ConTutto – A Novel FPGA-based Prototyping Platform Enabling Innovation in the Memory Subsystem of a Server Class Processor

Bharat Sukhwani
IBM T. J. Watson Research Center
1101 Kitchawan Rd,
Yorktown Hts., NY
bharats@us.ibm.com

Thomas Roewer
IBM T. J. Watson Research Center
1101 Kitchawan Rd,
Yorktown Hts., NY
roewer@us.ibm.com

Charles L. Haymes
IBM T. J. Watson Research Center
1101 Kitchawan Rd,
Yorktown Hts., NY
haymes@us.ibm.com

Kyu-Hyoun Kim
IBM T. J. Watson Research Center
1101 Kitchawan Rd,
Yorktown Hts., NY
kimk@us.ibm.com

Adam J. McPadden
IBM
1000 River St.,
Essex Junction, VT
mcpaddea@us.ibm.com

Daniel M. Dreps
IBM
11400 Burnet Rd,
Austin, TX
drepsdm@us.ibm.com

Dean Sanner
IBM
3605 Hwy 52 N,
Rochester, MN
dsanner@us.ibm.com

Jan Van Lunteren
IBM Research
Säumerstrasse 4,
Rüschlikon, 8803, CH
jvl@zurich.ibm.com

Sameh Asaad
IBM T. J. Watson Research Center
1101 Kitchawan Rd,
Yorktown Hts., NY
asaad@us.ibm.com

ABSTRACT

We demonstrate the use of an FPGA as a memory buffer in a POWER8® system, creating a novel prototyping platform that enables innovation in the memory subsystem of POWER-based servers. Our platform, called ConTutto, is pin-compatible with POWER8 buffered memory DIMMs and plugs into a memory slot of a standard POWER8 processor system, running at aggregate memory channel speeds of 35 GB/s per link. ConTutto, which means "with everything", is a platform to experiment with different memory technologies, such as STT-MRAM and NAND Flash, in an end-to-end system context. Enablement of STT-MRAM and NVDIMM using ConTutto shows up to 12.5x lower latency and 7.5x higher bandwidth compared to the respective technologies when attached to the PCIe bus. Moreover, due to the unique attach-point of the FPGA between the processor and system memory, ConTutto provides a means for in-line acceleration of certain computations on-route to memory, and enables sensitivity analysis for memory latency while running real applications. To the best of our knowledge, ConTutto is the first ever FPGA platform on the memory bus of a server class processor.

CCS CONCEPTS

- **Hardware → Emerging technologies →** Analysis and design of emerging devices and systems;
- **Computer systems organization → Architectures → Other architectures →** Reconfigurable computing;

KEYWORDS

FPGA, near-memory acceleration, non-volatile memory

ACM Reference format:

B. Sukhwani et. al. 2017. ConTutto – A Novel FPGA-based Prototyping Platform Enabling Innovation in the Memory Subsystem of a Server Class Processor. In *Proceedings of The 50th Annual IEEE/ACM International Symposium on Microarchitecture, Boston, MA, USA, October 14-18, 2017 (MICRO'17),* 12 pages.

https://doi.org/10.1145/3123939.3124535

MICRO-50, October 14–18, 2017, Cambridge, MA, USA

ACM 978-1-4503-4952-9/17/10…$15.00
https://doi.org/10.1145/3123939.3124535

1 INTRODUCTION

Modern computer server systems, such as IBM® POWER® Enterprise Servers, are complex machines comprising of multiple processor cores (packaged in one or more processor sockets), memory hierarchy, storage, networks, and peripherals. A software

stack, ranging from low-level machine firmware, to hypervisor, operating system, virtual machines, and applications, provides the necessary programmable services for Enterprise clients. Prototyping of future server systems is an increasingly difficult task, particularly when considering the system-level interactions between new hardware devices, such as new memory technologies, and software. Many approaches have been explored for modeling and prototyping of computer server systems, including full software models [1] [2], and FPGA-based system prototypes [3] [4]. Software-only models, by necessity, abstract away many hardware details and therefore make certain assumptions to approximate its behavior. Full-system FPGA-based prototypes, while providing cycle-accurate behavior, can become very complex, and hard to maintain.

This paper presents a novel hybrid approach, specifically focusing on the memory subsystem of a POWER-based Enterprise Server. We demonstrate the use of an FPGA as a memory buffer in a POWER8 system, creating a novel prototyping platform that enables innovation in the memory subsystem of POWER-based servers. Our platform is pin-compatible with POWER8 buffered memory DIMMs (CDIMM) and plugs into standard DMI slots of a POWER8 processor system. As will be described in more detail throughout the paper, we replace one or more of the memory buffer ASICs in a POWER8 server system with our FPGA-based memory buffers running at the same interface speeds, creating a unique insertion point of the FPGA in the system. Due to its flexibility, the FPGA allows us to experiment with multiple aspects of the memory subsystem in a full system context. The main contributions of this paper include:

1. A first-of-a-kind hybrid server prototype including FPGA devices along the path between the processor socket and main memory, providing the FPGA with direct access to processor's traffic to and from main memory.
2. Full system prototype with new memory technologies in addition to DRAM, which include STT-MRAM and NVDIMM-N.
3. Analysis of end-to-end system performance as a function of the latency to memory, carried out in a server-class system, running real software stack and applications.
4. Experiments with near-memory application acceleration.

The remainder of this paper is organized as follows: Section 2 describes the memory organization of a POWER8 server system, showing its unique differential memory interfaces (DMI) which are key to enabling this work. Section 3 provides details on the ConTutto FPGA board architecture, logic design, and firmware. Section 4 shows the use cases for a ConTutto based system to-date, including 1) analyzing the effect of memory latency on system performance for specific applications, 2) interfacing POWER8 to STT-MRAM and NVDIMM-N, and 3) application acceleration near memory using ConTutto. Section 5 concludes the paper.

2 POWER8 MEMORY ORGANIZATION

2.1 Overview

Most current server class processors employ "direct attached memory" wherein the memory controller and DDR electrical interfaces are fully embedded on the microprocessor chip and the microprocessor chip interfaces directly to the memory. POWER8 architecture, on the other hand, utilizes a separate memory buffer ASIC, also known as Centaur, that implements the memory controllers and uses its unique high-speed Differential Memory Interface (DMI) links to communicate between the processor and the Centaur chips using a memory-agnostic protocol.

Figure 1 shows the memory organization of a POWER8 processor [5]. The processor supports 8 high-speed DMI channels, each channel consisting of multiple links and each link running at up to 9.6 Gb/sec [6]. Each of the eight DMI channels connects to a Centaur memory buffer, which in turn has four DDR memory ports, providing a total of 32 DDR ports and achieving 410 GB/sec peak memory bandwidth and 230 GB/sec sustained bandwidth. This memory organization allows for up to 1 TB of memory capacity per fully configured processor socket. The memory buffer ASIC also contains 16 MB on-board cache to support prefetching and improve system performance.

