that connects the DDR4 controller with the NV-DIMM is a b_transport call and the ONFI interface that connects NV-DIMM Controller to the 3D RRAM memory model is a four phase approximately timed nb_transport call.

The primary motivation to have b_transport interface between DDR4 and NV-DIMM is because the transactions are inherently pipelined owing to the presence of data and command queue at the front end of NV-DIMM controller. Moreover, our VP is built with the consideration of providing seamless integration with third party system/cores and bus. Most of the VP vendors provide b_transport interfaces or transactors to convert their proprietary interface to b_transport calls. The transactor between DDR4 and NV-DIMM is a simple timing annotation and transaction recording block which de-couples the model behavior from the communication interface. This approach provide easy integration with other platform without loss of granularity needed to model approximately timed systems.

The AT (nb_transport) interface provides accurate modelling of transaction pipelining and timing points between ONFI interface of the DIMM and the memory. The four phase nb_transport protocol was found to be sufficient for measurement of timing details without loss of accuracy and simulation speed.

Pragmatic use of nb_transport call and mixing AT/LT modelling style ensured considerable simulation speed while accurately capturing the functional and timing details of the model and interfaces respectively.

Both the interfaces use TLM2.0 generic payload with timing calculated according to the bit rate of the interfaces.

Formula to calculate transaction transfer time are given below:
*DDR4 (1600 MHz, 64 bits) Bus Transaction Time = Payload Size (in Bytes) * 1000/ (8 *1600);*

*ONFI4 (800 MHz, 8 bits) Bus Transaction Time = Payload Size (in Bytes) * 1000 / 800;*

To facilitate load balancing on both the interface, as inferred from the above equation, we need at least 16 ONFI channels to balance traffic handled by DDR4 bus. However, the number of ONFI channels, ONFI speed and DDR speed is configurable at compile time. User can thus analyze the impact of change in number of channels and bus speed on the overall performance (IOPS and Latency).

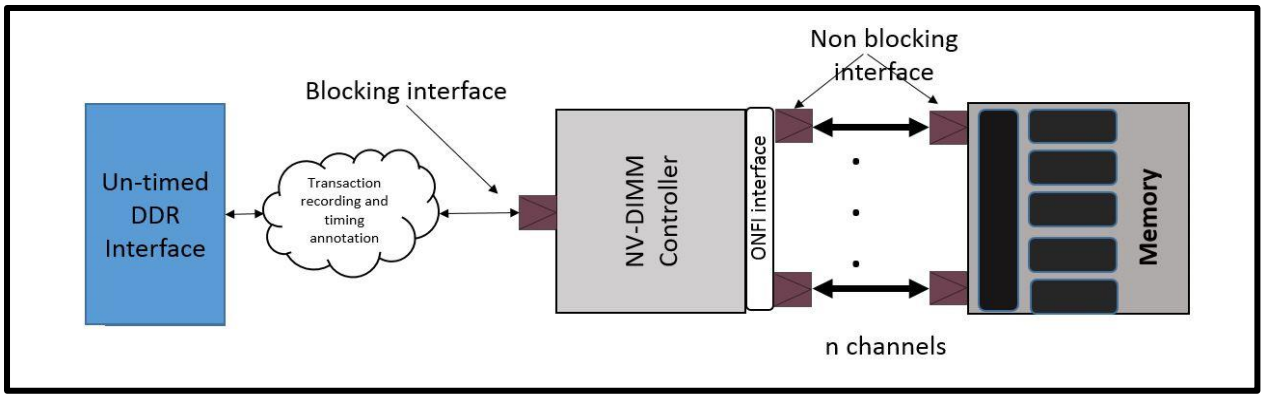

Figure 2: NV-DIMM Memory Sub-System Block Diagram

III. PCIE-NVME BASED SSD CONTROLLER

High speed SSD compared to HDD necessitates adaption and development of better high performance bus interface and streamline queuing architecture to handle high throughput data traffic. "NVMe over PCIe" fabric has been found to be a suitable solution to handle high bandwidth traffic of existing NAND Flash based SSD and also next generation 3D RRAM based SSD.

In the current project scope, we have used PCIe Gen3 x4 configuration for the simple reason that the speed supported by 3rd generation PCIe(985 MB/s per lane) lane 4 configuration can effectively balance 4 channels ONFI speed of 400 MB/s at SDR speed.

A. NVME protocol

NVM Express (NVMe) is a register level interface that allows host software to communicate with a non-volatile memory subsystem [3]. This interface and protocol is optimized for Enterprise and Client solid state drives, typically attached to the PCI Express interface. The first draft (ver. 1.0) of the standard came out in March 2011. The latest draft specification (ver. 1.2) was released in November 2014.

