IJFANS INTERNATIONAL JOURNAL OF FOOD AND NUTRITIONAL SCIENCES

ISSN PRINT 2319 1775 Online 2320 7876

Research paper Volume 11, Iss 9, 2022

On-Chip Cache Procedure and Device for Efficient CPU and GPU Access Request Arbitration

M.Maria Sampoornam, S.Saroja Devi, S.Madeline Arockiya Shiney

Assistant Professor, Department of Information Technology, J.J. College of Engineering and Technology, Trichy, Tamilnadu

Assistant Professor, Department of Information Technology, J.J. College of Engineering and Technology, Trichy, Tamilnadu

Assistant Professor, Department of Information Technology, J.J. College of Engineering and Technology, Trichy, Tamilnadu

DOI:10.48047/IJFANS/11/9/358

Abstract:

This research paper presents a novel on-chip cache procedure and device that effectively handles access requests from both CPUs and GPUs. The proposed procedure involves classification caching based on the access request type, arbitrating different types of access requests for caching, and optimizing access time for CPU requests through cache while bypassing cache for GPU requests. The device includes CPU and GPU request queues, a moderator, and cache performance elements. By considering the distinct access characteristics of CPUs and GPUs simultaneously, this approach offers high performance, simple hardware implementation, and minimal cost.

Keywords: CPU, GPU, on-chip cache, access request, classification caching, arbitration, performance optimization.

Introduction:

Modern computing systems rely on both central processing units (CPUs) and graphics processing units (GPUs) for executing diverse workloads. CPUs excel at general-purpose tasks, while GPUs are specialized for graphics-intensive computations. Efficiently managing the access requests from these two processing units is critical for optimizing system performance. This research introduces an on-chip cache procedure and device that addresses the challenges associated with handling CPU and GPU access requests simultaneously.[1]

ISSN PRINT 2319 1775 Online 2320 7876
Research paper Volume 11, Iss 9, 2022

Background:

With the rapid development of super large-scale integration and embedded technology, the transistor resources available on a single chip have significantly increased, leading to the emergence of System-on-Chip (SoC) technology. SoC chips represent the most integrated form of multiple functional components, encompassing various IP kernels. They offer comprehensive functionality, allowing for the integration of almost all the features of an embedded information processing system, such as mobile phones and PDAs, onto a single chip. This integration enables information gathering, input, storage, processing, and output on a single chip.[1,2]

Modern embedded systems, including mobile phones and game consoles, place high demands on multimedia processor performance, particularly for graphics, images, and videos. As a result, Graphics Processing Units (GPUs) are often integrated into SoC chips. In such chips, which integrate two different processing units, namely CPUs and GPUs, it becomes necessary for them to share and utilize on-chip resources, such as cache and storage control.[3]

However, the limited on-chip memory bandwidth often fails to meet the high bandwidth requirements of both CPUs and GPUs simultaneously, impacting the performance of both processing units. Moreover, CPUs and GPUs exhibit different memory access characteristics, which also pose distinct requirements for on-chip cache. CPU access requests are typically latency-sensitive, requiring quick service, while GPU access requests are bandwidth-sensitive, necessitating high-bandwidth service to ensure real-time image processing.[4,5]

Consequently, the shared utilization mode of on-chip cache has a certain impact on the performance of both CPUs and GPUs, as it becomes challenging to meet the low latency demands of CPUs and the high bandwidth requirements of GPUs simultaneously. As the integration of CPUs and GPUs on SoC chips continues to increase, the issue of memory access contention between the two processing units becomes a pressing technical problem that needs urgent resolution.[6]

In summary, the integration of CPUs and GPUs on SoC chips presents challenges related to memory access contention. The limited on-chip memory bandwidth affects the performance of both processing units, and the divergent memory access characteristics of CPUs and GPUs impose distinct requirements on on-chip cache. Addressing these challenges and optimizing the management of access requests from CPUs and GPUs in an efficient and effective manner is crucial for achieving high-performance computing on SoC chips.[7]