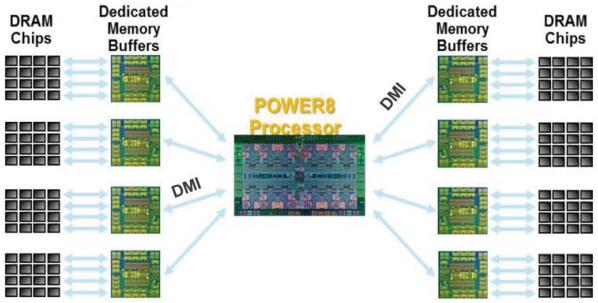

Figure 1. POWER8 memory organization utilizing DMI channels and memory buffer chips

The use of high-speed DMI links to connect to separate memory buffer chips not only allows the POWER8 processor to provide much higher bandwidth and memory capacity compared to direct-attached architectures, where the number of memory ports and hence the memory capacity and bandwidth is limited by the number of pins available on the processor chip, it also provides enormous flexibility. Direct attached architectures offer almost no flexibility in terms of communication between the processor and the memory; communication down the DDR channel must be deterministic and obey a strict set of JEDEC-specified timing commands. The POWER8 DMI protocol, on the other hand, is a packet based handshake protocol which allows for variable latencies between the request from the processor and the response from memory. This flexibility allows us to introduce an FPGA on the memory channel, the first ever in a server class processor, and enables us to experiment with different memory technologies in an end-to-end system running the full operating system and application stack.

2.2 DMI Memory Channel

The DMI channel is composed of differential unidirectional signals running at up to 9.6 Gb/sec. The DMI interface is high bandwidth clock forwarded architecture that has superior jitter tolerance characteristics compared to CDR (clock data recovery) designs. The channel loss is in the short reach category with up to 21dB s12 pad to pad at Nyquist.

The downstream link, from the processor to the memory buffer, consists of 14 data/command signals, plus 4 extra signals for clocking, sparing and calibration [6]. The upstream link, from the memory buffer to the processor, consists of 21 data signals and 4 support signals. The operations on the DMI link are performed on 128B cache line boundaries and the primary memory commands include 128-byte cache line read and write operations, along with partial 128-byte cache line write operations (read-modify-write). As stated before, communication on the DMI link is performed using a packet-based protocol. Commands and memory store data are interspersed within synchronous packets, four of which constitute a frame. Owing to the difference in the number of upstream and downstream signals, the upstream and downstream frames use different formats.

2.3 DMI Protocol

The DMI protocol is a packet-based handshake protocol that allows for variable latencies between the request from the processor and the response from memory. This handshake allows for flexibility in the response time of the memory and is a key attribute to enable the insertion of an FPGA in lieu of the memory buffer ASIC, which by necessity introduces additional latencies.

Broadly speaking, the DMI protocol consists of two levels of handshake, one for ensuring the correct transmission of packets in both directions and the other to establish a request-response relationship for the commands issued by the processor, wherein each command is composed of one or more packets. In other words, there is a tight loop with a continuous flow of packets and corresponding acknowledges, embedded in a longer loop of memory commands and responses. For the packet loop, both upstream and downstream frames are protected with strong cyclic redundancy check (CRC) for error detection, as well as contain a sequentially increasing ID (Sequence ID) to identify any missing frames. Once verified for correctness, each received frame is acknowledged (by the processor for the upstream frames/memory buffer for the downstream frames) by inserting the ACK bit into a frame being transmitted in the opposite direction. Thus, under normal operation, each frame is expected to contain an ACK bit corresponding to a previously transmitted frame. A missing ACK triggers automatic re-transmission (replay) of packets for error recovery.

To facilitate proper error recovery, a Frame Round Trip Latency (FRTL) is calculated during channel initialization, both by the processor and the memory buffer. FRTL is determined by transmission of frames with specific signatures and computing the latency between two such frames. Upon error during normal transmission, denoted by a missing ACK, this FRTL value is used by the transmitter to determine where to start the re-transmission of frames; no explicit frame ID of the erroneous frame needs to be communicated by the receiver back to the transmitter. Note that this dynamic determination of the actual FRTL value as opposed to using strict, pre-known latencies allows for the FPGA to be used in place of the Centaur ASIC. The processor, however, has a maximum tolerable FRTL value and the latency through the FPGA must be lower than that. We discuss the implications of this in detail in section 3.3

The second level of handshake is for the commands being issued by the processor to the memory buffer (note that the memory buffer is a slave and cannot issue commands to the processor). Each command is issued with an accompanying tag. The processor maintains thirty-two tags to identify the commands in flight. For the write and read-modify-write commands, the downstream data is paired with the appropriate command using the tag number. Similarly, data from the memory in response to a read request is paired with the tag of the original request. Finally, a done tag is also issued by the memory buffer to the processor, indicating that the command issued with that tag has been completed and that tag can be reused for subsequent commands. Since the number of tags maintained by the processor is fixed, for the FPGA-based design to not throttle the processor, the latency of response from the FPGA must not be so high that the processor cycles through all the tags and waits for a done to arrive before being able to issue another command. This would not only increase the latency to memory, it would also reduce the effective throughput. Thus, keeping the round-trip latency low is a key consideration of our design.

3 CONTUTTO

3.1 Overview of ConTutto

ConTutto is a custom-built FPGA card that is pin-compatible with POWER8 memory buffer DIMMs (CDIMM) and plugs into standard DMI slots of a POWER8 processor system. One or more CDIMM in a Power8 S814 / S824 server system [7] can be replaced by ConTutto cards (see Figure 2).

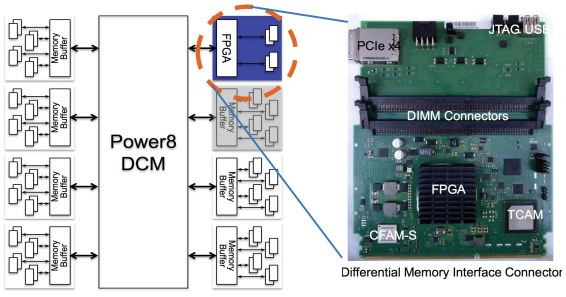

Figure 2. POWER8 memory sub-system with ConTutto

In terms of the physical form factor, however, a ConTutto card is larger than a CDIMM and plugging a ConTutto in a DMI slot blocks the adjacent DMI slot, thereby effectively forcing removal of two CDIMMs. Also, due to the plug-rules of POWER8

memory system, a ConTutto card can be plugged only in specific DMI slots. Multiple CDIMMs can be replaced by ConTutto cards, however, and we have tested system configurations with one ConTutto card and six CDIMMs as well as two ConTutto cards and four CDIMMs.

3.2 ConTutto Card Design

The block diagram of the ConTutto card is shown in Figure 3. We use an Altera Stratix V A9 FPGA to implement the memory buffer functionality. At first glance, the card design looks fairly standard; connecting industry-standard DDR3 DIMMs to current-generation FPGAs is a routine design effort and there are numerous example designs to draw from. In reality, the board architecture and design presented numerous challenges while interfacing the FPGA to the high-speed DMI links of POWER8 processor and making the card compatible with the processor's ecosystem. As mentioned before, our work is the first to attach an FPGA to a high-speed memory bus of a server class processor and the design required adapting some unconventional approaches. A noteworthy mention here is the SRC MAP system [8] where an FPGA and CPU in a system have access to shared common main memory via a network switch. While the SRC system allows the FPGA and CPU to access the same memory region, ConTutto inserts an FPGA directly on the path to processor's main memory, thereby providing the FPGA access to CPU's memory traffic and allowing fine-grained acceleration and experimentation opportunities not possible with the SRC architecture. Moreover, the SRC system is limited to DRAM whereas ConTutto enables different memory technologies.