NVMe protocol supports highly scalable queuing interface and command sets with up to 64K command queue, each command queue can support up to 64K outstanding commands resulting in a maximum of 2^{32} outstanding commands. NVMe drivers are NUMA optimized and almost all the operating systems provide support for NVMe and PCIe drivers. NVMe supports fixed size commands which simplifies command parsing, arbitration and error handling resulting in reduction in area and power of the PCIe-NVMe controller end point [4].

Figure 3 below shows a simplified NVMe Queue Pair mapping between host and controller in multi core configuration.

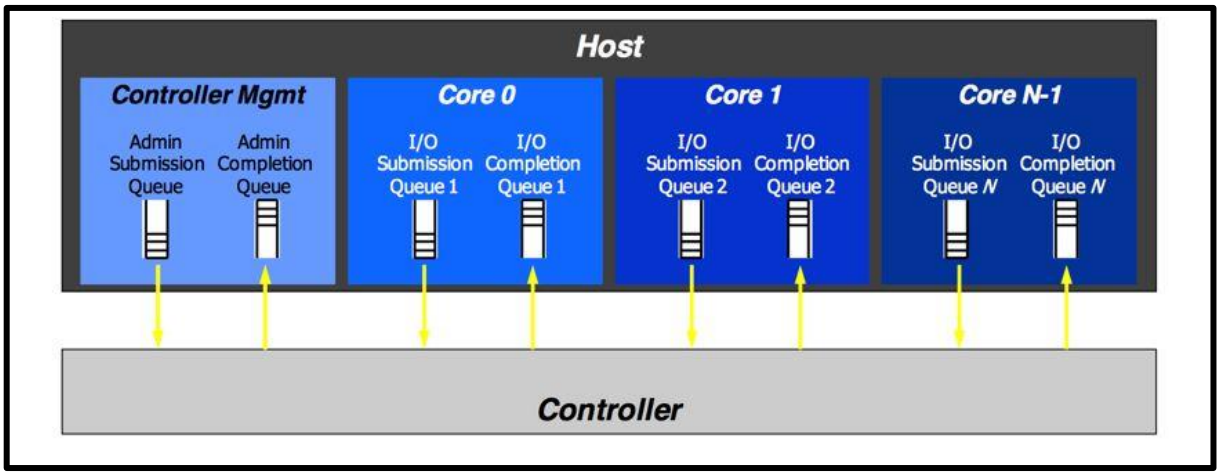

Figure 3: NVMe Queue Pair N:1 Mapping

NVMe standard also specifies efficient priority based arbitration mechanism for n:1 and n:n mapping of host with PCIe namespace. Our current work is a 1:1 mapping and future work includes scaling up the platform to n:1 configuration to enable running of cloud based and other data bandwidth intensive workloads on multiple cores.

NVMe standard is very involved specification complete with admin queue and admin command sets to identify and communicate the controller and host configurations to each other. We have abstracted out administrative protocol implementation to focus more on the performance measurement of the system at standard load. Thus initialization or housekeeping routine of the host driver and the controller has been modified to support faster initialization. Future work includes support for full command set so that the controller model can be plugged into a standard platform with full NVMe driver stack running.

B. PCIE-NVME TLM2.0 simulation consideration

PCIe-NVMe TLM2.0 simulation architecture is very similar to NV-DIMM except there is a PCIe Root Complex functional model implemented between system memory/host and the endpoint. Root Complex translates SQ/CQ and system memory reads/writes into PCIe TLP and vice versa.

PCIe root complex is register accurate in order to be compatible with off the shelf PCIe drivers [5]. Since the use model of PCIe is for performance analysis, we have only implemented Transaction Layer of PCIe, the data link layer is minimal with non- existent physical layer. PCIe explains a very elaborate flow control mechanism, we avoided use of flow control by initializing the queue size at compile time and keeping track of the queue full and empty status during run-time.

The transaction on the bus is TLM2.0 generic payload with timing calculated according to the bit rate.

