2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines

Microscope on Memory: MPSoC-enabled Computer Memory System Assessments

Abhishek Kumar Jain, Scott Lloyd, Maya Gokhale

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory, Livermore, CA 94550
{jain7, lloyd23, gokhale2}@llnl.gov

***Abstract*—Recent advances in new memory technologies and packaging options has focused attention on computer memory system design and evaluation. Examples include high bandwidth memories such as Hybrid Memory Cube and HBM, and 3D XPoint non-volatile memory. Emerging memories display a wide range of bandwidths, latencies, and capacities, making it challenging for the computer architect to navigate the design space of potential memory configurations, and for the application developer to assess performance implications of using such memories. In this work, we describe the Logic in Memory Emulator (LiME), a hardware/software tool specially designed for memory system evaluation and experiment. LiME uses the Xilinx Zynq UltraScale+ MPSoC on the ZCU102 board to capture any/all memory access, either from the Processing System (PS) or the Programmable Logic (PL). LiME employs novel loopback circuitry in conjunction with address map relocation to pass memory references from the PS into the PL side. The memory request is looped back into the PS DRAM memory controller and concurrently processed by LiME.**

We have demonstrated four high value use cases: full external memory access logging and replay, emulation of various memory system latencies by passing the memory request through delay registers before entering the PS memory subsystem, emulation of acceleration engines that can independently access memory, and performance comparison of two CPU architectures with respect to their memory behavior. In this paper we will describe this novel application of state-of-the-art MPSoC embodied by the LiME framework and highlight its uses.

I. Introduction

Computer memory systems, especially in the DRAM space, are approaching fundamental limits in which incremental improvements provide shrinking benefit for increasing effort. Spurred by latency, bandwidth, and capacity requirements of data centric applications, memory technologies fabricated from new materials, e.g. 3D XPoint (storage class memory), have recently emerged as potential main memory replacement or as additional slower tiers in the complex memory hierarchy. At the other end of the spectrum, 3D stacking has enabled the design of high bandwidth memory serving as scratchpad for highly concurrent application threads. An added benefit of new materials and packaging is the possibility of including logic and compute functions co-located with the memory. As memory system components are designed using such emerging devices, there is a need for system level exploration of the design space, with quantitative evaluation of the performance impact on applications using these memories, potentially incorporating near-memory accelerators.

In this work, we present a novel application using an MPSoC to accelerate design space exploration of these emerging memory systems. We leverage on-chip hard processors to run the main application, and route external memory accesses through the programmable logic to capture memory traces, provide programmable delay units to model latencies of a wide range of memories, and provide for inclusion of application specific logic for customized compute functions. The clock frequency of the emulated CPU can also be configured at run time. For an emulated 2.75 GHz CPU, our emulation framework runs at a 20× slowdown over real time while concurrently running acceleration logic and capturing every external memory access up to the trace memory size. Compared to the thousands of times slowdown of software simulation, our approach makes it feasible to quantitively assess application benchmark performance over a broad range of emerging or proposed memory systems.

Inclusion of hard IP modules, such as embedded ARM processors, cache hierarchy, memory controllers and AXI based system-interconnects now makes the modern MPSoC platform [1] an excellent choice for exploring architectural changes in memory hierarchy and composition. By using these hard IP blocks, we can avoid allocating FPGA resources to emulate the processor/cache/memory-controller components of the system, making the FPGA resources available for memory and acceleration logic emulation. The benefit of using MPSoC platforms for developing an emulator is reduced development effort and cost. Tight coupling of the FPGA fabric with hard processing system can further enable the design and analysis of novel architectures. One example is near memory computing using domain specialized heterogeneous accelerators.

Contributions of this work include, design, implementation, demonstration, and evaluation of an MPSoC-enabled infrastructure to

- enable replay and analysis of an application's memory behavior by capturing and logging external memory accesses to a separate off-chip memory device without affecting application execution
- emulate complex memory interactions in whole applications orders of magnitude faster than software simulation
- evaluate the use of emerging storage class memories
- emulate acceleration hardware co-located with the memory subsystem
- compare performance of a 32b/32B (processor/cache line) out of order processor with a 64b/64B in order processor with respect to memory

2576-2621/18/$31.00 ©2018 IEEE
DOI 10.1109/FCCM.2018.00035

- provide the Logic in Memory Emulator (LiME) hardware/software framework and working examples in the open source.

The ability to emulate memory hierarchy on a commercial MPSoC platform and have it execute software applications has significant ramifications for the computer architecture and reconfigurable computing community. With the LiME framework, architectural ideas can be prototyped in great detail yet with sufficient performance to support realistic evaluation on long running applications.

II. Related Work

Architecture simulators such as gem5 perform detailed simulation of processor and memory hierarchy and can emit many statistics and traces, including memory traces of arbitrary full system workloads [2], [3]. Due to the detailed software simulation of processor microarchitecture, these simulators have become extremely slow to the point where it is impractical to perform comprehensive evaluation of high level design alternatives. System-level architectural research, specially the exploration of near-memory acceleration and memory hierarchy, is hampered by the slow simulation speeds of software simulators and it is becoming increasingly difficult to evaluate the effect of architectural changes in the system components (memory hierarchy, processor/accelerator design etc.) for future heterogeneous computing systems. The work of [4] uses an MPSoC platform to evaluate a hardware memory trace collection device to augment a CPU.