ISSN PRINT 2319 1775 Online 2320 7876
Research paper Volume 11, Iss 9, 2022

Traditional cache designs primarily cater to CPU access requests, leading to suboptimal performance when GPUs are involved. GPUs often exhibit different access patterns and prioritize large-scale memory operations, making the traditional cache management approach less effective. To overcome these limitations, a novel on-chip cache procedure is proposed, which takes into account the unique access characteristics of both CPUs and GPUs.[8,9]

Research Objective:

The primary objective of this research is to develop an on-chip cache procedure and device that effectively handles CPU and GPU access requests by incorporating classification caching and arbitration mechanisms. The research aims to optimize CPU access time through cache while ensuring efficient memory operations for GPU requests. The procedure is designed to achieve high performance, simplicity in hardware implementation, and minimal cost. The proposed procedure involves classifying access requests based on the originating CPU or GPU, arbitrating different types of access requests for caching, and optimizing CPU access request time through cache. For CPU requests, data is read from or written to cache, whereas GPU requests bypass the cache and directly access external memory storage, except for cache hits during write operations. The device consists of CPU and GPU request queues, a moderator for access arbitration, and cache performance elements as shown in below table.

Aspect	CPU	GPU
Purpose	General-purpose processing	Parallel processing for graphics and compute-intensive tasks
Architecture	Fewer cores, optimized for sequential tasks	More cores, optimized for parallel tasks
Clock Speed	High clock speeds (GHz range)	Moderate clock speeds (GHz range)
Instruction Set	Complex instruction set architecture (CISC)	Reduced instruction set architecture (RISC)
Cache	Larger cache sizes	Smaller cache sizes
Memory Access	Lower memory bandwidth	Higher memory bandwidth
Floating-Point Precision	Support for higher precision (e.g., 64-bit floating point)	Support for lower precision (e.g., 32-bit floating point)

ISSN PRINT 2319 1775 Online 2320 7876
Research paper Volume 11, Iss 9, 2022

Power Consumption	Lower power consumption	Higher power consumption
Cost	Generally more affordable	Generally more expensive
Use Cases	General computing tasks, multitasking, single-threaded applications	Graphics rendering, video encoding, machine learning, gaming, parallel computing

Research

The research aims to provide a procedure and device that can effectively share the on-chip cache between CPUs and GPUs, considering their different access characteristics, while achieving high performance, simplicity in hardware implementation, and low cost. The technical solution proposed in this research involves the following steps:

Classification Caching:

The access requests from the CPU and GPU are classified and categorized according to their respective sources.

Arbitration of Different Types of Access Requests:

The different types of access requests are arbitrated to determine which requests will enter the cache pipeline. This arbitration process considers the priority state value, representing the priority level for different caching classifications. If the priority state value indicates a preference for the CPU, an access request from the CPU is selected as the winner and allowed to enter the pipeline. Similarly, if the priority state value indicates a preference for the GPU, an access request from the GPU is chosen as the winner and enters the pipeline.

Handling Access Requests:

The request type of each access request entering the pipeline is checked, and the corresponding actions are performed based on the type.

3.1) CPU Access Request:

If the request type is from the CPU, further actions are taken based on whether it is a read or write operation.

For a read operation, the cache is checked to determine if the requested data is already present (cache hit). If the data is found in the cache, it is directly returned to the CPU core. If the data is not in the cache (cache

ISSN PRINT 2319 1775 Online 2320 7876
Research paper Volume 11, Iss 9, 2022

miss), it is fetched from the external memory storage, cached in the cache, and then returned to the CPU core.

For a write operation, the cache is checked for a cache hit. If the data to be written replaces existing data in the cache, the new data is written, and an order is sent to cancel or update private data backup in the CPU core. If it is a cache miss, the data is written into a newly assigned cache block address based on the cache's writing distribution principle.

3.2) GPU Access Request:

If the request type is from the GPU, similar actions are performed for read and write operations as in the CPU access request.

