Pipelining an Open Source Last-Level Cache

Kevin Yunchuan Jiang Joseph Zuckerman Luca P. Carloni
Department of Computer Science, Columbia University
kyj2112@columbia.edu, {jzuck, luca}@cs.columbia.edu

1. INTRODUCTION

Systems-on-chip (SoCs) that integrate a heterogeneous mix of complex IP blocks can utilize on-chip shared memory to improve performance and simplify programming [1, 2, 3, 4]. Like that of traditional homogeneous architectures, the memory hierarchy of heterogeneous SoCs may feature multiple levels of caches, orchestrated by a cache coherence protocol. For example, Giri et al. propose a cache hierarchy for heterogeneous SoCs built on top of a network-on-chip (NoC) [5]. Their cache hierarchy supports multicore execution of processor cores and enables multiple different coherence modes for hardware accelerators. Utilization of these modes can improve execution time and reduce off-chip memory accesses for a variety of different workloads [6, 7].

ESP is an open-source platform for heterogeneous SoC design that simplifies the integration of new IP blocks within a scalable SoC architecture and allows for rapid prototyping of full systems [8]. ESP's cache hierarchy implements the protocol proposed by Giri et al – an extended MESI directory-based protocol. The L1 of open-source processors (that potentially implement different ISAs) interfaces to the ESP L2. The last-level cache (LLC) and directory controller not only orchestrate coherence across these L2 caches but also enable accelerators without a private cache to leverage the cache hierarchy to access on-chip shared data.

In this work, we present an improved version of the ESP LLC. While the original version is implemented with a multi-cycle datapath, the improved version features a pipelined datapath, allowing for significantly greater throughput. Since the last-level cache is the main synchronization point for the various IPs throughout an SoC, this can give significant performance improvements when there is a high density of requests at the LLC. In particular, we show how the pipelined datapath can improve the performance of accelerators that access their data directly from the LLC.

2. THE ESP CACHE HIERARCHY

Figure 1 shows an example of a 4x4 tile ESP system with elements of the memory hierarchy highlighted; the heterogeneous mix of tiles are connected by a multi-plane network-on-chip (NoC).

ESP's coherence protocol is implemented by the L2 and last-level caches. Each *Processor Tile* contains an open-source core, instantiated off-the-shelf with its own L1 cache. The ESP L2 allows different cores to seamlessly participate in ESP's coherence protocol, despite having potentially different ISAs, endianness, and interfaces. *Accelerator Tiles*, which instantiate a *loosely-coupled accelerator*, can also optionally be equipped with their own private L2 cache, which allows them to also participate in the coherence protocol, just like processor cores.

The *Memory Tile* contains the LLC and directory controller, which manage coherence among all L2 caches in the system, and a channel to external DRAM. The LLC can be partitioned across multiple memory tiles in the system with each partition serving a discrete partition of the global address space and providing its own channel to DRAM. The LLC can also handle DMA requests directly from accelerators, saving off-chip memory accesses in many circumstances.

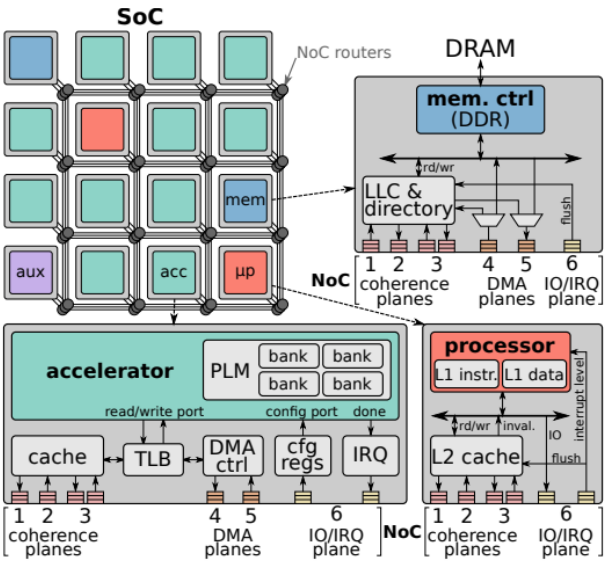

Figure 1: Detailed ESP Tile Grid [5]

Compared to the L1 and L2 caches, the LLC contains additional metadata for all cache lines active in the system at any point, such as which cores own or share the cache line. These metadata are necessary for the execution of the extended MESI directory-based cache coherence protocol. In addition to the standard MESI states (Modified, Exclusive, Shared, Invalid), the ESP coherence protocol introduces a Valid state, which corresponds to lines that are not owned or shared by any private cache. Maintaining a Valid state saves an off-chip memory access to subsequent requests for this line; it is also used to represent lines that have been accessed by an accelerator through DMA to the LLC. Valid lines are also the first choices for evictions.