FPGAs have been used historically for performing system simulations by emulating complex CPU, cache hierarchy and memory controllers [5], [6], [7], [8], [9]. Since the FPGAs were limited in capacity initially, multi-FPGA systems were used for the purpose of emulation. For decades, issues such as partitioning the design among multiple-FPGAs and circumventing limited I/O pins availability [10] limited FPGA-based CPU emulation primarily to CPU design companies [11]. Due to advances in process technologies, FPGA architectures, and logic density, FPGA developers can fit whole (and even multiple) CPUs into a single FPGA, reducing developer effort and time [12], [13]. For example, a Xilinx Virtex-4 device was used in [12] to emulate the Pentium microprocessor, apply architectural changes (increasing the size of L1 caches and branch target buffer) and evaluate their effects on the complete system. More recently, the RISC-V architecture is available on FPGA [14].

Although the growth in logic density of FPGA devices presented the capability to emulate complete processor systems, the approach of using FPGA logic blocks for emulating complex system components still requires tremendous resources. One such example is the emulation of Zynq SoC using Virtex-7 2000T device which requires more than half a million LUTs for emulating the Cortex-A9 cores, cache hierarchy and memory controller [15]. Additionally, prior approaches of emulation focus mainly on emulating the processor and cache hierarchy but not external memory with respect to latency and bandwidth. In contrast, our approach uses the native hard IP cores/cache hierarchy and focuses on external memory. High speed emulation of external memory system is necessary for assessing the potential benefit of emerging memory architectures/configurations on realistic application access patterns. The emerging opportunity for near memory acceleration units must also be carefully evaluated in realistic application scenarios.

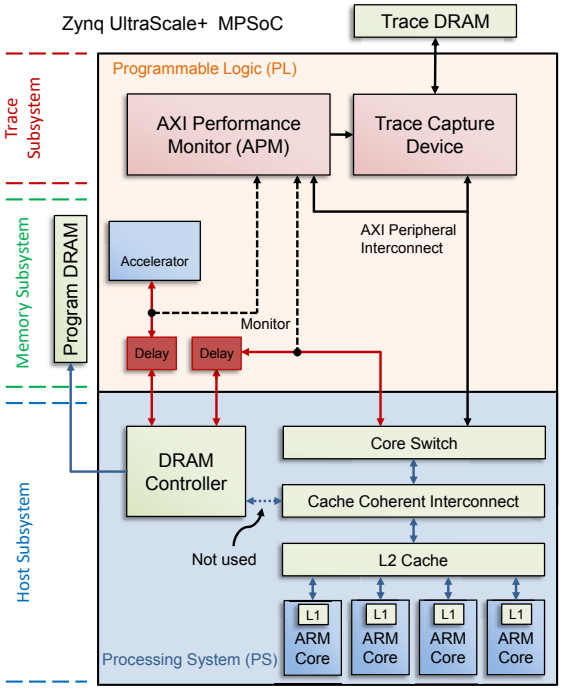


Fig. 1: Zynq UltraScale+ with emulation framework.

III. Method

LiME addresses the need for fast memory system evaluation by leveraging the Xilinx Zynq UltraScale+ MPSoC device on a ZCU102 development board.[1] As shown in Figure 1, the Zynq architecture consists of a programmable logic (PL) region where custom hardware can be implemented and a fixed logic processing system (PS) region. On the UltraScale+, the PS contains a quad-core 64-bit ARM Cortex-A53 processor with advanced NEON unit (double-precision SIMD support). The ARM CPU has a two level cache hierarchy with 64B cache lines. Apart from the MPSoC device, the ZCU102 board contains many peripherals. For memory system evaluation, we focus on the two on-board DDR4 memories. There is a 4 GB 64-bit DDR4 (SODIMM) connected to the PS via hard memory controller and used as the primary system DRAM. A second 512 MB 16-bit DDR4 connects to PL. We use this second DDR4 to log memory request traces, interfacing to it with a Memory Interface Generator.

Ordinarily, memory requests from the ARM cores go directly to the DRAM controller on the PS and are not visible to the PL. A key aspect of the LiME design is to direct all memory references that are not satisfied in cache to the PL. To accomplish this, we relocate a program's data addresses from the default zero base addressing to a high address base at 0x4_0000_0000 which maps to an AXI port from the PS to the

[1]The entire hardware/software code for a ZC706 version of LiME is available at https://bitbucket.org/PerMA/emulator_st/.

PL. In bare metal mode, this is done by locating a program's heap and stack above 0x4_0000_0000 through configuration in the linker load script. For Linux, the method is a bit more complicated, as the Linux image, device tree, and ramdisk all must be relocated [16]. In this paper, we present results using bare metal benchmarks. With all memory accesses from the PS going into the PL, and memory accesses initiated by accelerator modules already being in the PL, it is possible to capture and filter every memory access. Memory transactions are logged in the PL-attached circular memory trace buffer, giving complete visibility into a window of memory behavior of the system. The details of trace capture are given in Section III-A.

Whether or not a memory request is logged to the 512 MB DDR4, it must be fulfilled by doing a read or write to the PS 4 GB DDR4 module. We have developed a novel loopback mechanism to forward the memory request to the PS memory controller after passing through the PL. The details of the loopback mechanism are described in Section III-B.

In the loopback path, additional LiME components provide control over the latency of a memory operation, which enables us to use the single DDR4 memory to emulate multiple different memory types. Our emulation benchmarks use two memories concurrently, a low latency, low capacity scratchpad, and a higher latency, higher capacity main memory. Each memory is associated with a delay unit that has individual control over the read and write delay. This makes it possible to emulate asymmetrical read and write latencies characteristic of many byte addressable "storage class" memories. There are separate groups of delay units for the ARM in the PS and for the accelerator in the PL. The delay amounts are related to CPU clock speed, the speed of the loopback circuitry in the PL, bandwidth of the AXI datapaths, and the latency within the PS memory controller subsystem. The clocks and datapaths associated with all of these components must be carefully configured to meet the desired emulation goals (see Section III-C).