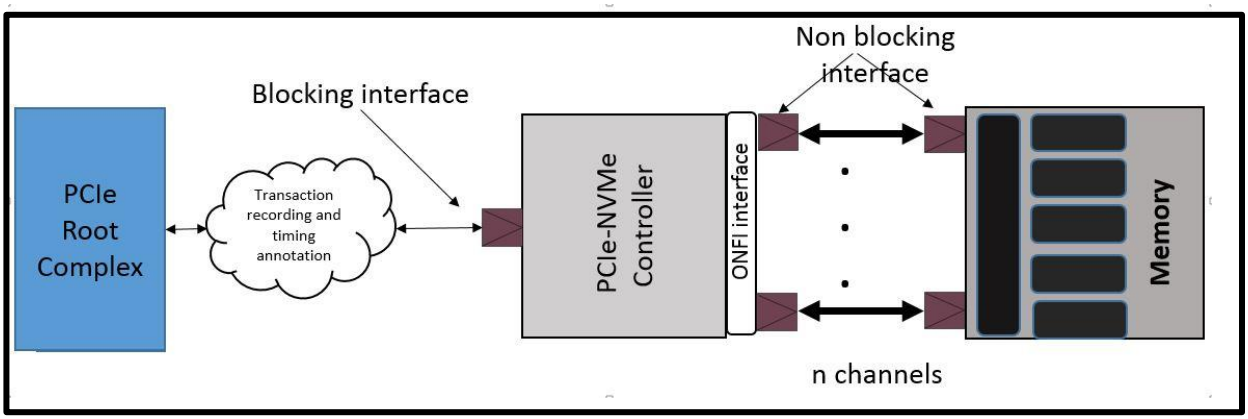

Figure 4: PCIe-NVMe based SSD sub system

The end point (PCIe-NVMe Controller) is an "NVMe over PCIe fabric" based architecture. It has support for NVMe registers. End point translates PCIe TLP carrying NVMe payload and pipelines commands and data decoded from the payload over the non-blocking interface to/from the 3D RRAM. The non-blocking interface supports four phase AT protocol.

IV. WORKLOAD AND PERFORMANCE ANALYSIS CONSIDERATION

As mentioned in section II, storage performance analysis is done by varying host IO Size and host queue depth. We carefully chose workload profile which covers all the IO data points and ran the simulation for different queue depth. We found fio and iometer to be good workload vector generation utility for our work. Future work includes use of TPC [9] and other Online Transaction Processing (OLTP) benchmark programs.

Before exploring performance analysis methodology we adapted for our work, the command and data flow of the memory sub-system has to be properly understood.

The host sends or requests transactions in a chunk of 512 bytes to a maximum of 8K. The controller stores this request in its queue and breaks down the request into smaller grain commands. The granularity of the commands is a configurable parameter. For example, if host request a read of 8K IO size, then the controller breaks down the command into smaller chunks of, let's say, 512 bytes each. Thus there are 16 sub-commands to be processed. These sub-commands are sent across the ONFI channel. If we assume configuration of 16 ONFI channels then all the 16 channels will be busy processing these sub-commands and we will get a fairly uniform read latency. It is to be noted that the memory is divided into banks and each channel can handle at most 16 banks (this is a configurable parameter too).

If we decrease the number of ONFI channels down to 8 then controller sends these 16 sub-commands into 2 groups of 8 sub commands pipelined on the 8 ONFI channels thereby increasing the overall read latency.

If consecutive accesses are on the same bank, then it will further exacerbate the read latency of the second command. But if consecutive accesses are on two different banks, due to pipelining of commands on the channel, the read latency reduces to half compared to a non-pipelined access as shown in figure 5 below.

