Review on – iCHAT : Inter Cache Hardware-Assistant Data Transfer for Heterogeneous Chip Multiprocessors

By:
Anvesh Polepalli
Raj Muchhala
Introduction

• Integrating CPU and GPU into a single chip for performance and power efficiency.

• Example- AMD’s Heterogeneous System Architecture.

• Communication cost between CPU and GPU result in performance degradation.

• Important to share and move data efficiently to achieve good performance and minimal power.
Process of communication

- Two steps:

  1. OWNER’S CACHE
  2. REQUESTOR’S CACHE
  
  UPPER LEVEL MEMORY
GPGPU Transfer Pattern

1. CPU initializes the data
2. do {
3.   GPU kernel executes and computes the data
4.   CPU re-processes data
5. } while (condition)
6. CPU post-processes data

• Data Transfer occurs when switching between CPU and GPU
• When switching to Phase 2, data (initialization data) will be transferred from CPU to GPU to be used.
• Within Phase 2, some particular data blocks (hot data) are frequently transferred between CPU and GPU so that CPU can check the computing progress.
• When switching from Phase 2 to Phase 3, sometime a small amount of data may be transferred back to CPU again
GPGPU Transfer Pattern Cont.

• The initialization and hot data usually take up the majority of communication data.

• For large problem size GPGPU applications or normal graphics applications, the data transfer will cause much more burden on cache controllers, directory and memory system.

• Even though GPU can achieve significant speedups for the kernel computing itself, with the communication overheads the final performance will be withheld from the promised potential.
Baseline Heterogeneous System

- Private L2 and Shared L3. L3 Cache only holds the write back cache from the L2 Cache.

- Heterogeneous System Coherence (HSC) adds ‘Region buffer’ to both CPU L2 and GPU L2 caches to track access permissions at the region granularity.

- All the L2 misses first checks with the region buffer. If valid permission for that region (Shared for reads, Private for writes) is found in the Region Buffer, data requests are sent directly to memory.

- The direct memory accesses have much higher bandwidth, especially for GPU.
Baseline Heterogeneous System Cont.

• If permission is not found, requests are forwarded to Region Directory to acquire permission for the region. The Region Directory connects the Region Buffers and manage the permissions of all the regions on-chip.

• By obtaining permission at region granularity, HSC significantly reduces the bandwidth to the directory compared to the traditional directory based coherence protocols. The grain size used in this paper is set as page size.

• Data between CPU and GPU only transferred on the requestor’s demand.

• When the requestor (either CPU or GPU) needs some data from the owner (GPU or CPU), it has to go through a 4-hop process.
Baseline Heterogeneous System Cont.

• First, the requestor sends a data request to the region directory.

• Second, the directory sends an invalidation request to the owner’s region buffer.

• Third, the owner’s cache evicts the region from owner’s region buffer and writes data from its private cache back to the L3 cache.

• Fourth, the requestor gains the region permission and fetches the data from L3 cache into its own cache.

• The four-hop data transfer results in high latency and significant cache and directory traffic. When the transferred data block is large, the directory will become the bottleneck.
iCHAT

• Inter-Cache Hardware-Assistant data Transfer.

• Detects and learns communication patterns and stores the information about which data blocks have been transferred.

• Predicts when the communication will happen again and starts data transfer before request.

• Reduces communication latency and related cache traffic.
Inter-cache Hardware Communicator
• iCHAT sits between L2 caches and shared L3 Cache, connected to the directory.

• Interacts with cache hierarchies through directory and watch data transfers between CPU and GPU.

• iCHAT has 3 components:
  1. Communication Detector.
  2. Last Evicted Page.
  3. Hot Block Table.
• **Communication Detector** detects communication patterns and predicts when the communication happens.

• **Last Evicted Page** records the last evicted page during transfers of initialization data.

• **Hot Block Table** stores the address of the data blocks frequently transferred.

• iCHAT transfers data ahead of time through these 2 steps:
  1. Communication Detection.
  2. Eager Eviction.
Communication Detection

• Critical for communicator to detect the communication data and their patterns.

• Communication detector captures those information by watching the traffic between CPU and GPU.

• Captured information stored in Last evicted page buffer and Hot Block Table separately.

• Detection mechanisms different for Initialization Data and Hot Data.
Communication Detection (Initialization Data)

• Initialization data has following characteristics:
  1. Usually transferred just once.
  2. CPU streams through each element of data structure to initialize it.

• Initialization data consists of large amount of continuous pages with all cachelines touched by CPU.

• A Validation vector of 64 bits is added to the directory.
Communication Detection (Initialization Data)

• Each bit in the vector represents one cache block of the page to indicate whether the block is in the cache.

• When GPU request CPU’s data, detector checks the validation vector.

• If all the cachelines in the page are accessed, then it is identified as Initialization data.

• Current Page Id stored in Last evicted page buffer for Eager Eviction.
Communication Detection (Hot Data)

• The basic idea to speed up hot data communication is to learn and capture the hot pages which are frequently transferred between CPU and GPU, then predict and proceed the data transfer before communication data is requested.