A. Trace capture

When a software program runs on an ARM core and trace is enabled, memory transactions on the loopback path are captured by the AXI Performance Monitor (APM). Trace events created by the APM include a time stamp and transaction details such as the size and the AXI ID of the requester. We modified the APM IP to additionally capture memory addresses. A stream of trace events from the APM are forwarded to the Trace Capture Device which writes the events to a separate trace memory (512 MB 16-bit DDR4). The AXI Memory Interface Generator (MIG) is used to interface the Trace Capture Device with the PL attached DDR4 memory providing a peak bandwidth of 4.8 GB/s. This bandwidth is sufficient to capture the activity of several cores given that, under emulation, a single core running a data-intensive application generates trace events at a rate of 241 MB/s. In case emulation is not needed, memory traces can be captured for memory access pattern analysis when running the application at the highest ARM clock frequency.[2]

[2]We have not tested this mode with multiple cores, and our experiments use emulation so that we can capture timing-accurate memory traces representative of a server class 2.75 GHz core.

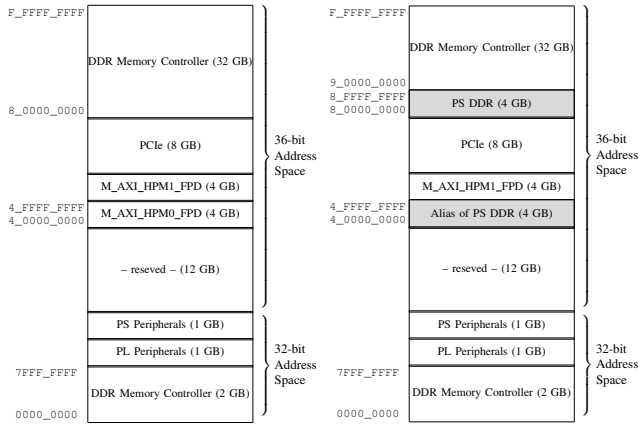


Fig. 2: (a) Original address space and (b) Modified address space of Zynq UltraScale+.

B. Loopback

A loopback path refers to an AXI connection between a PS master and PS slave through the programmable logic, so that hardware IP blocks built using FPGA resources can monitor or manipulate AXI transaction address and/or data in the programmable logic before the data reaches its intended target in the PS. As the application executes on the ARM CPU, the loopback circuit captures memory requests as they appear on the AXI M_AXI_HPM0 interface and routes them through the PL to the AXI S_AXI_HP0 interface. M_AXI_HPM0 is an AXI master port for accessing AXI slaves in the PL. S_AXI_HP0 is an AXI slave port that is accessed by AXI masters in the PL. The loopback logic uses a combination of Xilinx IP blocks and in-house hardware IP blocks to translate the destination address from an address in the PL to an address in the PS memory.

Figure 2 shows the address map of the platform. The default DDR controller configuration splits the 4 GB of PS memory into two 2 GB address ranges, as shown in Figure 2 (a). The range from 0000_0000 to 7FFF_FFFF access the first 2 GB, and the range 0x8_0000_0000 to 0x8_7FFF_FFFF addresses the upper 2 GB. Under this default setting, the processor can send memory requestes in either of these two address ranges to the external memory entirely through the PS.

Under loopback, we modify the fixed logic memory controller settings to recognize addresses starting at 0x8_0000_0000 to refer to the entire 4GB range as shown in Figure 2 (b). This eliminates the default split range of two separate 2 GB regions. Then using the loader script (in bare metal mode) the program stack and heap are assigned starting at 0x4_0000_0000, with the result that all data accesses go to the PL on the AXI M_AXI_HPM0 interface. The details of address translation in the loopback path are shown in Figure 3. The top AXI Shim remaps the lower 1M address range (R1) from 0x04_000X_XXXX to 0x08_000X_XXXX, thus getting it into the 4 GB region. The second AXI Shim remaps the remaining address range (R2) from 0x04_XXXX_XXXX to 0x18_XXXX_XXXX. The purpose of this relocation is to separate addresses into two memory ranges that could have

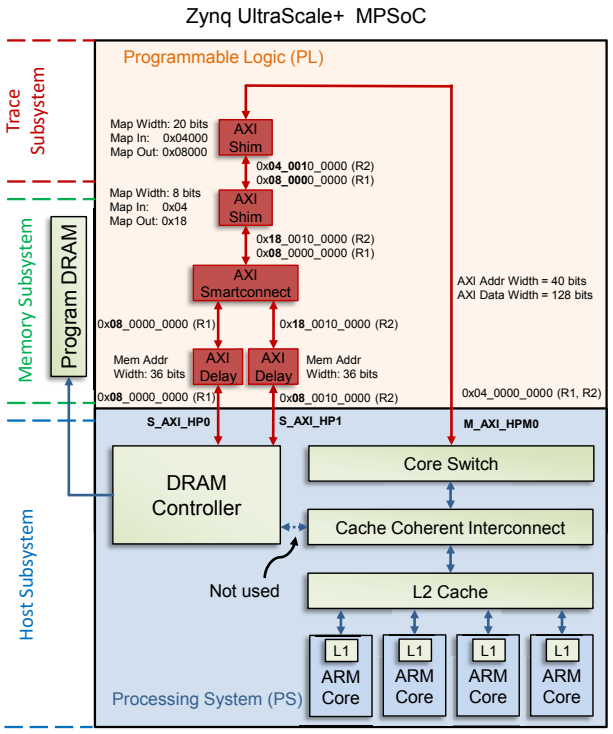


Fig. 3: Loopback circuit for emulating two memories, one with address range R1 (1 MB) and the other with address range R2 (4 GB minus 1 MB).