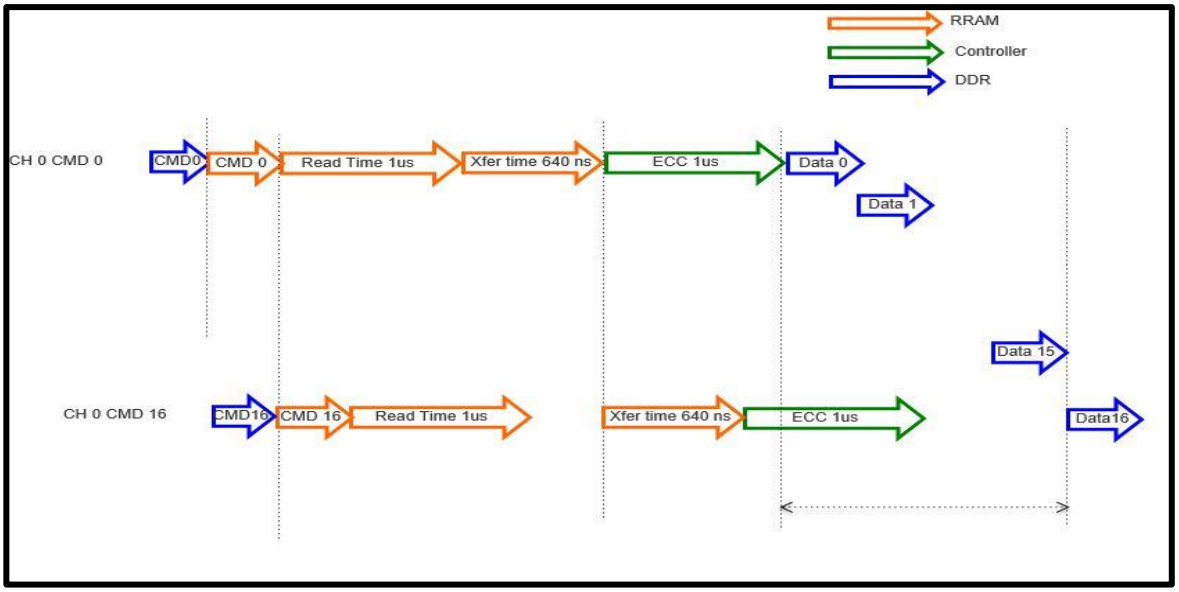

Figure 5: Example of two read commands accessing two different banks

Similarly, increase of host queue depth means more commands to be processed by the controller which effectively saturates the ONFI bus thus increasing the command latency. However, increasing queue depth results in increase in throughput (IOPS) as more work is done in a given window of time.

There are several other design trade-offs (e.g. controller buffer depth, ONFI speed, completion queue read mechanism) to be made in order to maximize performance from such a system and it is a non-trivial problem.

Results given in section VI highlights the maximum performance we achieved after carefully considering all the design trade-offs in the host, controller and the device memory.

In real world, these design trade-offs are made based on application requirement. For example, low latency application like data base access and/or financial transactions cannot have a large host queue depth but at the same time it can have reasonable IO Size if there are sufficient number of channels available to support it.

V. SYSTEM ARCHITECTURE

We leveraged open source processor core, Qbox (Qemu in a box) from GreenSocs. Qbox is a Qemu based virtual CPU which has support for i7 core. It runs Linaro Linux version 4.1.0.

Qemu [6] is a generic and open source machine emulator and virtualizer which allows engineers and developers to execute their software binaries (operating systems and applications) made for one machine

(processor and its peripherals) on another one. For example, the execution of ARM or PowerPC binaries on an x86 processor based computer. QEMU enables simulation of multiple kinds of processor cores and is based on a JIT (Just in Time compiler) code generation technology. JIT technology enables code recompilation / translation at run time. It can achieve emulation of the simulated processor core at near real time speed.

QBox [7] is an integration of QEMU virtualizer and emulator in a SystemC model. QBox or QEMU in a (SystemC) Box, treats QEMU as a standard SystemC module within a larger SystemC simulation context. SystemC simulation kernel remains the "master" of the simulation, while QEMU has to fulfill the SystemC API requirements. This solution is an open source QEMU implementation wrapped in a set of SystemC TLM-2.0 interfaces. QBox allows the powerful JIT based CPU simulations to be totally exploited within a TLM-2.0 context. Thus, this solution leverages the speed of JIT based CPU simulation with the timing accuracy of the standard SystemC TLM2.0 interfaces.

QBox is customizable. Each QBox can be customized to support processor cores from a particular vendor. We used i7 core. The decision to use i7 core for workload performance estimation was driven mainly by the predominance of Intel based system in enterprise server market. QBox comes with a tightly coupled memory in SystemC environment to boot OS kernel from. In our case, i7 core boots the kernel image from the tightly coupled memory. For faster boot-up, it is Direct Memory Interface (DMI) enabled. Thus, the execution speed of a binary in QBox is typically indistinguishable from native QEMU [7].

The kernel mounts the appropriate drivers as a device node and then workload utility is run targeting that particular node. For example, to send transaction vectors from fio to RRAM NV-DIMM, the vectors are piped to the RRAM node being initialized during the driver initialization phase of boot sequence.

Transaction recording hooks are provided at the controller and RRAM memory interfaces as well as in the host driver. Performance metrics are saved in files for post-processing of data using a GUI based tool developed internally. Figure 6 below shows the complete system developed for this work.