For a read operation, the cache is checked for a cache hit. If the data is present, it is directly returned to the GPU. If it is a cache miss, the requested data is fetched from the external memory storage and directly returned to the GPU without being written into the cache.

For a write operation, the cache is checked for a cache hit. If the data to be written replaces existing data in the cache, the new data is written, and an order is sent to cancel or update private data backup in the CPU core. If it is a cache miss, the data is written to the external memory storage without being written into the cache.

Classification of Access Requests:

The procedure begins by classifying the access requests from the CPU and GPU based on their respective memory access features. The access requests are categorized into different types.

Arbitration of Caching Access Requests:

The different types of access requests for caching are arbitrated to determine which requests will enter the streamline. This arbitration process considers factors such as priority and characteristics of CPU and GPU access. The access request that wins the arbitration enters the streamline for further processing.

Streamline Execution:

When executing the access requests in the streamline, the procedure distinguishes between CPU and GPU access requests and performs the corresponding actions.

3.1) CPU Access Request:

For an access request from the CPU, the procedure performs the access request using the on-chip cache. The read-write data of the access request are processed through the cache. If it is a read operation, the

ISSN PRINT 2319 1775 Online 2320 7876
Research paper Volume 11, Iss 9, 2022

procedure checks if the data is already present in the cache (cache hit) and returns it to the CPU core. If it is a write operation, the procedure determines if the data replaces existing data in the cache and sends a notification to cancel or update private data backup in the CPU core.

3.2) GPU Access Request:

For an access request from the GPU, the procedure handles the access request differently. The read-write data reading or writing external memory storage are bypassed and directly performed without going through the cache. If it is a read operation, the procedure checks if the data is in the cache (cache hit) and returns it directly to the GPU. If it is a write operation, the procedure determines if the data replaces existing data in the cache and sends a notification to cancel or update private data backup in the CPU core.

Optimization for CPU and GPU Access Characteristics:

The procedure optimizes the process based on the memory access features of the CPU and GPU. For CPU programs, which exhibit higher data locality, the procedure improves program performance by accessing data that enters the cache during streamline execution. On the other hand, for GPU programs, which have poor data locality and exhibit a streaming behavior, the procedure avoids accessing data through the cache, reducing the impact on CPU programs.

Advantages of the Procedure:

The procedure offers several advantages:

It simultaneously considers the different access characteristics of CPUs and GPUs, allowing for high performance.

It introduces cache sharing without significant changes to the existing processor structure, resulting in a simple hardware configuration and low cost.

It provides flexibility in selecting different resolving strategies based on the memory access features of CPUs and GPUs, improving system agility and responsiveness.

Corresponding Device for CPU and GPU Cache Sharing:

The procedure corresponds to a device that enables CPU and GPU cache sharing. The device shares the same technical effects as the procedure, optimizing CPU and GPU access and improving system performance. The details of the device are not repeated here.

In summary, the research proposes a procedure and device for sharing on-chip cache between CPUs and GPUs. The procedure classifies and arbitrates different types of access requests, while the device facilitates

ISSN PRINT 2319 1775 Online 2320 7876
Research paper Volume 11, Iss 9, 2022

cache sharing. The approach optimizes CPU and GPU access characteristics and offers advantages such as high performance, simplicity in hardware implementation, and flexibility in resolving strategies.

Conclusion:

The presented on-chip cache procedure and device offer a practical and efficient solution for managing access requests from both CPUs and GPUs in modern computing systems. One of the key advantages of this procedure is its ability to consider and optimize for the distinct access characteristics exhibited by CPUs and GPUs.

The procedure begins by classifying the access requests from CPUs and GPUs based on their respective memory access features. By categorizing the requests into different types, the procedure can effectively handle and prioritize the access requests during the caching process.

The next step is the arbitration of caching access requests. The procedure determines which types of access requests will enter the streamline based on factors such as priority and the specific characteristics of CPU and GPU access. This arbitration process ensures that the most relevant and important access requests are given priority, enhancing the overall efficiency of the system.