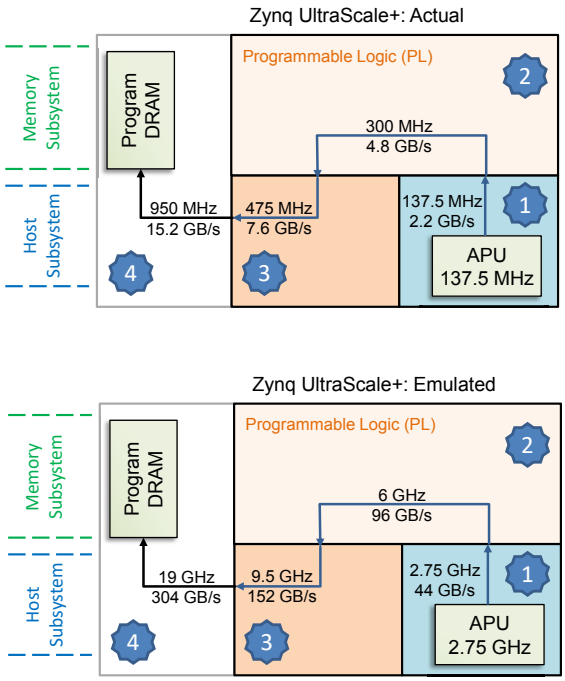


Fig. 4: Different clock domains in the loopback path for Zynq UltraScale+.

different latencies that are routed separately to eventually re-enter the PS at ports S_AXI_HP0 and S_AXI_HP1 into the DRAM controller.

C. Time accurate emulation

For timing accurate emulation, the frequencies of multiple clocks across the PS and PL must be coordinated. In our experiments, we emulate a 2.75 GHz CPU and, optionally, a 1.25 GHz accelerator in the PL that exchanges data with the CPU through an SRAM buffer. The ARM's maximum clock frequency is 1.2 GHz, and our PL logic runs at 300 MHz, indicating that at minimum, the ARM must be slowed down. However, there is an additional limiting factor of maximum AXI bus bandwidth between PS and PL imposed by the loopback path. As shown in Figure 4 upper figure, routing memory traffic through loopback imposes a bandwidth limit of 4.8 GB/s, which is 63% of the direct path from CPU to memory. To accommodate these constraints as well as maintain compatibility with a prior ZC706 LiME implementation, we have chosen a scaling factor of 20 in our emulation experiments. For these studies, the CPU runs at 137.5 MHz to give an emulated CPU frequency of 2.75 GHz, and the accelerator runs at 62.5 MHz. To maintain full memory bandwidth that the platform can sustain, the DRAM clocks run at maximum rate. All memory requests go through the delay units to emulate scaled memory latencies.

The delay unit is an in-house hardware IP block which is run-time programmable in a range of 0–174 us in 0.16 ns increments. It has one AXI-lite interface for loading the appropriate values for particular read/write latencies. It also has one AXI slave interface to receive the memory-traffic and one AXI master interface to send the traffic out.

In the loopback path, a combination of AXI delay, shim and smartconnect blocks provide control over the latency of a memory operation, which enables us to use the single DDR4 memory to emulate multiple memory types. Our benchmarks use two emulated memories concurrently: a low latency, low capacity scratchpad, and a higher latency, higher capacity main memory. Figure 3 shows the memory requests passing through AXI shim blocks, AXI smartconnect, AXI delay blocks and finally looping back to PS, through the DRAM controller and out to the DDR4 memory.

All clocks use a set of PLLs that generate high-frequency clocks. A PLL is selected based on the use case and system-level performance requirement. The programmable dividers determine the final clock-out frequency. There are four main clock regions in the loopback path within the system as follows:

1) Processor region: It consists of system interconnect and Application Processing Unit (APU) block (ARM cores and cache hierarchy). The clock frequency of the APU and the system interconnect can be controlled individually. The APU PLL (APLL) is used to generate the clock for the APU. By setting the 6 bit divider value within ACPU_CTRL register, the APU can be

configured to run at the desired frequency. We set the divider value accordingly to generate a clock of 137.5 MHz for the APU. The width of the datapath within the processor region is 128 bits (16 Bytes) and hence the bandwidth is limited to 2.2 GB/s (137.5 MHz $\times$ 16 Bytes). Since the scaling factor is 20$\times$, the clock frequency and the bandwidth limit under emulation is 2.75 GHz and 44 GB/s, respectively. This region determines the CPU bandwidth limit since the downstream regions have higher bandwidth.

2) Loopback region: It gets clocked using PL clocks and the PL clocks are generated using I/O PLL (IOPLL). The clock frequency of the loopback region is 300 MHz and the width of the datapath is 128 bits (16 Bytes). Hence the bandwidth of the loopback region is limited to 4.8 GB/s.
3) DDR controller region: DDR PLL (DPLL) is used to generate the DDR controller reference clock which clocks the interface side of the DDR subsystem. DDR controller reference clock is 475 MHz and the width of the datapath is 128 bits (16 Bytes). Hence the bandwidth of this region is limited to 7.6 GB/s.
4) DDR region: It consists of the external memory itself which runs at 950 MHz. Since it is a 64-bit DDR4 SODIMM interface, the bandwidth is limited to 15.2 GB/s.