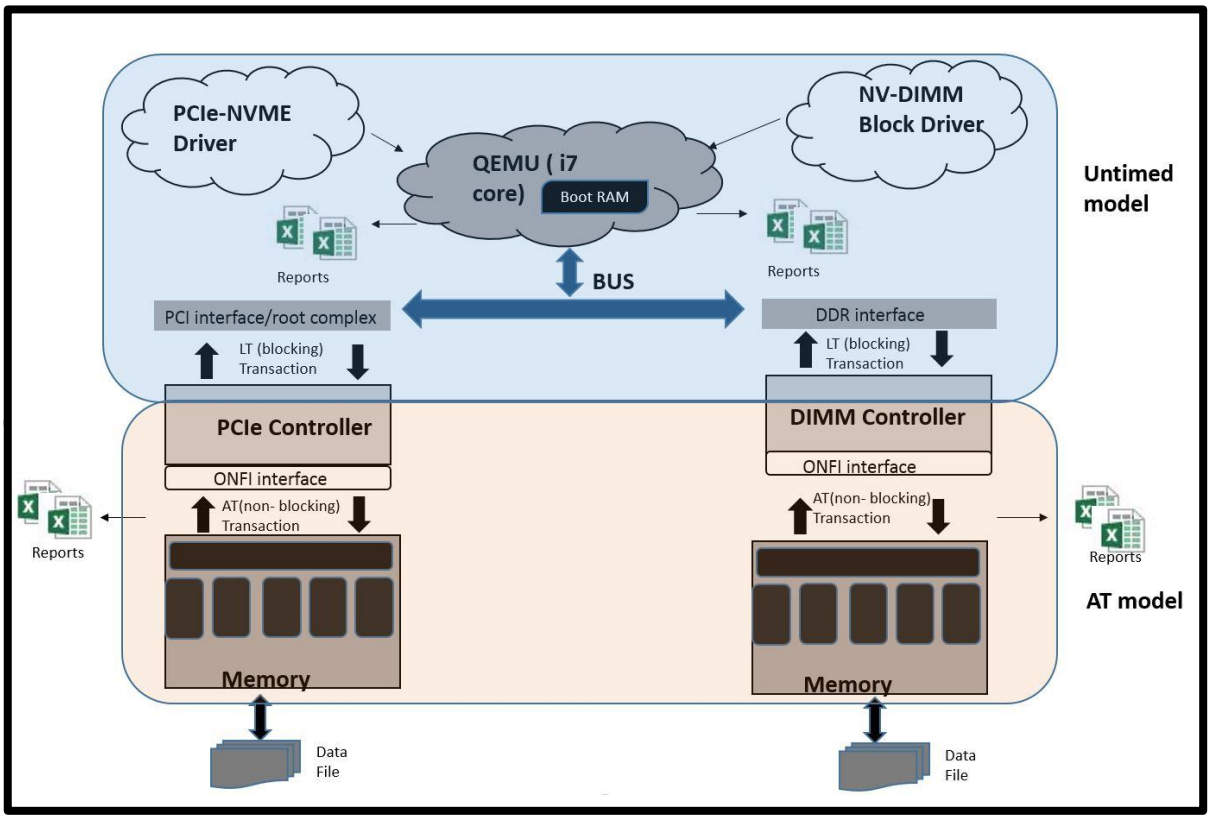

Figure 6: System Block Diagram

VI. RESULTS

The results shown below are the maximum performance numbers obtained after aggressive optimization of host queue and bus interface and carefully considering other design trade-offs mentioned in section IV. Some of the optimization technique include making sure the queues have full utilization and the bus interfaces are close to saturation under stable load. This can be achieved by tweaking the controller queue depth and bus speed. The performance numbers are based on running 100% Sequential LBA Reads.

Performance displayed here are measured under different host Queue Depth (QD) and memory access IO Size of 512 bytes and 4K.

A. NV-DIMM Controller Performance Results

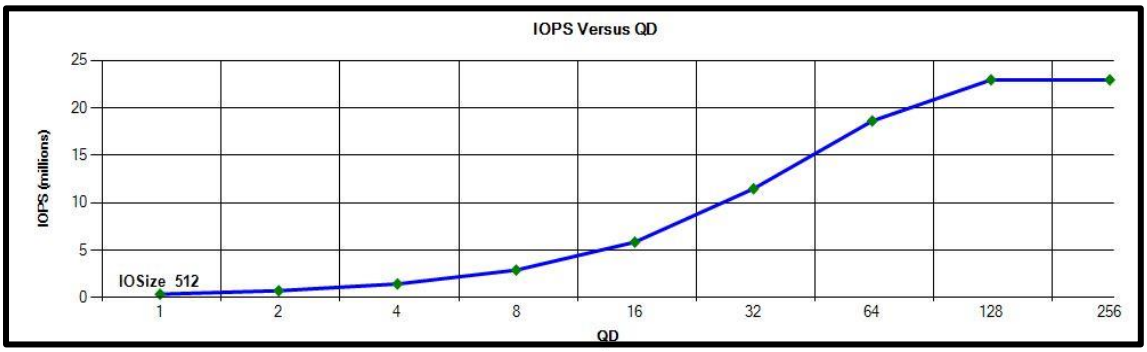

Figure 7: IOPS versus QD (IO Size = 512 Bytes)

IO Size = 512 Bytes:

