Pipelining the ESP Last-Level Cache

COMS E6901 - Projects in Computer Science - Fall 2022

Kevin Yunchuan Jiang
kyj2112@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 acceses for a variety of different workloads [6, 7].

ESP is an open-source platform for hetergeneous 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 acceess their data directly from the LLC.

2. THE ESP CACHE HIERARCHY

The ESP cache hierarchy consists of L1/L2 caches and a LLC in order to achieve cache coherence on SoCs with multiple processors and accelerators. Figure 1 provides a high level functional view of each component of the hierarchy. The directory controller refers to the LLC. The LLC interfaces directly with main memory, and services requests from private cache controllers or DMA controllers. The private cache controllers refer to the L1/L2 caches processors (μP) or accelerators, which service memory requests from the processors and sends requests to the LLC if needed. The DMA controller belongs to accelerators without private caches. Rather than interfacing directly to main memory, these accelerators can receive data directly from the LLC if available.

Compared to the L1/L2 caches, the LLC contains additional metadata for the cache lines stored inside the L1/L2 caches at any point in time, such as which L1/L2 caches share the cache line or which L1/L2 owns the cache line. These metadata are necessary for the execution of the extended MESI directory-based cache coherence protocol. MESI stands for the four states that a cache line can be in at any point in time (Modified, Exclusive, Shared, Invalid). The ESP LLC also maintains cache lines that are no longer being used by any L1/L2 cache in a Valid state. These cache lines are the first choice for evictions but can also be sent directly to accelerators if requested through DMA, which is much faster compared to accelerators reading the data directly from main memory. Valid cache lines can also be sent directly to the L1/L2 caches if requested.

Figure 2 shows the components of the cache hierarchy in the context of an example ESP tile grid system. The LLC is located on memory tiles, while and the DMA controllers and L1/L2 caches are located on accelerator tiles and processor tiles. The LLC communicates with the private caches and DMA controller through the ESP Network on Chip (NoC), which reserve 5 separate planes for cache coherence requests/responses and DMA requests/responses.

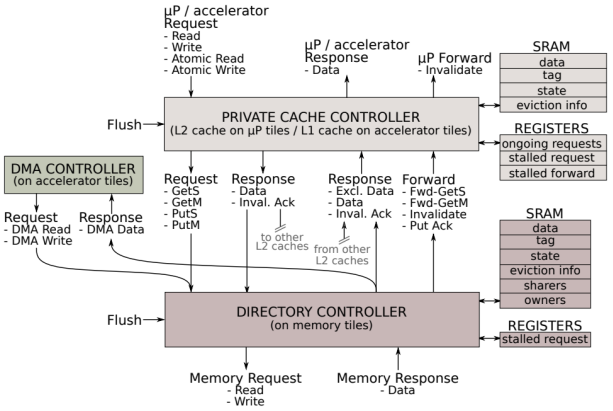

Figure 1: ESP Cache Hierarchy [5]

3. THE ESP LLC IMPLEMENTATIONS

3.1 Original LLC

The original LLC implements a multi-cycle datapath for handling requests according to the extended MESI protocol. The multi-cycle datapath consists of six stages, and the LLC contains an FSM to control each stage of the datapath. The FSM transitions between six states when servicing a request, each state corresponding to a stage in the datapath. When

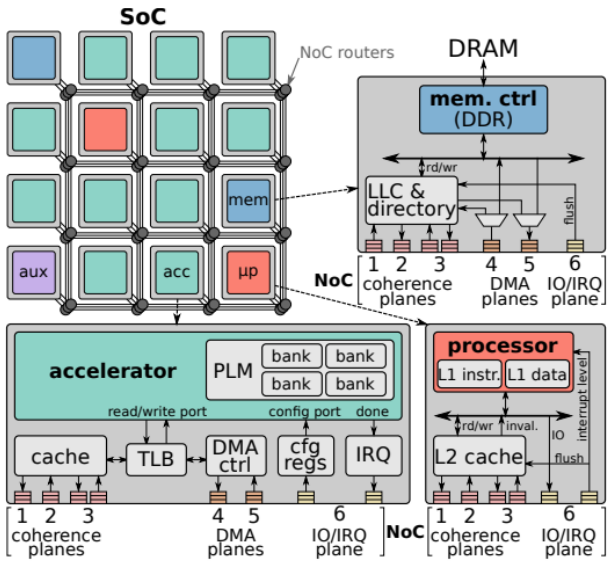

Figure 2: Detailed ESP Tile Grid [5]

servicing a requests from a private cache or a DMA controller, only one stage of the datapath is active during any state of the FSM. The first stage is responsible for accepting new requests coming in from the NoC. However, during the later states of servicing a request, the earlier stages including the first stage will be idle, building up backpressure against later requests.

3.2 Pipelined LLC

The pipelined LLC implements a six-stage pipeline datapath for handling requests. Each stage is taken from the multi-cycle datapath and modified with additional logic that allows for backpressure between individual stages rather than just between the entire LLC and the NoC. Because there is no longer an FSM for controlling the datapath, all stages are always active and handling a request, while the first stage still provides backpressure to the NoC if theres is backpressure from later stages. This implementation eliminates idle times in earlier stages and allows multiple requests to exist the datapath simultaneously.

Converting a multi-cycle datapath into a pipeline datapath presents numerous challenges. While later stages in the multi-cycle datapath can rely on the outputs of earlier stages to stay constant throughout the entire duration of servicing a request, this is not the case in the pipeline datapath, where new requests will enter earlier stages before older requests have finished traversing all pipeline stages. These signals will inevitably change before the completion of one request, and therefore required careful redesign to enable them to be sent through the pipeline.

Another challenge is avoiding read-after-write (RAW) hazards and read/write collisions. The ESP LLC implements a set-associative cache, meaning the local memory of the LLC is addressed with the set portion of a memory address. Two stages in the pipeline interface with the local memory in LLC. The second stage reads from local memory, while the sixth stage writes to local memory. To prevent the two hazards mentioned, whenever there are two requests addressing the same set, the second stage can handle the later request only after the first request leaves the sixth stage and writing is complete. Our solution is adding a set table which keeps track of the sets of all requests currently in the pipeline datapath. The second stage checks the set table before handling a new request, while the sixth stage removes a set from the table once writing is complete. While these hazards do not apply to requests addressing different sets, the local memory in the multi-cycle datapath was not configured for reading and writing to different sets in the same cycle. In the pipeline datapath, the local memory is configured to be capable of same cycle read/write of different sets in order to maximize throughput of independent requests.

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 L1/L2 for eviction, and when there is a cache line in a transient state waiting for a data response from L1/L2. In both cases, the LLC should not service any newer requests until the data response from L1/L2 has been recieved and completely serviced. However, these cases are realized only at the fifth stage, and newer independent requests will enter earlier stages before the data response. 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, and the 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 signifacntly improved DMA speed for accelerators, the pipelinined LLC takes it a step further. Large DMA request spans multiple cache lines, with each cache line functionally acting as a new independent "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. This provides a significant speedup if the data being requested exists in the LLC, because later stages do not need to request data from main memory and force backpressure to earlier stages.

4. 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 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.