During the streamline execution, the procedure differentiates between CPU and GPU access requests and performs the corresponding actions accordingly. For CPU access requests, the procedure leverages the on-chip cache to process the read-write data. By utilizing the cache, the procedure significantly improves the program's performance by accessing data that has been stored in the cache, thus reducing the latency associated with external memory access.

On the other hand, for GPU access requests, the procedure takes a different approach. It bypasses the cache and directly performs the read-write operations on the external memory storage. This strategy is based on the observation that GPU programs typically exhibit a streaming behavior and have poor data locality. By avoiding the cache and directly accessing the external memory, the procedure minimizes the impact on CPU programs, enhancing overall system performance.

The presented on-chip cache procedure and device provide a practical solution for efficiently managing CPU and GPU access requests in modern computing systems. By considering the distinct access characteristics of CPUs and GPUs, the proposed procedure achieves high performance, simple hardware implementation, and cost-effectiveness.

References:

ISSN PRINT 2319 1775 Online 2320 7876
 Volume 11, Iss 9, 2022

1. Jaeguk Ahn, Jiho Kim, Hans Kasan, Leila Delshadtehrani, Wonjun Song, Ajay Joshi, and John Kim. 2021. Network-on-Chip Microarchitecture-based Covert Channel in GPUs. In MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21). Association for Computing Machinery, New York, NY, USA, 565–577. https://doi.org/10.1145/3466752.3480093
2. Janibul Bashir and Smruti R. Sarangi. 2020. GPUOPT: Power-efficient Photonic Network-on-Chip for a Scalable GPU. J. Emerg. Technol. Comput. Syst. 17, 1, Article 8 (January 2021), 26 pages. https://doi.org/10.1145/3416850
3. Hazarika, A., Poddar, S., & Rahaman, H. (2020). Survey on memory management techniques in heterogeneous computing systems. IET Computers & Digital Techniques, 14(2), 47-60. https://doi.org/10.1049/iet-cdt.2019.0092
4. Li, B., Wei, J., & Kim, N. S. (2021). Virtual-Cache: A cache-line borrowing technique for efficient GPU cache architectures. Microprocessors and Microsystems, 85, 104301. https://doi.org/10.1016/j.micpro.2021.104301
5. M. Khavari Tavana, Y. Sun, N. Bohm Agostini and D. Kaeli, "Exploiting Adaptive Data Compression to Improve Performance and Energy-Efficiency of Compute Workloads in Multi-GPU Systems," 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Rio de Janeiro, Brazil, 2019, pp. 664-674, doi: 10.1109/IPDPS.2019.00075.
6. J. Zhang, M. Jung and M. Kandemir, "FUSE: Fusing STT-MRAM into GPUs to Alleviate Off-Chip Memory Access Overheads," 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), Washington, DC, USA, 2019, pp. 426-439, doi: 10.1109/HPCA.2019.00055.
7. S. Van Winkle, A. K. Kodi, R. Bunescu and A. Louri, "Extending the Power-Efficiency and Performance of Photonic Interconnects for Heterogeneous Multicores with Machine Learning," 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), Vienna, Austria, 2018, pp. 480-491, doi: 10.1109/HPCA.2018.00048.
8. S. Pal et al., "A 7.3 M Output Non-Zeros/J Sparse Matrix-Matrix Multiplication Accelerator using Memory Reconfiguration in 40 nm," 2019 Symposium on VLSI Technology, Kyoto, Japan, 2019, pp. C150-C151, doi: 10.23919/VLSIT.2019.8776507.
9. Jagadish B. Kotra, Michael LeBeane, Mahmut T. Kandemir, and Gabriel H. Loh. 2021. Increasing GPU Translation Reach by Leveraging Under-Utilized On-Chip Resources. In MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21). Association for Computing Machinery, New York, NY, USA, 1169–1181. https://doi.org/10.1145/3466752.3480105

ISSN PRINT 2319 1775 Online 2320 7876

Research paper Volume 11, Iss 9, 2022