IOPS (minimum) = 370 K at QD of 1
IOPS (maximum) = 23 million at QD of 256

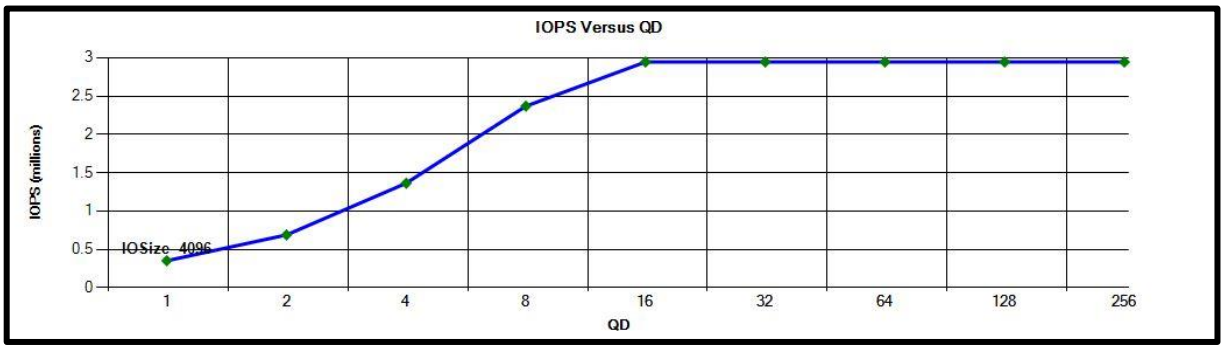

Figure 8: IOPS versus QD (IO Size = 4 K Bytes)

IO Size = 4 K:

IOPS (minimum) = 351.86 K at QD of 1
IOPS (maximum) = 2.944 million at QD of 256

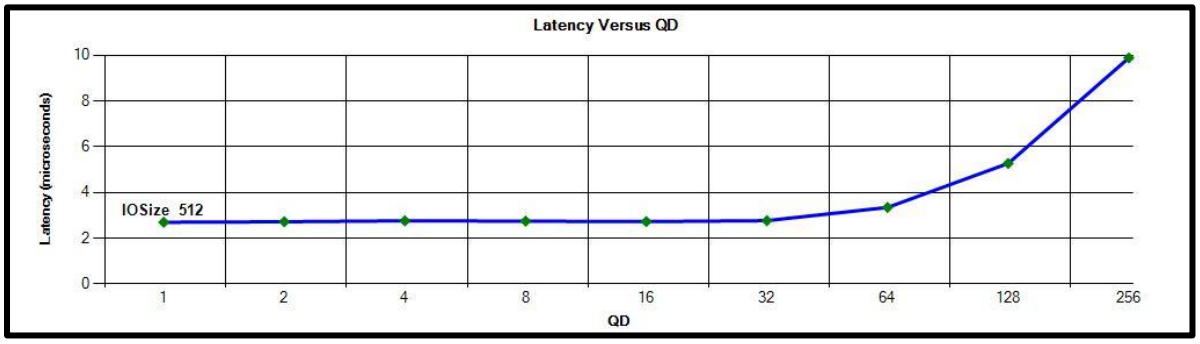

Figure 9: Average Latency versus QD (IO Size = 512 Bytes)

IO Size = 512 Bytes:

Average Latency (minimum) = 2.702 microseconds at QD of 1
Average Latency (maximum) = 9.89 microseconds at QD of 256

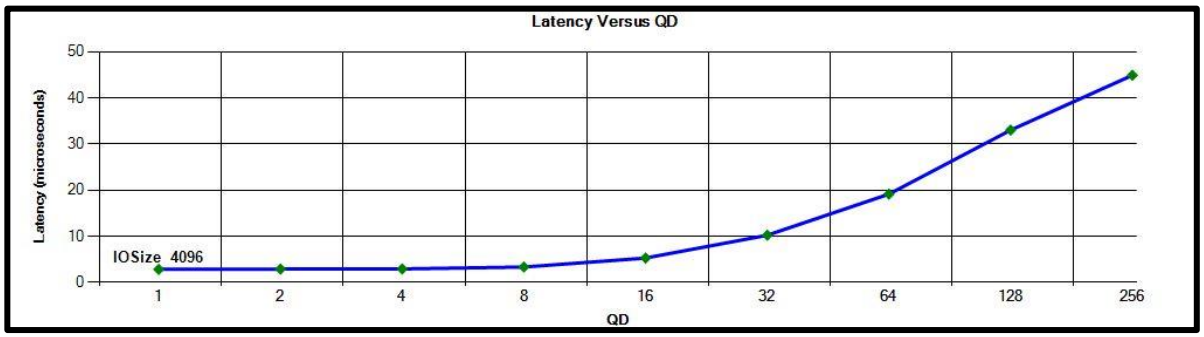

Figure 10: Average Latency versus QD (IO Size = 4 K Bytes)