The various caches and processing elements in ESP communicate through the NoC. Three NoC planes are used for coherence requests, responses, and forwards, while two planes are used for DMA requests and responses from/to accelerators.

3. PIPELINING THE ESP LLC

3.1 Original LLC

The original LLC is implemented as a multi-cycle datapath for handling requests according to the extended MESI protocol. The multi-cycle datapath consists of six stages, and an FSM manages the execution of a single request across these six stages. In any given cycle, only one stage of the datapath is active. For example, the first stage, which prioritizes and accepts the incoming requests, is idle while later stages of the datapath are operating. This means that only one request is processed by the LLC at a time, which can create backpressure on the NoC if many requests arrive in a short time. The microarchitecture for this implementation is shown in Figure 2.

3.2 Pipelined LLC

In this work, we took the existing six-stage datapath and enabled the pipelined execution of requests. We use the same six stages from the multicycle datapath and add pipeline registers in between each stage and logic for exerting backpressure between individual stages. The FSM that governs the execution of an individual request is removed, and all stages can be active in any given cycle. This implementation allows for the simultaneous processing of multiple requests. The microarchitecture for this implementation is shown in Figure 3.

Pipelining a multi-cycle datapath presents numerous challenges. Signals from previous stages that would previously stay constant throughout the lifetime of a request now can be overwritten by new requests. This required significant refactoring of the RTL to pass these signals with each request through the pipeline.

The cache itself is a multi-bank, multi-port memory that allows for an entire set to be read in a single clock cycle. This applies to the cache lines themselves and also their associated metadata. Another key challenge is avoiding read-after-write (RAW) hazards and read/write collisions within a particular set. RAW hazards can occur because the memory is not updated until the final pipeline stage, while it is read in the second stage. Without careful consideration, it would be possible for a new request to read information about a set before an old request updates the data. It is also necessary to avoid reading and writing the same location in this memory during the same clock cycle to avoid non-determinism in accesses. To solve these two hazards, we introduce a *set table*, which keeps track of the sets of all requests currently active in the pipeline. The second stage of the pipeline checks the set table before accepting new requests in order to avoid multiple requests to the same set in flight at once. The final pipeline stage removes an entry from the set table when it is retired. Requests in different sets do not face these hazards, but they pose a different problem: the memory architecture of the original LLC was not designed for simultaneous reading and writing; we expanded the number of ports in the memory to sustain higher throughput.

Pipelining the multi-cycle datapath also introduced potential for undesired out-of-order completion of requests. There are two problematic cases: when the LLC wants to recall an Exclusive or Modified cache line from an L2 for eviction and when there is a cache line in a transient state waiting for a data response from an L2. In both cases, the LLC should not service any newer requests until the data response from the L2 has been received and completely serviced. However, these behaviors are only determined at the fifth stage of the pipeline, and so new requests can enter the pipeline before the response is received.

Our solution was to add a feature to the fifth stage that stalls the first stage and flushes all the newer requests in the pipeline into a FIFO. After the data response has been received and serviced, the LLC will process the requests in the FIFO before accepting any new requests from externally. LLC will then prioritize feeding requests from the FIFO back into the pipeline once the data response has been received and serviced.

While the original LLC significantly improved DMA latency for accelerators by potentially eliminating off-chip memory accesses, the pipelined LLC further improves DMA performance. Large DMA requests span multiple cache lines, and the LLC handles each cache line as a separate "request" for a different address. The pipelined LLC is designed to generate a new "request" to insert into the second pipeline stage every cycle, allowing the DMA request to fully utilize all stages of the pipeline. When the accelerator data resides in the LLC, this allows a new cache line to be sent or written every cycle, as opposed to every six cycles in the old implementation.

4. PERFORMANCE TEST RESULTS

To compare the performance of the two implementations, we synthesized each implementation as part of a SoC, programmed the SoCs onto the same FPGA boards, and ran 3 types of accelerator applications on the FPGAs. Both SoCs have three different accelerators for the three applications: Fast Fourier Transform (FFT), General MAtrix Multiply (GEMM), and 2D convolution (CONV2D). For each application, we used 5 workload sizes from extra small (XS) to extra large (XL). We also tested each application under 2 different coherence modes: LLC-Coherent (Accelerators have no L2 cache and only interface with the LLC for coherence) and Coherent (Accelerators have L2 caches that are coherent with other L2 caches in processors). While running the applications, we measure the number of cycles that the accelerator spends communicating with memory.