An application program controls the clocks and delay amounts by writing to memory mapped control registers using macros and utility functions provided by the LiME framework. For example, a utility function takes the desired memory latency in nanoseconds, converts it to a delay value, and writes that value into the specified delay register. Typically the program performs initialization and finalization at the default ARM clock frequency with no delays to the memory subsystem. Emulation and trace capture are enabled within Regions of Interest (ROI) such as main work loops.

IV. Resource requirements of the emulation system

LiME has four main parts in the programmable logic, the *CPU-memory path* containing two AXI shims, one AXI smartconnect and two AXI delay IP blocks; the *Accelerator-memory path* containing two AXI shims, one AXI interconnect and two AXI delay IP blocks; the *Trace logging circuitry* containing AXI performance monitor, Trace capture device, Memory Interface generator (MIG); and the *Accelerator* containing a Microblaze soft processor (MCU), Load Store Unit (LSU) and AXI stream switch IP blocks.

The Zynq UltraScale+ fabric contains 274K LUTs, 548K FFs, 900 BRAMs and 2520 DSP Blocks. Our emulation system consumes 40912 LUTs, 40698 FFs, 75 BRAMs and 10 DSP Blocks. LiME uses only 15% of the device resources, leaving 85% of the resources for other purposes.

Figure 5 shows the breakdown of resources required for each part. Trace circuitry takes a major portion (more than 40%) of the CLB resources while also consuming 52 BRAMs and 3 DSP blocks. The accelerator requires around 15% of the CLB resources while also consuming 23 BRAMs and 7 DSP blocks. The remainder of the CLB resources (around 45%) are used for the CPU-memory and Accelerator-memory datapaths.

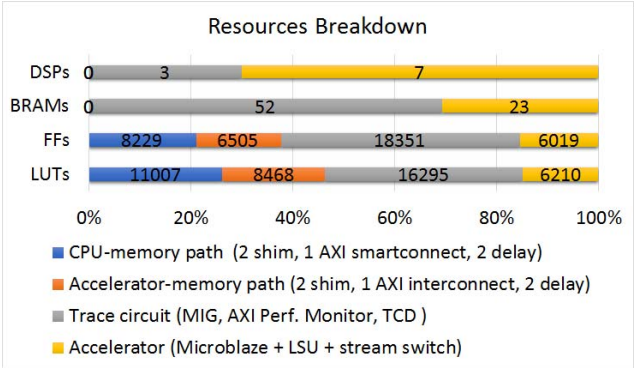


Fig. 5: % Resource breakdown of each part of the system.

V. Use Cases

The LiME framework can be used for many kinds of memory-related studies. In this section, we highlight four use cases: trace capture and logging, evaluating emerging memory technologies, memory interactions and performance assessment when using an accelerator, and comparing performance differences of two CPUs using similarly configured memory systems.

A. Memory trace capture and logging

Memory traces are useful to study application memory access patterns. Without separate program and trace memories, capturing timestamped traces from running programs on real hardware introduces inaccuracies. First, the memory system is perturbed by injecting the logging traffic in addition to the application accesses. To mitigate this issue, and additionally mitigate overhead, a sampling approach is typically used. Thus collecting a full trace for a meaningful period is rarely possible. If a software simulator is used, slowdown can be many orders of magnitude over real hardware. A full FPGA emulator is very labor intensive to create, and would still need a separate path to memory to store the traces.

The LiME trace capture mechanism is controlled in software through setting memory mapped control registers. The C support library provides simple macros to turn a memory trace on and off, so that traces can be collected as desired, typically within Regions of Interest such as an inner loop. For the accelerators we've built, all accelerator memory requests are logged when tracing is turned on. Our experiments to date collect a trace while a single core is active in bare metal mode. This provides an application-focused data set for analysis.

Table I shows an example with read and write requests from both ARM core and accelerator. Note the fixed 64-byte memory requests from the ARM and the 8/16-byte requests from the accelerator. The LiME framework provides for multiple memory request sizes, enabling performance studies on the impact of narrow and wide memory access on applications.

Memory events can be studied for many purposes. Memory system architects can extract the read/write mix of an application or workload to optimize for specific use cases such as read-heavy data analytics or write-heavy HPC checkpoints. Memory traces that include addresses enable the study of spatial and temporal locality.

TABLE I: Memory trace, showing both ARM (Source 0) and accelerator (Source 1) memory accesses.

Source	Type	Address	Length	AXI ID	Time
0	W	0x400082F80	64	525	481545
0	W	0x400082FC0	64	525	482101
0	R	0x4002011C0	64	1165	482432
0	R	0x400201200	64	1293	482441
0	R	0x400201240	64	1037	487379
0	R	0x400201280	64	1037	492539
1	W	0x400080000	8	3	498523
1	R	0x400082000	16	0	493495
1	W	0x400080008	8	3	498557
1	W	0x400080010	8	3	503270
1	W	0x400080018	8	3	503304
1	W	0x400080020	8	3	503400

Figure 6 shows a timeline of the number of bytes read and written per second as the random access benchmark (see Section V-C) executes. The plot distinguishes between reads and writes, between CPU and accelerator accesses, and between accesses from the "LSU" and "MCU" components of the accelerator. In addition, when combined with an energy model, the trace can show power use over time as the application runs. The trace can also be replayed on a real memory system to study bank conflict, strided access patterns, and dependency chains such as pointer chasing. [17] combines memory traces from multiple executions to generate a synthetic concurrent workload, and replays the composite trace on a new memory device to understand bandwidth and latency profiles as load is increased.