IO Size = 4 K:

Average Latency (minimum) = 2.842 microseconds at QD of 1
Average Latency (maximum) = 44.928 microseconds at QD of 256

B. PCIe-NVMe Controller Performance Results

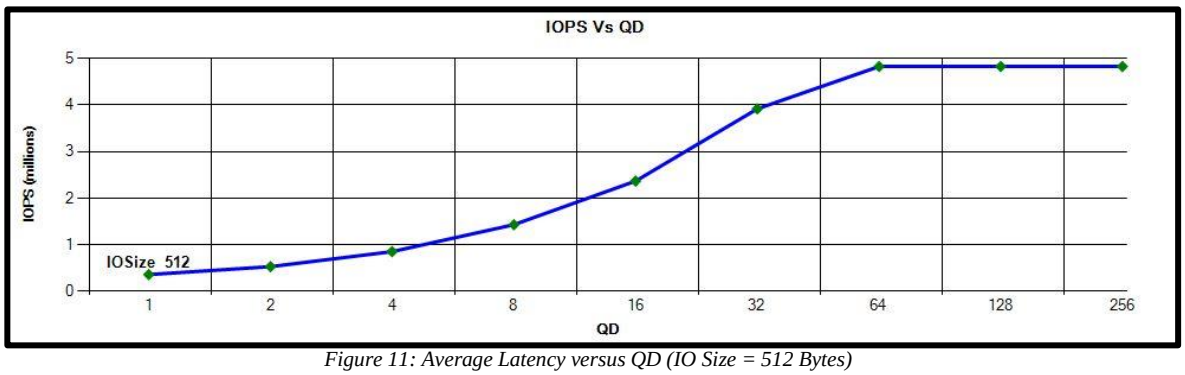

Figure 11: Average Latency versus QD (IO Size = 512 Bytes)

IO Size = 512 Bytes:

IOPS (minimum) = 360.968K at QD of 1
IOPS (maximum) = 4.83 million at QD of 256

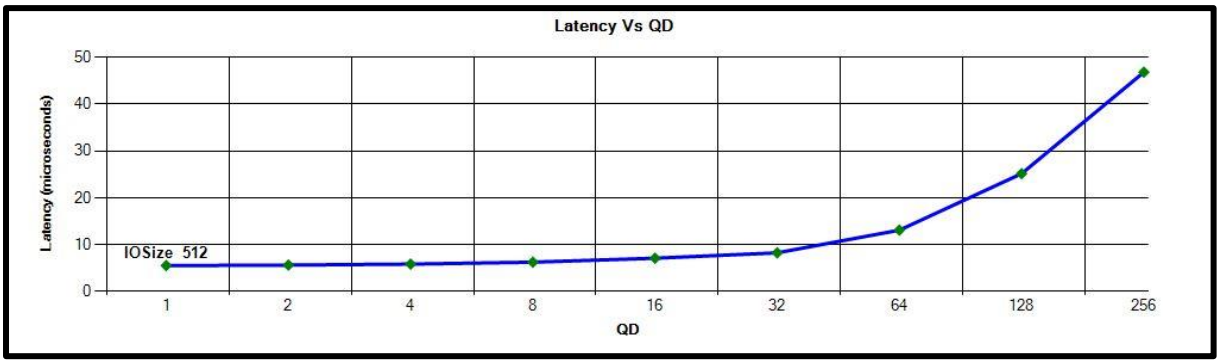

Figure 12: Average Latency versus QD (IO Size = 512 Bytes)

IO Size = 512 Bytes:

Average Latency (minimum) = 5.537 microseconds at QD of 1
Average Latency (maximum) = 46.8403 microseconds at QD of 256

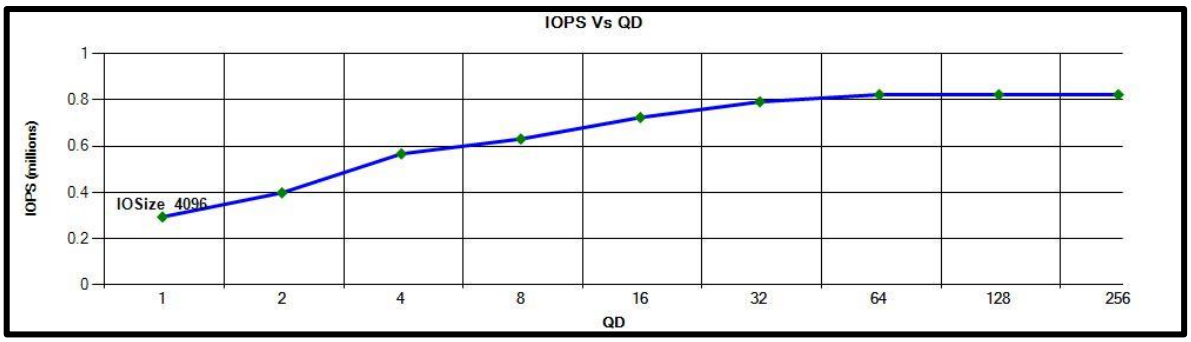

Figure 13: IOPS versus QD (IO Size = 4K Bytes)