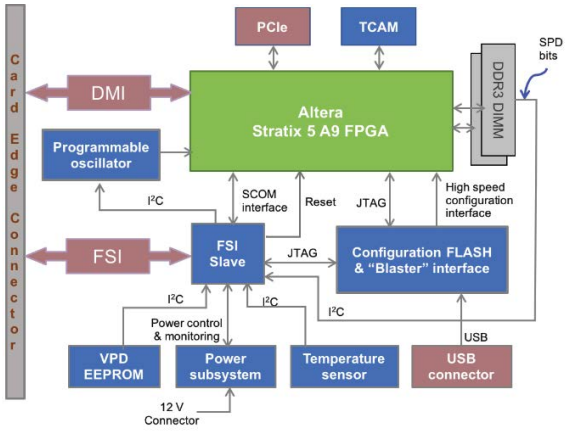

Figure 3. Details of the ConTutto card

The functional blocks on the ConTutto card can be roughly grouped into four distinct categories: 1) the data path from POWER8 processor, via the DMI links, to the DDR3 DIMMs, 2) support functions to allow ConTutto to operate in a POWER8 machine, 3) power and clocking to support the FPGA, and 4) extra blocks we added to allow future expansion and more complex machine level connectivity.

The main data path from the processor to the memory consists of the POWER8 DMI links interfacing to the high-speed transceivers on the FPGA, followed by the memory controllers on the FPGA interfacing to the DDR interface. The card contains two standard DDR3 DIMM connectors which can be populated with industry standard DIMMs.

Interfacing the FPGA to the DMI link requires special handling that's different from CDIMM. Conceptually, DMI is a full-duplex source-synchronous interface whereby the clock is sent from the data source to the data sink along with the data and this incoming, or "forwarded" clock is used to sample and latch the data. This creates an exact relationship between the sampling clock and the data, thereby in essence ensuring that there is little to no clock jitter. On ConTutto, we are forced to deviate from this data recovery mechanism; on the FPGA, the only interfaces with the capability to handle the high data rates are the high-speed serial channels (transceivers) which do not use a separate signal for clock but instead recover the sample clock from the incoming data itself. This makes our design operate the DMI interface in a non-symmetric manner; data flowing from ConTutto back to the processor is sampled with the explicitly transmitted clock, but data flowing from the processor to the FPGA is sampled with a clock recovered from the data. The source-synchronous "forwarded" clock from the processor, however, is used to feed the clock recovery PLL inside the FPGA, the output of which is used for the data recovery. This asymmetry requires careful control of all the jitter tolerances in the clock path to allow accurate and consistently low-error data recovery. Likewise, when the FPGA is the data source and the POWER8 processor is the sink, the clock accompanying the data is generated from a high-speed serial transmitter. The "data" on the clock channel is a sequence of ones and zeroes which the processor sees as a clock of the correct frequency, and proper phase alignment, to recover the data.

System-level enablement of the ConTutto card requires it to be compatible with the POWER8 ecosystem. All IBM POWER systems contain a low level "service processor" generally referred to as a "Field Service Processor" or FSP. It talks to the "slave" processors over a "Field Service Interface" or FSI. The purpose of this service architecture is to automatically derive the structure of the machine and configure each feature card prior to boot. It also periodically checks the correct operation of all the hardware, and recovers from errors and system faults. The service processor maintains long-term logs of faults and errors on each piece of hardware, and disables hardware that generates too many errors. As shown in Figure 3, the ConTutto card contains a service processor interface (FSI block) and supporting hardware to facilitate all the service architecture functions.

Current generation FPGAs require many ancillary voltages and on ConTutto, these are generated locally via a combination of high efficiency switching power regulators and low drop out (LDO) linear power regulators. The former supplies the high current required for core logic on the FPGA and the digital I/O blocks. The latter are used for the quiet analog voltages needed for the high-speed serial channels. All the local voltages are derived from a single bulk 12 volt power rail. We use the 12 volt GPU power connector available on POWER8 systems to supply the

main voltage. Both monitoring and control of the numerous voltage supplies are passed back to the service processor via the FSI slave and service interface. During start up, the service processor is responsible for maintaining the proper time sequencing of the voltage rails in accordance with the FPGA device power sequencing guidelines.

The clock structure on ConTutto is split into two parts. A free running crystal is used to configure the FPGA from a high speed FLASH. All the remaining clocks have to be synchronous to the rest of the POWER8 system and are derived from the source synchronous clock coming over the DMI interface. The "programmable oscillator" block shown in Figure 3 is in practice, a narrow band PLL with good jitter rejection and generates all the operational FPGA clocks.

Finally, the PCIe and TCAM blocks are included in the ConTutto design to allow for future experimentation. The TCAM is a ternary CAM, which could be potentially used to contain routing tables or tag entries on a data cache or for the acceleration of other applications requiring look-up. The PCIe interface could be potentially used for direct memory-to-memory transfers between ConTutto cards without burdening the POWER8 memory bus.

3.3 FPGA Logic for ConTutto

In the case of a CDIMM, the memory controller and supporting logic is implemented in Centaur. For ConTutto, the functionality of the Centaur ASIC such as the DMI protocol handling, packet parsing, command handling and the actual memory controller is implemented in the FPGA fabric. The FPGA design replicates a large subset of the functionalities of the Centaur chip and includes support for a variety of read, write, and read-modify-write commands from POWER8 via the DMI memory link. Many important features of Centaur such as the cache and other auxiliary functions, however, have been omitted in the FPGA design for simplicity. In this regard, the FPGA and its performance is not representative of that of the Centaur chip and must not be compared to it. The FPGA, however, contains all the logic necessary for it to act as a DRAM memory controller buffer for the POWER8 processor, albeit with a higher latency than Centaur, as one would expect.

Figure 4 shows the high-level block diagram of the FPGA logic implemented for the base configuration of ConTutto. Due to the differences between the ASIC and FPGA design methodologies and their vastly different operating frequencies, the actual design on the FPGA differs significantly from that of Centaur and is optimized for FPGA implementation, with a target frequency of 250 MHz. Additionally, to achieve throughput performance on par or near that of the Centaur ASIC while running at a much lower frequency, the logic has been redesigned with increased parallelism to handle two full DMI frames per FPGA clock cycle, as opposed to 1/4th of a frame, as is done in the Centaur logic. Note that we refer to this design as base ConTutto design as it provides the bare minimum logic to enable ConTutto to replace a CDIMM, i.e. to act as a DRAM memory buffer for POWER8. As will be discussed in later sections, we extended this base design to (i) enable support for non-DRAM memory types, (ii) add variable latency to memory for application characterization, and (iii) enable ConTutto as a platform for near-memory acceleration.

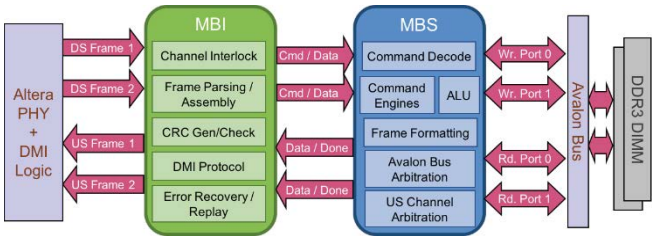

Figure 4. Block diagram of FPGA logic on ConTutto

The FPGA logic is divided into five key components: (i) DMI interface and link training, (ii) Memory Buffer Interface (MBI) logic, (iii) Memory Buffer Synchronous (MBS) logic, (iv) On-chip bus, and (v) Memory controllers.

(i) DMI interface and link training: This block handles the physical DMI interface between the POWER8 processor and the FPGA logic on ConTutto, providing bit, word and frame-level alignment and link training before any functional loads & stores can be executed.