Figure 4 shows the speedups calculated from the results of our testing. While the speedup for CONV2D stays relatively constant across different workloads, the speedup is much larger for FFT and GEMM at smaller to medium workloads. This is to be expected, as these workload sizes will allow the accelerators to benefit the most from the improved DMA execution of the pipelined LLC. For larger workloads, the accelerators will be primarily receiving un-cached data from main memory, so the speedup is not as significant.

5. REFERENCES

[1] Johnathan Alsop, Matthew D. Sinclair, and Sarita V. Adve. Spandex: A flexible interface for efficient heterogeneous coherence. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ISCA '18, page 261âĂŞ274. IEEE Press, 2018.

[2] Tianrui Wei, Nazerke Turtayeva, Marcelo Orenes-Vera, Omkar Lonkar, and Jonathan Balkind. Cohort: Software-oriented acceleration for heterogeneous socs. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and*

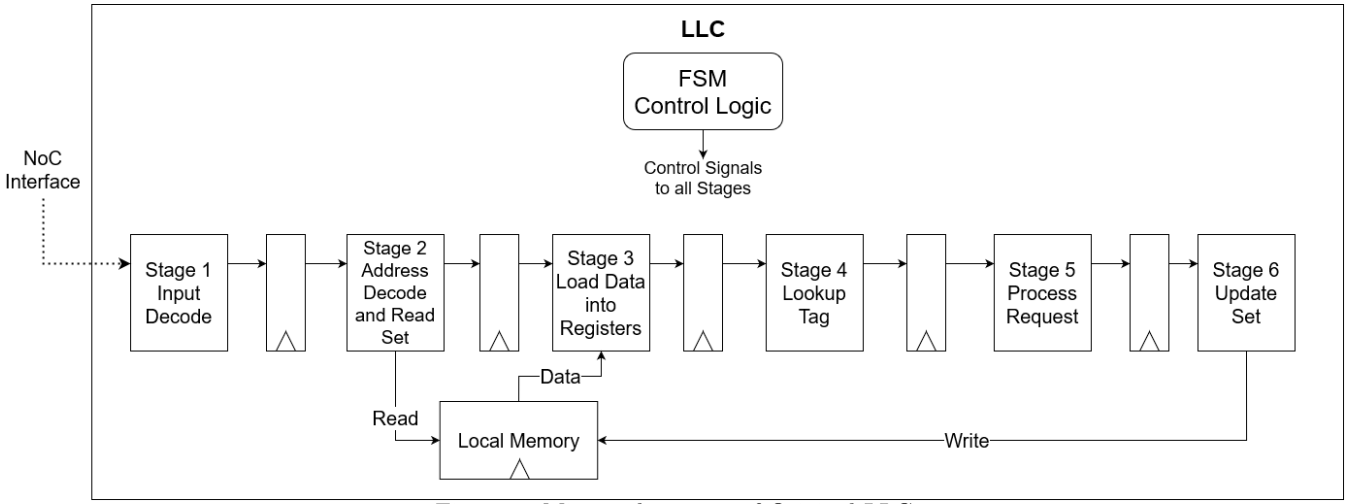

Figure 2: Microarchitecture of Original LLC

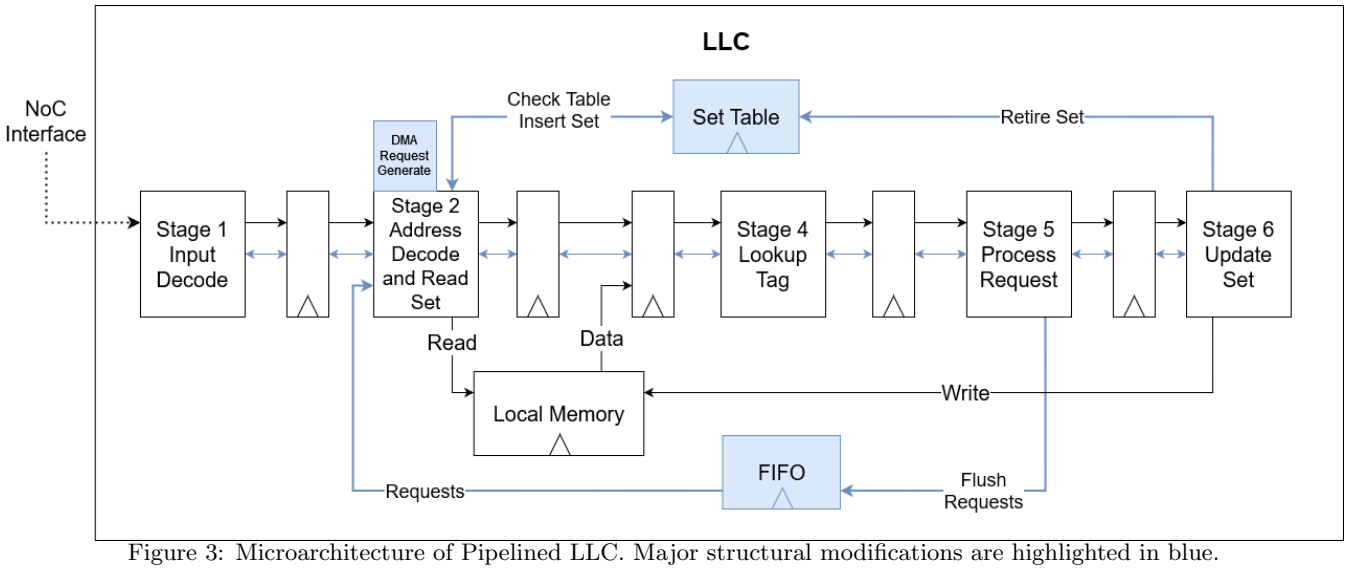

Figure 3: Microarchitecture of Pipelined LLC. Major structural modifications are highlighted in blue.

Operating Systems, Volume 3, ASPLOS 2023, page 105âĂŞ117, New York, NY, USA, 2023. Association for Computing Machinery.

[3] Ang Li, August Ning, and David Wentzlaff. Duet: Creating harmony between processors and embedded fpgas, 2023.