IO Size = 4 K:

IOPS (minimum) = 294.31 K at QD of 1
IOPS (maximum) = 823.029 K at QD of 256

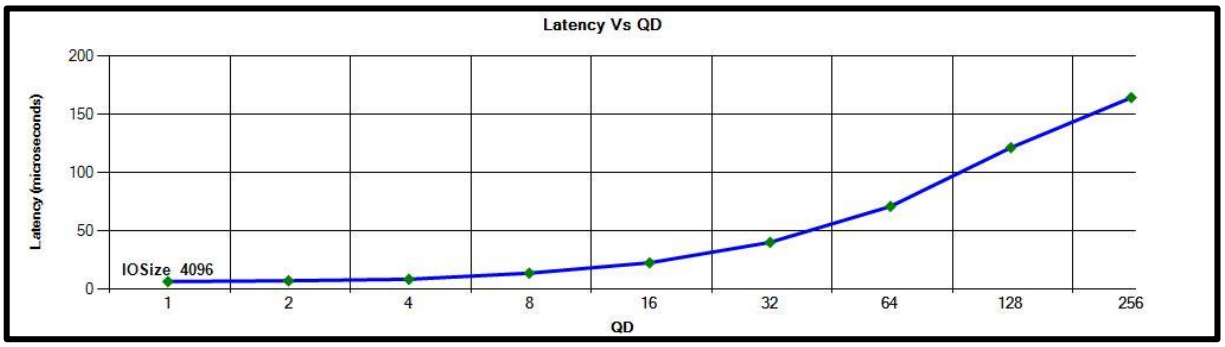

Figure 14: Average Latency versus QD (IO Size = 4K)

IO Size = 4 K:
Average Latency (minimum) = 6.78 microseconds at QD of 1
Average Latency (maximum) = 164.46 microseconds at QD of 256

C. Accuracy of Results

The performance figures obtained have been found to be more than 91% accurate with respect to spreadsheet based analysis. The numbers for NV-DIMM is close to 96%. We are still working on optimization of NVMe-PCIe sub-system to increase its accuracy. It is now around 91% accurate.

VII. CONCLUSION

Initial testing with fio and iometer suggests that the performance models developed has an accuracy of more than 91%. We are further optimizing the models, especially, PCIe-NVMe to increase the accuracy.

Future work includes running full application workload and standard OLTP benchmark on the system and comparing the results with comparable NAND Flash based storage system.

Future work also includes integration of our models with GEM5 [8] simulator for fine grained analysis of the system. This will enable system architect to perform detail analysis of cache configuration, memory sub-system and its impact on the performance numbers at full load.

We will also compare the results with the RTL currently under development.

ACKNOWLEDGEMENT

Mehdi Asnaashari, Chief System Architect, Crossbar Inc. for his valuable guidance and mentoring.

Anil Kamboj, Arvind Kumar and O.P Srivastava, HCL Technologies Pvt. Ltd. for their valuable support.

REFERENCES

[1] Crossbar Inc. (www.crossbar-inc.com).

[2] Nathan Edwards, “Theoretical vs. Actual Bandwidth: PCI Express and Thunderbolt” (http://www.tested.com/tech/457440-theoretical-vs-actual-bandwidth-pci-express-and-thunderbolt).

[3] NVMe Express Specification, Revision 1.2, 2014.

[4] Peter Onufryk, “How the Streamlined Architecture of NVM Express Enables High Performance PCIe SSDs”, Flash Memory Summit 2012, Santa Clara, CA.

[5] Mike Jackson, Ravi Budruk,”PCIe Express Technology: Comprehensive Guide to Generations 1.x, 2.x and 3.0”, pp. 170 – pp.192.

[6] http://wiki.qemu.org

[7] Guillaume Delbergue, Mark Burton, Frederic Konrad, Bertrand Le Gal, Christophe Jego, “Qbox:an industrial solution for virtual platform simulation using QEMU and SystemC TLM 2-0”, pp. 1, pp. 2.

[8] http://www.gem5.org

[9] www.tpc.org