The DMI links on POWER8 can run at link speeds of up to 9.6 GHz. When using ConTutto, we run the links at 8 GHz. Our design uses a 32:1 mux ratio between the DMI link and the fabric to bring the 8 GHz link speed down to a more manageable 250 MHz in the FPGA fabric. In other words, each downstream physical link operating at 8 GHz clock is de-multiplexed into 32 signals on the FPGA at 250 MHz clock. Comparing this to Centaur, where the mux ratio is 4:1, the FPGA handles 8x more data than Centaur in each clock cycle. With 14 downstream links, this amounts to two full frames per cycle. Similarly, 32 signals from the FPGA are multiplexed into each upstream DMI link. To minimize the link latency, we sample all 14x32 bits coming from the receiver in the core clock domain instead of using the clock crossing FIFO in the receiver macro. For link bring-up and training, we use different patterns for bit/word and frame alignment between the processor and the FPGA. Once alignment is achieved, the data gets descrambled and forwarded 2 frames/cycle to MBI. The transmit side logic accepts 2 frames every cycle from MBI, scrambles them and then sends them out across the DMI link.

(ii) Memory Buffer Interface (MBI) logic: The Memory Buffer Interface logic handles DMI protocol handshaking to ensure the proper flow of frames between the processor and ConTutto. Sequence IDs and CRC bits are used for detecting missing or erroneous frames and to trigger frame replay for error recovery. The MBI logic generates frame sequence IDs and CRC bits for upstream frames as well as verifies the CRC, sequence ID and ACK of downstream frames. Properly received frames are forwarded to the command parsing and processing logic (MBS)

for further processing while an incorrect CRC, a non-contiguous sequence ID or a missing ACK bit triggers a replay operation for error recovery.

While the MBI logic does not provide any functionality towards serving the actual loads/stores, it is critical in proper, timely and reliable operation of the DMI link. It handles the determination of frame round trip latency during channel initialization as well as frame retransmission when an error occurs. The hardware implementation of the DMI logic in the POWER8 processor puts an upper limit on the FRTL value and for the channel to be successfully trained, the round-trip latency through the memory buffer must be lower than that. Similarly, when an error occurs, the replay operation must be started within a certain time limit. This presents us with numerous challenges while designing the MBI logic on the FPGA.

Recall that the FPGA receives two full downstream frames per clock cycle, each 224 bit (28B) in size. Thus, in effect our FPGA design maintains two parallel 224 bit wide datapaths and meeting timing for 250 MHz for such a wide datapath is non-trivial. One obvious way to deal with such requirement is heavy pipelining and buffering for fan-out reduction. However, since the FPGA operates at a much lower frequency compared to the memory bus in POWER8 nest [5], each additional pipeline stage on the FPGA amounts to an increase in the FRTL value by multiple cycles on the memory bus, 8 to be precise (we run the memory bus at 2 GHz). This limits the amount of pipelining we can employ. Additionally, in ConTutto, we use a very high link-to-fabric mux-ratio of 32:1 which adds substantial latency. Significant design effort and careful tuning of FPGA placement and routing is required to meet the POWER8 maximum FRTL value. In particular, we perform two optimizations to meet this requirement. First, instead of using the receiver macro clock crossing FIFO which adds extra latency, we capture the phase-offset data from the 14 receiver channels directly in the core clock domain. Second, we reduce the initially designed 4-stage CRC logic on the FPGA down to two stages, similar to the design on Centaur, packing a lot more logic in each stage than usually done in FPGA designs. Both these changes introduce significant timing violations and complicate the timing closure process. To mitigate the impact of these design decisions, we pre-place the first stage of fabric flip-flops directly at the receiver-to-fabric interface. Also, we slightly over-constrain the next stage that feeds all 14x32 data bits into the CRC logic. With the reduced pipeline stages, combined with these fine-tunings, we are able to achieve an FRTL value that's lower than the allowable maximum set by POWER8.

The second important role of the MBI logic is frame replay (or re-transmission) in the event of an error. When a replay is requested, the MBI logic fences off the MBS and starts re-transmission of earlier transmitted frames from an internal replay buffer. This switch requires extra logic and inserts added latency in responding to the replay request from the processor, causing the processor to incorrectly identify the start of the replay operation prematurely. In other words, while removing the pipeline stages from the CRC logic and DMI PHY enables ConTutto to meet the POWER8 FRTL value, it leaves no room to add the replay logic and start the replay operation in a timely fashion. This forces the design on the FPGA to yet again deviate from the design on Centaur. Note that a number of simple solutions could be applied to solve this if we could modify the processor design to interpret the replay start in a different manner. This, however, is not possible since it is part of the POWER8 hardware design. We address this by "cheating" to the processor – we repeatedly re-transmit the last upstream frame, effectively "freezing" the flow of frames from the processor's perspective, until the FPGA is ready to switch to replay. With this workaround, the processor identifies the start of the replay operation from the actual frame being replayed and not prior to that.

(iii) Memory Buffer Synchronous (MBS) logic: The MBS logic is responsible for receiving and executing the downstream commands from the processor. It parses and decodes the frames received from the processor, executes the corresponding memory operations (reads, writes, atomic RMWs) by interfacing with the memory controllers, and returns the requested data (in case of reads) to the processor.

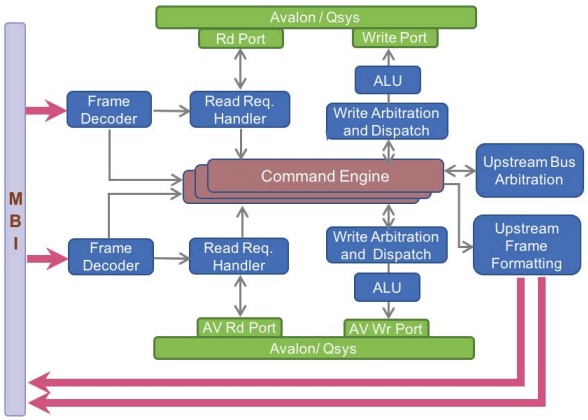

Figure 5. MBS logic on FPGA

To match the throughput of the DMI link on the POWER8 side with an 8x slower clock on the FPGA, the datapath on the FPGA is 8x wider than that on Centaur. The MBS logic contains two parallel datapaths to parse and decode two frames every cycle (see Figure 5) and the decoded memory commands are passed on to the command engines for further processing. A command engine owns the command until its completion and sends notification to the processor upon command completion. To simultaneously support multiple commands in flight, MBS maintains 32 identical command engines. For interfacing to the memory controller, we use Altera's Avalon bus, with two read and two write ports.

Read requests are issued directly to the Avalon by the frame decoder, not by the command engines. This avoids the need for arbitration for the Avalon read ports among the 32 engines. Each frame decoder uses a dedicated read port. Read ports return the data back to the command engines via the read request handler which matches the data from the memory with the issued commands using command tags. Write commands cannot be

issued directly as the data for the write command arrives in multiple frames and write data for multiple downstream commands can be interleaved. The command engines, thus, collect all the write data for their respective commands and issue write requests to the Avalon write port. Each write port serves 16 command engines and arbitration is used to grant access to one of the engines. To support atomic read-modify-write commands, data read from the memory is merged with downstream data from the processor, before being written back to the memory. The ALU used for merging is placed on the path to the Avalon write port, thereby sharing each ALU among 16 engines. For normal write commands, the ALU acts as a NOP.

Finally, arbitration among the command engines is needed for access to the upstream channel to send the data (for read commands) and command completion notification to the processor. Notice that we use a single unified arbitration unit for the upstream channel as opposed to two parallel, independent paths for downstream parsing. This is needed since depending upon the command being completed, one or both of the upstream frames may need to be assigned to a single command engine. For example, for read commands, upstream data must be sent in contiguous frames and hence both frames are assigned to a single command engine, whereas in the case of completion, the two upstream frames may contain completion notification from two separate command engines.