[4] Paolo Mantovani, Emilio G. Cota, Christian Pilato, Giuseppe Di Guglielmo, and Luca P. Carloni. Handling large data sets for high-performance embedded applications in heterogeneous systems-on-chip. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, pages 3:1–3:10, New York, NY, USA, October 2016. ACM.

[5] Davide Giri, Paolo Mantovani, and Luca P. Carloni. Noc-based support of heterogeneous cache-coherence models for accelerators. In *2018 Twelfth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pages 1–8, 2018.

[6] Davide Giri, Paolo Mantovani, and Luca P. Carloni. Accelerators and coherence: An soc perspective. *IEEE Micro (Special Issue: Hardware Acceleration)*, 38(6):36–45, November 2018.

[7] Joseph Zuckerman, Davide Giri, Jihye Kwon, Paolo Mantovani, and Luca P. Carloni. Cohmeleon: Learning-Based Orchestration of Accelerator Coherence in Heterogeneous SoCs. In *Proceedings of the IEEE/ACM Symposium on Microarchitecture (MICRO)*, 2021.

[8] Paolo Mantovani, Davide Giri, Giuseppe Di Guglielmo, Luca Piccolboni, Joseph Zuckerman, Emilio G. Cota, Michele Petracca, Christian Pilato, and Luca P. Carloni. Agile SoC Development with Open ESP. 2020.

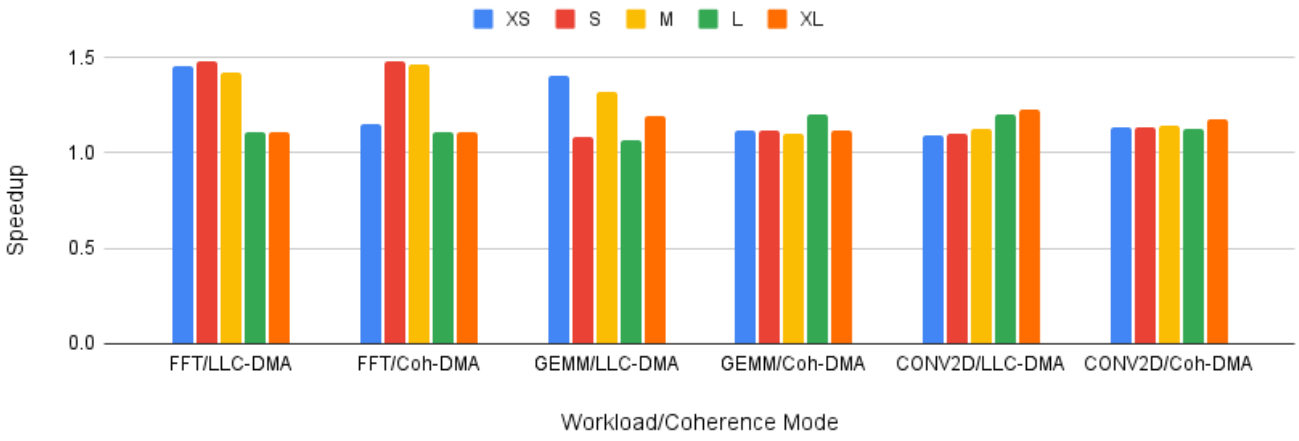

Figure 4: Speedup of Pipeline LLC compared to Original LLC for various sizes of applications under different coherence modes. LLC-DMA refers to LLC-Coherent, and Coh-DMA refers to Coherent.