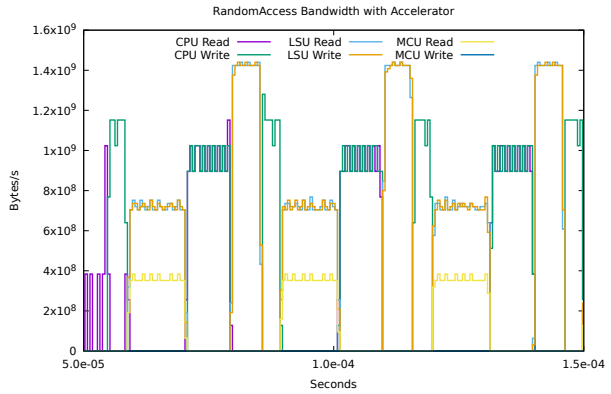


Fig. 6: Timeline of bytes read and written per second by CPU and accelerator components.

B. Evaluating future storage class memories

After many years of R&D, innovations in memory technology are appearing in commercial products. At one end of the capacity spectrum, stacked memory such as High Bandwidth Memory and MCDRAM offer a jump in bandwidth with small capacity and slightly longer latencies. At the other end, high capacity storage class memories (SCM) like 3D XPoint are emerging with unique characteristics such as asymmetric read/write latency, wear from writes, and small granularity read/write payloads. Combinations of these memories are being proposed for future computing systems as an alternative to high performance DDR.

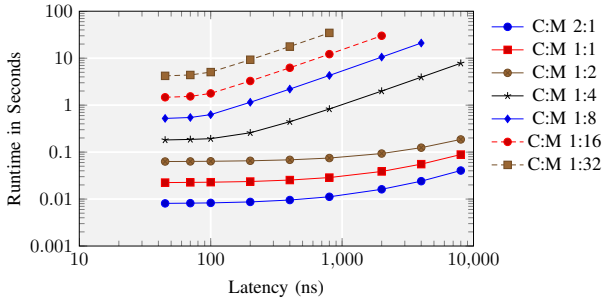


Fig. 7: DGEMM execution time on 64-bit processor at varying latencies and varying cache-to-memory ratios.

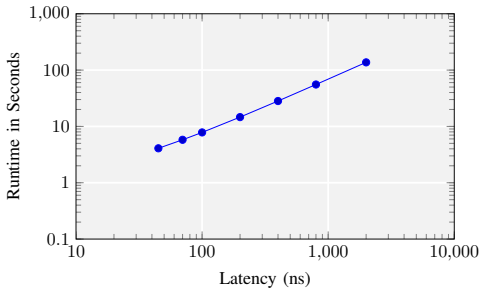


Fig. 8: SpMV execution time on 64-bit processor at varying latencies with a cache-to-memory ratio of 1:112.

Using LiME, we have conducted parameter sweeps over memory systems in which the main memory latencies model the entire range from fast DRAM or STT-MRAM to latencies of projected storage class memories, e.g. 3D XPoint. Using the delay units we model latencies from 45 ns to 800+ ns. Since the delay units provide separate read and write delay amounts, we can model longer write latency for SCM. By varying problem size, we can evaluate performance at different ratios of cache to working set size in main memory. Results of this study for two of the benchmarks evaluated are shown in Figures 7 and 8. The benchmarks represent dense and sparse matrix operations respectively. DGEMM is double precision dense matrix multiply, and SpMV is sparse matrix vector multiply. The set of curves in the DGEMM plot show an inflection point between a cache-to-memory ratio of 1:2 and 1:4, indicating that for small cache to memory ratios, the DGEMM kernel tolerates considerable main memory latency. This suggests that block methods typically used in matrix libraries that keep the working set in cache can tolerate the longer latencies of SCM. In contrast, sparse matvec shows consistent and steady increase in execution time as latency increases. For sparse operations that depend on SCM, applications will need a high level of concurrency and greater throughput to offset the loss in performance at longer latencies. This strategy is used by GPUs to schedule a large number of concurrent threads.

Note that the run time for sparse matvec goes up to 100 seconds of emulated time. Hardware acceleration through the FPGA greatly speeds up experiments that sweep a large parameter space, and makes it tractable to investigate more parameter combinations. While a single curve is shown in the sparse matvec plot for a matrix of size 2^{18}, we have run all the benchmarks over many application-specific problem sizes to determine trends. For SpMV, the plots for other matrix sizes were similar to the one displayed, but this was not true across all the benchmarks tested.