(iv) On-chip bus: To ensure our design is modular and flexible for future additions or changes, MBS connects to the memory controllers via the Altera Avalon bus. MBS has 2 read- and 2 write-ports on the bus, because it processes 2 DMI frames every clock cycle. Also, the crossing between the core- and DDR-clock domain is accomplished by the Avalon bus. Using a bus-based design as opposed to direct connections offers great flexibility, because other macros such as PCIe or USB can be added to this bus without impacting the rest of the design. Also, memory controllers for alternative memory technologies can be developed independent of the rest of the ConTutto design. We only require a compatible bus interface and the integration of the new memory controller into ConTutto is plug-and-play.

(v) Memory controllers: We have used ConTutto to enable DRAM, MRAM, and NVDIMM memory in a production POWER8 system. Supporting these different memory types mainly requires changes only to the memory controller that interacts directly with the memory devices; the design of MBS, MBI, and DMI remains largely unchanged.

For DRAM enablement, we use the soft DDR3 memory controller from Altera [9]. To enable MRAM and NVDIMM devices, we use the generated code for the DRAM memory controller as a starting point and make the necessary changes as suggested by the memory vendors. Thus, simply changing the memory controller allows us to support different memory technologies on POWER8, something that would not have been possible without the ConTutto card, short of spinning a new Centaur-like ASIC for every new memory type.

Table 1 shows the FPGA resources used by the base ConTutto system. The system uses just 43% ALMs and less than 10% block RAM, leaving a significant portion of resources for architectural exploration and in-memory application acceleration.

Table 1. FPGA resource utilization

Resource	Available	Utilized
ALMs	317000	136,856 (43%)
Registers	634,000	191,403 (30%)
M20K	2,640	244 (9%)

3.4 Firmware Support to Enable ConTutto

Enabling the ConTutto card in the POWER8 memory subsystem requires significant changes to the existing firmware. Broadly, firmware support for ConTutto is split into two facets. The first is to fit the DMI interface training sequence into the existing firmware sequence. The DMI training consist of triggering a pattern write and then checking for receipt of the data on each side. While the logic to do this is identical to Centaur, the path to get to the registers is completely different in the case of ConTutto. Instead of direct FSI access to read and write the internal chip registers (as in Centaur), ConTutto contains an FSI slave external to the FPGA and the register space inside the FPGA is accessed via I^2C. Thus, each access becomes an indirect path of FSI Slave to I^2C Master to FPGA register, requiring modifications to the existing firmware routines.

Beyond access to the FPGA registers, the auxiliary FSI slave on the card provides some additional controls which enable the firmware to control the FPGA's reset and power-on sequences independently from the rest of the system. This allows for repeated retries of the training sequence without bringing down the entire system. This is critical since link training often does not complete successfully in a single try and bringing down the entire system would be prohibitively slow. This path is also used for presence detection and differentiation from the standard CDIMMs that are plugged into the system, allowing for a mixed configuration of ConTutto and regular CDIMM in the same system. The final use of the external FSI slave is to directly read the SPD (serial presence detect) on the DIMMs plugged into ConTutto, which is critical for detecting and controlling the NVDIMMs.

The second challenge for implementation is fitting the ConTutto into the memory map. Depending on the type of memory present, special rules must be in place – specifically when the memory is nonvolatile. When ConTutto is booted with DRAM, the memory can be treated just like regular memory and sorted to form a contiguous memory block. However, for MRAM or NVDIMMs, these need to be placed at a non-zero location as Linux requires DRAM at the start of the memory map. Linux also needs to be able to recognize this memory as being different from standard memory, one that requires to be treated special. To do this, firmware enforces that nonvolatile memory is placed at the top of the memory map, and with flags that indicate the type

(DRAM/MRAM/NVDIMM) and whether the content is preserved. This enables Linux to detect and map this memory into the appropriate drivers.

Enabling MRAM memory in ConTutto requires special handling as the current sizes for MRAM are in the Megabyte range, but the smallest memory size supported by the POWER8 processor is 4 GB behind a DMI link. We address this by "lying" to the processor, indicating a 4 GB MRAM space, but only communicating up to Linux the actual size of the MRAM in Megabytes. Hence the hardware operates at its smallest size, but Linux would only ever touch the first few Megabytes.

4 CONTUTTO USE CASES

The primary motivation for developing the ConTutto card was its potential to be used as a hardware-based experimentation vehicle in an end-to-end system context with server class OS and applications running in real scenarios. Examples of such experiments include characterization of different memory technologies, varying memory latencies, acceleration close to memory and enablement of advance software development. To that end, we extended the base ConTutto design to support various use cases beyond its basic use as a Centaur replacement. Below we discuss three different experiments we performed using ConTutto in a POWER8 server system. All these experiments were running the full standard Linux stack utilizing either the pmem.io driver stack [10] or raw slram driver.

4.1 Variable Latency to Memory

In recent years, various new trends in memory organization have been proposed. One such model is disaggregated remote memory [11] [12] whereby a large pool of memory is maintained as a shared resource, separate from the processing nodes. Processing nodes can request memory from this pool as per their application requirements. While this model offers the benefit of resource sharing and higher utilization, it also increases the latency to memory. Understanding the effects of such increase in memory latency on end-to-end application performance is vital to knowing the viability of such models for different applications. Same applies to non-volatile memory technologies which offer the promise of persistence but at higher latencies. Our first experiment includes characterization of application performance under varying latency to memory, to help understand these effects. While we do not create a disaggregated memory server and do not address other aspects of resource sharing, our experiments capture the effects of increased memory latency.

All the variable latency experiments were performed with DDR3 DRAM. We vary the latency to memory first by using a standard CDIMM and adjusting different performance-related knobs available in it. Table 2 lists the different latency settings for Centaur used to characterize application performance. The latency to memory is the measured latency of a single memory command, averaged over multiple single commands issued from POWER8.

To characterize the effects of increasing memory latencies, we ran two sets of experiments. As the first experiment, the average time for running 29 database queries in DB2 BLU was measured on Centaur for the different latency settings shown in Table 2. We observed that increasing the latency by more than 3x, from 79 ns to 249 ns, resulted in less than 8% increase in query evaluation time.

Table 2. Memory latencies and DB2 BLU query runtime on Centaur with different settings

Configuration	Latency to memory (ns)	DB2 BLU query runtime (sec)
Centaur	79	5387
Centaur with L4 disabled	83	5451
Centaur + L4 disabled + ECC bypass disabled	116	5484
Centaur + L4 disabled + ECC bypass disabled + L2/L3 bypassed	249	5802

As the second experiment, we ran SPEC CINT2006 benchmarks on POWER8 with the different latency settings. Figure 6 shows how the SPEC CINT2006 ratios change with varying latencies to memory.