• Resulting in no wait time for the data to transfer from the owner’s side.

• The major benefit is that it can hide the 4-hop data transfer latency between CPU and GPU.

• Mechanisms to detect and predict the time for communication.
  • Interval-based prediction
  • Write-activation mechanism
  • Requestor initialized
Eager Eviction

• Evicts the detected data from owner’s cache ahead of time.

• iCHAT sends invalidation request through directory to owner’s cache which evicts data blocks and writes back to L3 Cache.

• Eager Eviction requests sent when the directory is free to avoid bandwidth conflicts with processor's data demand.
Eager Eviction For Initialization Data

• Triggered when GPU starts to request the first page of Initialization Data.

• iCHAT communicator checks the validation vector of the page with ID next to the last evicted page.

• If all the cacheline blocks of the next page are in the CPU cache, next page will also be taken as initialization data and eagerly evicted.

• Eviction stops when features of initialization data no longer applies.
Eager Eviction For Hot Data

• Eviction for Hot Data is straightforward.

• Data blocks are recorded in Hot Table block during communication detection.

• When communication is predicted to start, only data blocks recorded in hot block table are evicted.
Eager Eviction Cont.

• With Eager Eviction, when requestor later requests communication data, it fetches directly from L3 cache without waiting for owner to evict the requested data.

• Transforms the original 4-Hop data transfer into 2-Hop transfer, i.e., Request and Fetch.

• Traffic on Directory also reduced.
Communication Detection (Further Optimizations)

• A further optimization technique of eagerly transferring the communication data is to inject them from the L3 cache to the requestor’s private caches.

• L2 cache size is not big as L3, so the communication injection has to be designed to avoid cache pollution.

• The hot pages for GPGPU applications are usually small and also will be consumed pretty soon, so the injection will not cause cache pollution.

• Initialization data blocks are very large, to avoid cache pollution, a threshold based stepped injection mechanism was proposed to be used.
Communication Detection (Further Optimizations) Cont.

• Threshold based stepped injection
  • Dynamically determining an injection page number threshold based on the total number of communication data.
  • Inject threshold number of the pages in the requestor’s cache, when the requestor is starting to consume the data, inject another threshold number of the data pages.

• This is NOT implemented.
Preliminary iCHAT Results

• Preliminary results evaluate the proposed iCHAT for initialization data.

• Eager Eviction can evict data from CPU L2 cache to L3 cache in advance.

• Eviction requests will be reduced as the 4-Hop data transfer reduces to 2-Hops.
Eviction Requests Issued by GPU

(a) The eviction requests issued by the GPU.
Eviction Requests Issued by GPU (contd.)

• Results fall into two groups.

• First seven benchmarks on the left side, traffic reduced by 60% on an average.

• Rest six benchmarks on the right side, traffic reduced by less than 20% on an average.
Total Traffic Through Directory

(b) The total traffic through directory.
Total Traffic Through Directory cont.

• In iCHAT, Directory does not send invalidation requests to the owner, reducing the traffic through directory.

• Total traffic decreases from a few percent to 25% for nm, and 8% on an average.

• Results fall into two groups.

• Total traffic for kmeans benchmark increases due to the miss predicted eviction.
Miss Eviction Rate
Miss Eviction Rate Cont.

• Miss rate for kmeans is very high resulting in an increase in eviction traffic and requires extra requests to bring the miss evicted data back from L3 cache.

• For other benchmarks with relatively high miss rate, there is eviction of some non-initialization data with same features of Initialization data.

• Setting a certain threshold for the number of pages evicted each time is a possible solution.
Quantitative Bounding Experiment Results

• Assumed iCHAT can detect and inject the communication data into CPU’s and GPU’s L1 caches.

• iCHAT can track data at both region and cacheline block granularity, thus this bounding experiment could provide a more accurate evaluation

• Only the data that travels across CPU and GPU, the real inter CPU-GPU transfers, as communication data.

• Ideal iCHAT can move the data immediately into the requestor’s cache.
The quantitative evaluation of inter CPU-GPU transfers out of all data requests.
Quantitative Bounding Experiment Results Cont.

Percentage of requests to CPU-GPU share data out of all data requests
Quantitative Bounding Experiment Results Cont.

Bounding evaluation of speedup with iCHAT’s assistance compared to baseline architecture.
Conclusion

• Categorizes communication data into two classes: initialization data and hot data, is a great step.

• Results seem promising, using iCHAT has reduced cache traffic and the long latency involved in CPU and GPU communication.

• Results showing reductions of 40% in cache eviction traffic and 8% in total directory traffic, is substantial improvement.

• Increase in area cost due to additional hardware is not mentioned.
References

• Junli Gu; Beckmann, B.M.; Ting Cao; Yu Hu, "iCHAT: Inter-cache Hardware-Assistant Data Transfer for Heterogeneous Chip Multiprocessors," Networking, Architecture, and Storage (NAS), 2014 9th IEEE International Conference on, vol., no., pp.242,251, 6-8 Aug 2014

Thank you