C. Evaluating a gather/scatter acceleration engine

Prior work by [18] and [19] present programmable gather/scatter engines colocated with a memory subsystem. On an FPGA, a "data rearrangement engine" (DRE) can be constructed using AXI data mover IP blocks (Load/Store Unit or LSU), a Microblaze soft processor (Microblaze control unit or MCU) with associated block ram and BRAM interfaces, AXI-Stream FIFO and AXI Interconnect along with custom logic. In our evaluation, the DRE supports strided DMA, and full indirect addressing of the form A[B[i]]. We've adapted several benchmarks to use a DRE to transform random access of cache unfriendly data structures to streaming sequential access to gathered/scattered data. In this method, the CPU program, at run time, initializes the DRE with base addresses and strides (if applicable). Within a loop, the CPU program issues commands to the DRE through memory mapped registers to gather or scatter a block of data. The data block is staged in an SRAM buffer. The DRE performs the requested operation and then sets a completion bit in a memory mapped register. Our benchmarks include cache coherence operations to maintain a consistent copy of the data structures in memory since both the ARM and Microblaze have data caches. In this study, the CPU is configured to be a 2.75 GHz single core processor, and the DRE runs at 1.25 GHz. Figures 9 and 10 compare performance of CPU-only with CPU+DRE. Image difference is an image decimation and differencing benchmark. It computes the pixel-wise difference of two reduced resolution images of size 16000 x 8000 elements with 32-bit pixels and a decimation factor of 16. The RandomAccess benchmark comes from the HPC Challenge benchmark suite. Performance is shown at an array size of 0.5 GB and 4M updates. These results demonstrate that substantial speedup can be gained with a DRE due to the higher number of in-flight requests issued by the near-memory DRE.

D. Comparing performance across CPUs

The LiME framework focuses on memory system experiments, using the MPSoc hard logic CPU to run application code. This approach eliminates the need to simulate or emulate the CPU and cache hierarchy. However, the specific CPU (in order vs. out of order, word size) and cache hierarchy determine which application loads/stores result in external memory transactions. To evaluate the effects of CPU and cache on application performance, both in run time and in trend lines with different memory subsystems, we compare LiME emulation results on the 64b Zynq UltraScale+ A53 core with those on a 32b Zynq-7000 A9. Both ZC102 and ZC706 development boards have separate memories, enabling memory trace capture and logging under emulation.

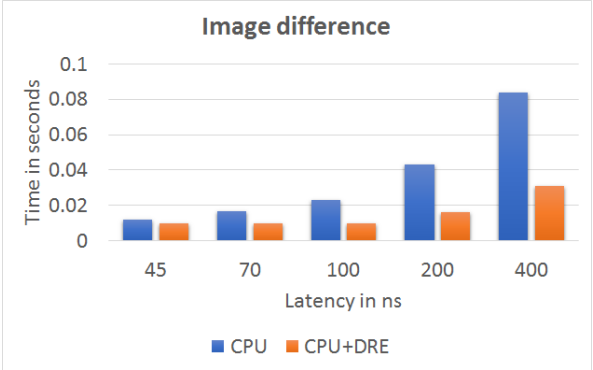


Fig. 9: Comparison of execution time between CPU and CPU+DRE at varying latencies for Image difference.

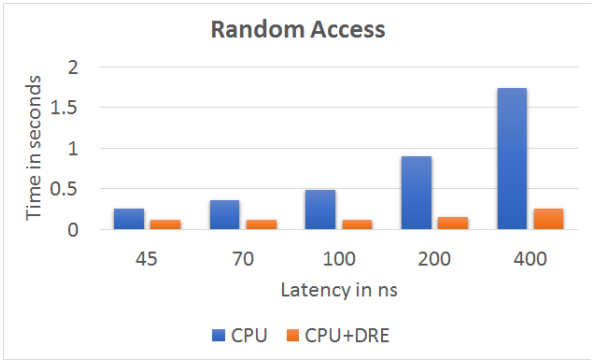
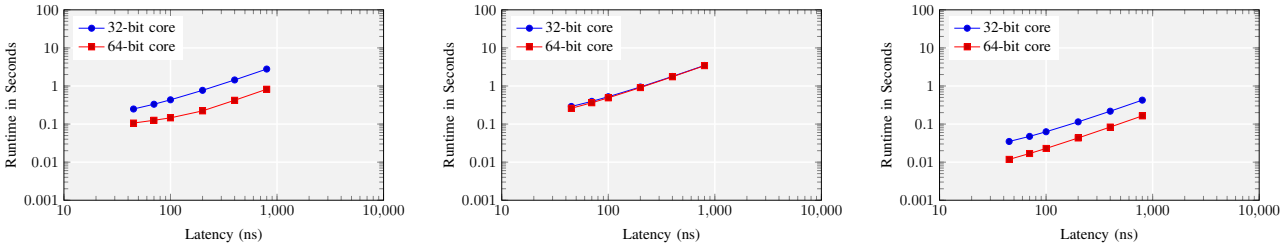


Fig. 10: Comparison of execution time between CPU and CPU+DRE at varying latencies for Random Access.

Figures 11(a), 11(b), and 11(c) compare performance of the STREAM-triad, random access, and image difference benchmarks on the two platforms over a range of memory latencies. For these tests, the gcc compiler version 6.2.1 from the Vivado 2017.1 release was used for both architectures with -O2 optimization.

As expected, the bandwidth-dominated STREAM-triad benchmark runs significantly faster on the 64-bit processor over the entire range of latencies. Up to 100 ns, run time is slightly flatter than for the 32-bit processor, but that effect disappears at high latencies. Image difference requires some computation, giving the 64-bit core an advantage. However, random access is mostly dependent on memory latency and Figure 11(b) shows that performance is nearly the same for both architectures between 45 and 800 ns.

From the perspective of memory focused emulation, we note that although the projected run times are different on the two processors, the trend lines are very similar. Thus for the purpose of studying application behavior as memory latency varies, results from both CPUs point to the same conclusions. The new insight from this study is that when access patterns are extremely random with little computation or reuse, the CPU capability is largely irrelevant.



(a) STREAM-triad (b) Random Access (c) Image Difference

Fig. 11: Execution time for application benchmarks at varying latencies.