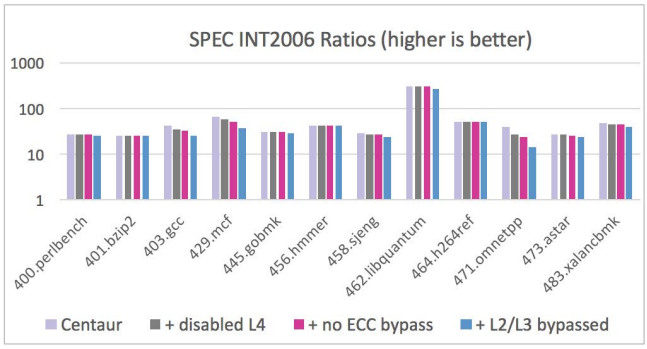

Figure 6. SPEC CINT2006 ratios with variable latency on Centaur

We further obtained additional latency range by instrumenting the ConTutto logic in the FPGA with a knob for adding latency in fixed increments. We add variable latency on ConTutto by delaying the issuance of commands to the memory by inserting delay modules between the MBS logic and the Avalon bus. Each knob position, controllable from software, adds 6 extra cycles of latency, equivalent to 24 ns. Different latency settings tested on ConTutto are shown in Table 3. The raw memory latency measurements were taken using the same routines as for Centaur latencies reported in Table 2. The measurement represents the total roundtrip latency through software, processor, caches, Power bus nest, DMI link and ConTutto. Memory latency on the same system with a single Centaur configured to match the hardware functionalities implemented in ConTutto was measured to be 293 ns, whereas the latency with the most latency-optimized Centaur configuration was measured to be 97 ns. This represents a 27% higher latency incurred by ConTutto when comparing against a Centaur with similar hardware capabilities whereas

about 280% higher latency comparing against the most optimized Centaur configuration.

Table 3. Variable latency settings on ConTutto

Configuration	Latency to memory (ns)
Centaur	97
ConTutto base	390
ConTutto + knob @ 2	438
ConTutto + knob @ 6	534
ConTutto + knob @ 7	558

Replacing a CDIMM with the ConTutto card adds significant latency to memory, which is expected due to the use of FPGA with limited Centaur functionalities as opposed to the highly optimized ASIC. Note, however, that ConTutto is not meant to be a CDIMM replacement for DRAM or to be a lower-latency solution to DRAM. It is meant to be a flexible platform to enable experimentation with different memory technologies, something not possible with existing server-class memory solutions.

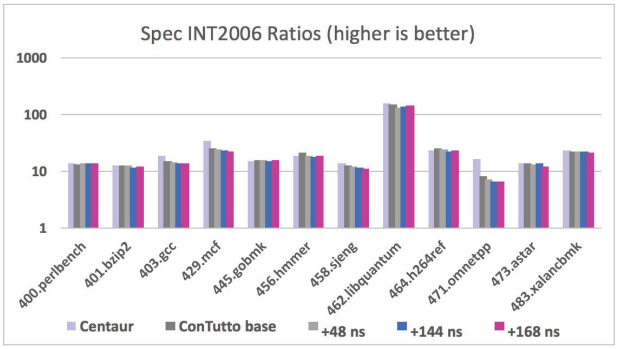

Figure 7. SPEC CINT2006 ratios with variable memory latency on ConTutto (with Centaur as baseline)

Figure 7 shows the effects of increasing memory latency on SPEC benchmark performance with the DRAM behind ConTutto being used as the system's main memory. Measurements were performed with a single ConTutto card in a POWER8 system, with a total of 8 GB DDR3 memory behind ConTutto (4 GB in each DIMM slot) and the rest of the DMI slots deconfigured. As can be seen from both Figure 6 and Figure 7, with almost 6x (600%) increase in latency to memory, about half of the applications incur less than 2% performance degradation whereas two-thirds of the applications remain under 10% degradation. For the rest, the performance degradation is in the range of 15% to 35%, with one benchmark application showing performance degradation of more than 50%. While some applications suffer higher degradation than the others, the overall performance degradation is not proportional to the increase in latency. Judging by these applications alone, a case for remote, disaggregated memory can be made, at least for a class of applications. Having said that, there can be other memory-bound applications such as graph and pointer chasing application where the performance degradation could be much higher. The effects on such computations need to be further studied and ConTutto provides a unique platform to study such effects in the full system context.

4.2 Support for Different Memory Technologies

One of the key features of ConTutto is the support for various types of memory. ConTutto is memory technology agnostic; as long as the interface supports DDR3, the backing memory cell technology could be based on resistive filaments, chalcogenide, magnetic tunnel junctions or capacitive cells, to name a few. This, however, does not imply that the memory technologies can be swapped without any changes. Supporting different memory technologies requires changes to the memory controller on ConTutto as well as firmware modifications (as discussed in section 3).

We have enabled and tested the following memory technologies on ConTutto:

(i) DRAM: DRAM is used as part of the base ConTutto design for card bring-up, testing, validation as well as for characterizing the system level impacts of variable memory latency, as mentioned in the previous sections.

(ii) STT-MRAM: Spin-Transfer Torque Magnetic RAM (STT-MRAM) is a class of non-volatile memory with higher performance, lower power consumption and higher endurance than FLASH. Endurance of non-volatile memory technologies is of significant concern when used on a high bandwidth memory bus. STT-MRAM exhibits much better endurance as compared to FLASH (see Figure 8). Its high endurance, combined with high performance, makes STT-MRAM a potential candidate for use as storage-class memory; its low-density and small capacity, however, is currently a limiting factor.

We have enabled STT-MRAM via ConTutto, demonstrating, for the first time, storage class memory as part of an enterprise system, running and accelerating storage and server applications. Using the ConTutto-enabled STT-MRAM, we have developed a persistent memory (pmem) kernel driver, guaranteeing persistence on the memory bus. The development of the kernel driver is an example of the use of ConTutto to enable early software development.

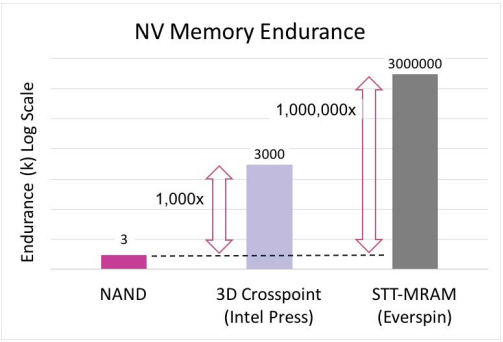

Figure 8. Endurance comparison between different non-volatile memory technologies. Source [13], [14]

Our setup consists of a POWER8 system with two ConTutto cards and a total of 1 GB of STT-MRAM. Each ConTutto card is populated with two 256 MB MRAM DIMMs. Our initial technology demonstration of MRAM used iMTJ (inline magnetic tunnel junction); we have since migrated to pMTJ (perpendicular MTJ) which shows improved power/performance characteristics.

To characterize the benefits of STT-MRAM, we ran General Parallel File System (GPFS) [15] on the above mentioned system, utilizing STT-MRAM behind ConTutto as a write cache in front of a hard disk drive to aggregate small random writes into larger sequential writes to the disk, thereby avoiding the latency hit of repositioning the drive head for each of the original small writes. Table 4 shows the configurations of the three persistent stores being compared and the measured performance in terms of input/output operations per second (IOPS); STT-MRAM on ConTutto achieves 8.3X single thread performance improvement over state of the art SSD.

Table 4. Performance comparison for different technologies while running GPFS

Technology	Size	System Interface	IOPS
Hard Disk Drive	1.1 TB	SAS	75
SSD	400 GB	SAS	15K
STT-MRAM	256 MB	DMI (Memory link)	125K

We also evaluated these technologies as well as different attach points using the FIO benchmark [16]; the IOPS and latency measurements are shown in Figure 9 and Figure 10. All measurements except the MRAM on PCIe were taken on a POWER8 system; MRAM-on-PCIe numbers are those published by the vendor. The results show that MRAM on ConTutto achieves 6.6x lower read latency and 15x lower write latency compared to FLASH-backed DRAM on PCIe (NVRAM) while providing 4.5x and 6.2x higher IOPS for reads and writes, respectively. Comparing STT-MRAM on ConTutto to STT-MRAM on PCIe, ConTutto achieves 2.4x and 5x lower latency and 1.5x and 2.2x higher IOPS for reads and writes, respectively. The results clearly demonstrate that ConTutto provides a much higher bandwidth and lower latency attach point than PCIe, even with NVMe.

(iii) NVDIMM-N: NVDIMM refers to FLASH-backed DRAM DIMMs which combine the performance of DRAM with non-volatility of FLASH. The main idea is to use DRAM for memory operations and copy the data over to FLASH when the power is removed; a backup power source such as a battery or a super-cap is used to support the copying operation. The copy is performed by the NVDIMM itself and does not need the FPGA or the CPU to stay powered up during this operation. As a second non-volatile memory on ConTutto, we enabled support for NVDIMM-N. While the memory interface is still the same as DRAM, enabling NVDIMM-N requires non-trivial firmware/BIOS support to implement the engineering wrapper to persist DRAM. The sequence of operations to be performed to persist DRAM are being standardized through JEDEC for DDR4; the sequence is vendor specific in the case of DDR3.

The performance of NVDIMM enablement on ConTutto using FIO to benchmark is shown in Figure 9 and Figure 10, alongside other persistent memory technologies and attach points. Compared to NVDIMM on PCIe (NVRAM), ConTutto provides 7.5x and 12.5x latency improvement for read and write operations, respectively as well as achieves 6.5x and 7.5x higher IOPS. The improvement is even greater when compared to FLASH on x4 PCIe.

Supporting non-volatile memory on ConTutto not just requires modification to the memory controller, but also changes to the command engines to ensure that commands in flight are handled in a deterministic manner. To be precise, the persistent memory controller in the software stack requires support for flush and sync commands to ensure that outstanding commands have been written to memory. We extended the MBS logic to add a special flush command to accomplish this. Note that this functionality does not exist in the Centaur ASIC; the use of FPGA provides the flexibility to add new commands. The flush command is an example of how ConTutto can be extended to perform special in-memory operations at a fine-grained level.

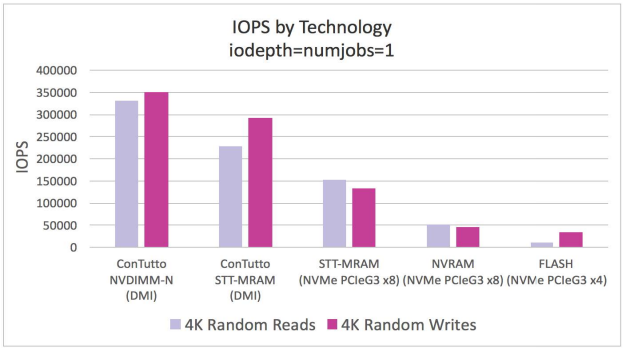

Figure 9. IOPS comparison from FIO test for different non-volatile memory technologies

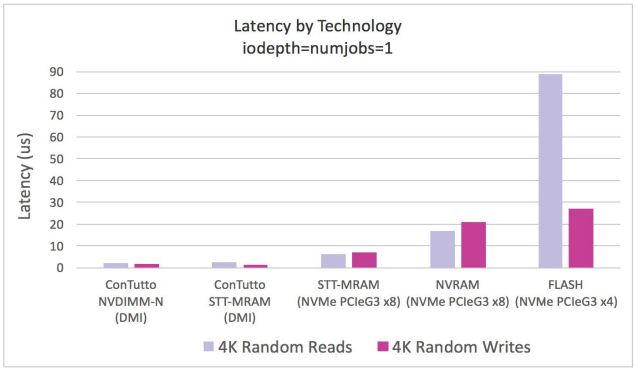

Figure 10. Latency comparison from FIO test for different non-volatile memory technologies

4.3 ConTutto for Application Acceleration

A third and important application of ConTutto is application acceleration close to memory. Since ConTutto contains an FPGA

directly on the path to, and with direct access to system's main memory, it provides an opportunity to perform certain tasks in-memory, without the processor being in the loop. This can provide substantial benefit for a variety of block acceleration applications such as in-memory databases, in-memory sort and search acceleration as well as certain fine-grained acceleration tasks such as linked-list traversal, atomic operations, etc.

Figure 11 shows an in-line acceleration unit in ConTutto whereby acceleration tasks, identified using special load/store instructions, can be handled by command engines augmented to perform special operations. These command engines can implement the required fine-grained acceleration operation (e.g. min-store, max-store, conditional swap etc.) as part of the regular ConTutto pipeline. One example of such a special command engine is the flush command discussed previously in the context of non-volatile memory support on ConTutto. Since the accelerator is in-line with the main ConTutto pipeline, it has access to the upstream DMI channel and can send direct response to the processor without the need for the processor to poll.

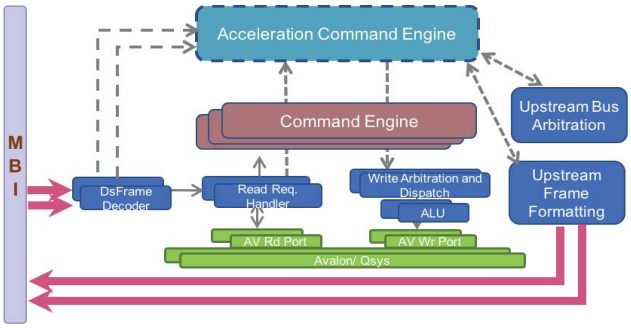

Figure 11. In-line acceleration close to memory using ConTutto

Another insertion point for an accelerator in ConTutto is shown in Figure 12 where the accelerator (attached to an Access processor that will be discussed below) appears as a special memory-mapped region on the Avalon bus. Such an attach point is more suitable for block acceleration tasks whereby the accelerator receives a control block from the processor describing the acceleration task and a range of data or memory addresses to operate on and write the results to. Store instructions targeting a buffer region inside the acceleration unit can be used to send the control block to the accelerator. Upon task completion, the accelerator writes processing status and completion information into specific fields in the control block, which can be retrieved respectively polled using load instructions on the corresponding addresses.

Enabling a block accelerator with access to the system memory requires special consideration of memory allocation, virtual memory management and cache hierarchy and support from OS level drivers. We created a device driver to enable such support on ConTutto. The details of the device driver are beyond the scope of the current paper.

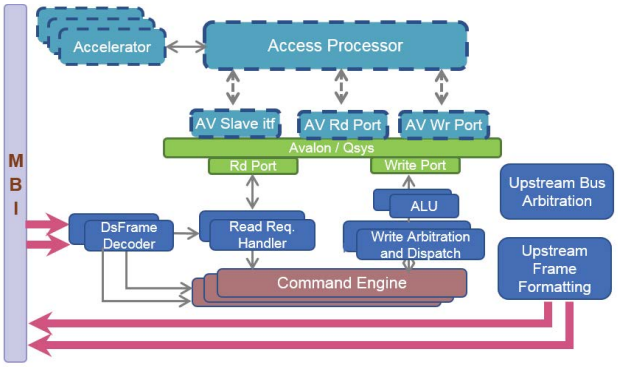

Figure 12. Block acceleration close to memory using ConTutto

In order to enable one or more accelerators to share the memory controllers with the POWER8, we use a programmable component called Access processor to arbitrate and schedule the load and store instructions to the DDR3 DIMMs, thereby supporting various schemes for allocating and distributing the available memory bandwidth between the POWER8 and the individual accelerators. The Access processor also includes a programmable address mapping scheme that allows to change the way in which addresses, and consequently data structures, are mapped on the physical storage locations in the DIMMs. To simplify the accelerator design, the Access processor can optionally issue load and store instructions to the DIMMs, including address generation, on behalf of the attached accelerators, leaving the accelerators only to deal with the actual data processing. The Access processor is programmed by loading pre-compiled executable code that is retrieved from the DDR3 DIMMs into an internal instruction memory. This is triggered by the reception of a special control block, and is performed dynamically without interrupting the base operation. The micro architecture of the Access processor has been designed as a programmable state machine to enable support for the functions described above and supports multithreading.