VI. Summary, Conclusions, and Future Work

The Logic in Memory Emulator takes a novel approach to evaluating memory systems from the perspective of application performance. By employing a state of the art MPSoC, a platform with two separate memory subsystems, and a unique loopback datapath, LiME emulates application execution with current and future memory systems at 1/20th real time. Our method makes it possible to search a larger design and parameter space than would be possible with software simulation without requiring the extreme design effort of designing an entire CPU on FPGA, or the cost in logic and build time of existing soft processors. Using LiME we've conducted experiments to capture and analyze memory traces, evaluate the use of storage class memories as main memory for future computer systems, and evaluate a near memory gather/scatter accelerator. To study the effects of different CPUs and cache hierarchies, we've run LiME on two variations of ARM CPUs and gotten insight into performance trends for different types of access patterns. The LiME hardware/software infrastructure is available as open source. For future work, we plan to expand our studies to full workloads by performing performance evaluations of application suites under Linux. These studies will give insight into the mix of OS and application memory accesses. We also plan to map additional accelerators in soft logic to further study potential performance benefits as well as the issues of synchronization and communication between CPU and accelerators.

VII. Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. DE-AC52-07NA27344. This work was supported by Lawrence Livermore National Laboratory LDRD project 16-ERD-005. LLNL-CONF-748088.

References

[1] S. Ahmad, V. Boppana, I. Ganusov, V. Kathail, V. Rajagopalan, and R. Wittig, "A 16-nm multiprocessing system-on-chip field-programmable gate array platform," *IEEE Micro*, vol. 36, no. 2, pp. 48–62, 2016.

[2] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 92–99, 2005.

[3] T. Austin, E. Larson, and D. Ernst, "Simplescalar: An infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, 2002.

[4] N. C. Doyle, E. Matthews, G. Holland, A. Fedorova, and L. Shannon, "Performance impacts and limitations of hardware memory access trace collection," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 506–511.

[5] H. Angepat, D. Chiou, E. S. Chung, and J. C. Hoe, "Fpga-accelerated simulation of computer systems," *Synthesis Lectures on Computer Architecture*, vol. 9, no. 2, pp. 1–80, 2014.

[6] M. Dahl, J. Babb, R. Tessier, S. Hanono, D. Hoki, and A. Agarwal, "Emulation of the sparcle microprocessor with the mit virtual wires emulation system," in *IEEE Workshop on FPGAs for Custom Computing Machines (FCCM)*. IEEE, 1994, pp. 14–22.

[7] K. Oner, L. A. Barroso, S. Iman, J. Jeong, K. Ramamurthy, and M. Dubois, "The design of rpm: an fpga-based multiprocessor emulator," in *Proceedings of the Third International ACM Symposium on Field-Programmable Gate Arrays (FPGA)*. IEEE, 1995, pp. 60–66.

[8] M. Dubois, J. Jeong, Y. H. Song, and A. Moga, "Rapid hardware prototyping on rpm-2," *IEEE Design & Test of Computers*, vol. 15, no. 3, pp. 112–118, 1998.

[9] G. Schelle, J. Collins, E. Schuchman, P. Wang, X. Zou, G. Chinya, R. Plate, T. Mattner, F. Olbrich, P. Hammarlund *et al.*, "Intel nehalem processor core made fpga synthesizable," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM, 2010, pp. 3–12.

[10] J. Babb, R. Tessier, and A. Agarwal, "Virtual wires: Overcoming pin limitations in fpga-based logic emulators," in *IEEE Workshop on FPGAs for Custom Computing Machines (FCCM)*. IEEE, 1993, pp. 142–151.

[11] P. K. Joshi, "Emulation using fpgas," in *Designing with Xilinx® FPGAs*. Springer, 2017, pp. 219–235.

[12] S.-L. L. Lu, P. Yiannacouras, R. Kassa, M. Konow, and T. Suh, "An fpga-based pentium® in a complete desktop system," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM, 2007, pp. 53–59.

[13] P. H. Wang, J. D. Collins, C. T. Weaver, B. Kuttanna, S. Salamian, G. N. Chinya, E. Schuchman, O. Schilling, T. Doil, S. Steibl *et al.*, "Intel® atom processor core made fpga-synthesizable," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM, 2009, pp. 209–218.

[14] "Support for rocket chip on zynq fpgas," github.com/ucb-bar/fpga-zynq, 2018.

[15] "Virtex-7 2000t fpga for asic prototyping & emulation," www.xilinx.com/video/fpga/virtex-7-2000t-asic-prototyping-emulation.html, 2018.

[16] L. W. Li, G. Duc, and R. Pacalet, "Hardware-assisted memory tracing on new socs embedding fpga fabrics," in *Proceedings of the Usenix Annual Computer Security Applications Conference*. Usenix, Dec. 2015.

[17] S. Lloyd and M. Gokhale, "Evaluating the feasibility of storage class memory as main memory," in *Proceedings of the Second International Symposium on Memory Systems*. ACM, 2016, pp. 437–441.

[18] J. C. Beard, "The sparse data reduction engine: chopping sparse data one byte at a time," in *Proceedings of the International Symposium on Memory Systems, MEMSYS 2017, Alexandria, VA, USA, October 02 - 05, 2017*, 2017, pp. 34–48. [Online]. Available: http://doi.acm.org/10.1145/3132402.3132431

[19] S. Lloyd and M. Gokhale, "In-memory data rearrangement for irregular, data-intensive computing," *Computer*, vol. 48, no. 8, pp. 18–25, 2015.