The programmable nature of the Access processor, in combination with its extensive control capabilities, enables to improve the performance and, to a certain extent, the energy efficiency of the accelerator operation in two ways: 1) the bandwidth and latency for accessing the data in the DIMMs can be optimized by programming the (optional) access generation, address mapping and access scheduling in a way that best matches the accelerator's pattern of accessing that data, that pattern being available at compile time or being determined during runtime by performance monitoring functions integrated into the Access processor. 2) its fine-grained control allows the Access processor to schedule the access and transfer of data (e.g., operands, results, configuration) to and from the DIMMs in an almost seamless fashion. This "just-in-time" access, transfer and processing, minimizes overall task execution times and intermediate buffering and related overhead for many applications.

The Access processor and the accelerators are implemented in 250 MHz clock domain. Each accelerator is connected to the two

DDR3 DIMM ports through the Access processor, resulting in a peak memory access bandwidth (combined for loads and stores) in the range from 10 GB/s to 12 GB/s, observed during our experiments. The details of the Access processor microarchitecture, instruction set and the process to generate the executable code is beyond the scope of the current paper and will be presented in a future paper.

Table 5 shows the results for the three functions that are accelerated on ConTutto, and are started from a main task executed on the processor running Linux:

(i) Memory copy of 1 GB block of data from one location in the DIMMs to another location,

(ii) Finding the minimum and the maximum in a series of blocks of 256M 32-bit integer values (1GB in each block),

(iii)) Calculation of 1024-point FFTs based on 8B complex 32-bit floating point samples.

Table 5. Performance of accelerated functions on ConTutto

Accelerated function	Performance (throughput)	
	ConTutto (2 DIMM ports)	Software (with CDIMMs)
Mem. Copy	6 GB/s	3.2GB/s
Min/Max	10.5 GB/s	0.5GB/s
1024-pts FFT	1.3 Gsamples/s	0.68 Gsamples/s (4 CDIMMs, 16 DIMM ports)

The min/max accelerator processes the data on-the-fly while being retrieved from the DIMMs under control of the Access processor. The FFTs are calculated in parallel on multiple FFT accelerators, in such way that, through appropriate scheduling by the Access processor, the sample and result transfers between a given accelerator and the DIMMs are overlapped with computation on the other accelerators. In this way, all three functions are capable of exploiting the full access bandwidth to the DIMMs, and, consequently, achieve the maximum possible performance.

Table 5 also lists the performance of software implementations of the same functions executed on the POWER8 using CDIMMs, with the FFT results being taken from [17]. The table shows that performing these tasks close to memory using ConTutto-based accelerators achieves 2x to 20x improvement over software, with the ConTutto FFT accelerator accessing only two DIMM ports versus 16 DIMM ports being accessed by the software implementation.

5 CONCLUSIONS

We have presented a novel prototyping platform called ConTutto that enables innovation in the memory subsystem of POWER-based servers. ConTutto provides a unique platform to perform end-to-end experiments on various aspects of system level research in a real hardware environment. In particular, we have demonstrated three use cases for ConTutto. First, using ConTutto, we enabled different emerging memory technologies such as STT-MRAM and NVDIMM and demonstrated the first ever use of storage class memory as part of an enterprise system running and accelerating storage and server applications. Secondly, we used our platform to characterize application performance under varying memory latencies. Finally, we demonstrated the use of ConTutto for acceleration close to memory. Though FPGA fabric has previously been shown to interface to the memory bus of embedded processors like in Zynq [18], to the best of our knowledge, ConTutto is the first ever FPGA platform attached to the memory bus of a server class processor, providing the highest bandwidth, lowest latency path between the FPGA and a server grade processor.

ACKNOWLEDGMENTS

This work was partially supported by the Department of Energy via Award No. B609911.

REFERENCES

[1] F. Bellard, “QEMU, a fast and portable dynamic translator”, in *Proceedings of the annual conference on USENIX Annual Technical Conference*, April 2005.
[2] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner, “Simics: A Full System Simulation Platform”, *Computer*, Vol. 35, Issue 2, February 2002 Page 50-58
[3] S. Asaad, R. Bellofatto, B. Brezzo, C. Haymes, M. Kapur, B. Parker, T. Roewer, P. Saha, T. Takken, and J. Tierno, "A Cycle-accurate, Cycle-reproducible multi-FPGA System for Accelerating Multi-core Processor Simulation”, in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, February 22–24, 2012.
[4] Q. Wang, R. Kassa, W. Shen, N. Ijih, B. Chitlur, M. Konow, D. Liu, A. Sheiman, and P. Gupta “An FPGA based Hybrid Processor Emulation Platform”, in *Proceedings of the International Conference on Field Programmable Logic and Applications*, Aug. 31st - Sep. 2nd, 2010
[5] W. J. Starke, J. Stuecheli, D. M. Daly, J. S. Dodson, F. Auernhammer, P. M. Sagmeister, G. L. Guthrie, C. F. Marino, M. Siegel, and B. Blaner, “The cache and memory subsystems of the IBM POWER8 processor”, *IBM Journal of Research and Development*, Volume: 59 Issue: 1, Jan.-Feb. 2015
[6] “POWER8 Memory Buffer User’s Manual” 22 April 2014 Version 1.1
[7] A. B. Caldeira, B. Grabowski, V. Haug, M. Kahle, A. Laidlaw, C. Diniz, M. M, Sanchez, and S. Y. Sung, “IBM Power Systems S814 and S824 Technical Overview and Introduction”, *IBM Redbook REDP-5097-00*, August 2014, http://www.redbooks.ibm.com/abstracts/redp5097.html
[8] SRC Computers Inc. Colorado Springs, CO, SRC MAP Architecture
[9] “DDR3 SDRAM High-Performance Controller MegaCore Functions” https://www.altera.com/products/intellectual-property/ip/memory-interfaces-and-controllers/m-alt-hpddr3.html
[10] “Persistent Memory Programming” http://pmem.io
[11] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, “Disaggregated Memory for Expansion and Sharing in Blade Servers”, in *Proceedings of the 36th annual International Symposium on Computer Architecture*, June 20 - 24, 2009
[12] B. Abali, R. J. Eickemeyer, H. Franke, C. S. Li, and M. A. Taubenblatt, “Disaggregated and optically interconnected memory: when will it be cost effective”, arXiv:1503.01416 [cs.DC]
[13] T. Hady, “Wicked Fast Storage and Beyond”, *7th Annual Non-Volatile Memories Workshop*, March 6-8, 2016
[14] T. V. Hulett , “All MRAM NVMe SSD - It is Fast!”, *Flash Memory Summit*, August 8-11, 2016
[15] “General Parallel File System”, https://www.ibm.com/support/knowledgecenter/en/SSFKCN/gpfs_welcome.html
[16] J. Axbeo, “Flexible I/O Tester”, https://github.com/axboe/fio
[17] H. Giefers, R. Polig, and C. Hagleitner, “Accelerating arithmetic kernels with coherent attached FPGA coprocessors”, in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE'15)*, Grenoble, FR, March 2015, Pages 1072-1077
[18] “Zynq: All Programmable SoC with Hardware and Software Programmability”, https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html