Intel® Iris® Xe and UHD Graphics Open Source

Programmer's Reference Manual

For the 2020-2021 11th Generation Intel Xeon®, Core™, Celeron®, Pentium® Gold Processors based on the "Tiger Lake" Platform

Volume 7: Memory Cache

December 2021, Revision 1.0

Notices and Disclaimers

Table of Contents

Memory Cache

This section describes the GFX L3 Cache, which is a large storage that backs up various L2/L1 caches in the GPU's internal units. It supports a simple, way-based partitioning for segregating the cache among groups of clients.

It also has a provision to dedicate a (programmable) section of the storage for the GFX Unified Return Buffer.

L3 Cache

This volume describes the L3 and the Unified Return Buffer. Much of the design is similar to prior generations, with important changes to enhance performance.

Overview

To cater to the bandwidth demands, the L3 cache is organized as multiple independent banks which can be accessed concurrently. The L3 banks are no longer clocked at 2x and hence support one operation per clock.

The L3 cache and URB data form a single contiguous memory space across all the banks and sub banks in the design. The vision is to build a compute scalable cache where with each additional compute block, both the size and bandwidth of L3 Cache are scaled while maintaining the monolithic cache concept. Each added bank becomes a part of a unified cache rather than an independent localized segment. The concept is to be able to keep a single copy of a line and service all requesters via distributing their accesses over many physical cache banks. The L3 cache can operate concurrently in the non-IA-coherent GFX virtual address space as well as the IA-coherent address space.

- Each logical bank consists of:
  - The Data Array - This array stores the actual data
  - The Tag Array - This array stores the tags for the cachelines above
  - The LRU Array - This array stores information that helps determining the cache line that will be evicted when a fill arrives for a set
  - The State Array - This array stores the cache state information (MESI States of the cache lines and some additional internal information)
  - The SuperQ Buffer - This array stores data temporarily on the way in or out of the data array for each access that is in progress
  - the Atomic Processing Units - This unit houses the ALU and associated logic to perform atomic operations on the data
- The rest of the support logic around L3 consists of:
  - The SuperQ: This is the main scheduler of the micro-sequences to be followed for each access to the cache
  - Ingress/Egress queues: These queues buffer incoming accesses and outgoing data on their way into or out of the cache

- CAM structures: These structures are used to maintain coherency in cases of proximal accesses to the same address
- Crossbars: These are used for routing the data to and from the various sub-banks/arrays

Note: A portion of the L3 cache can be allocated as a Unified Return Buffer (URB) region

L3 Bank Configuration

Each L3 bank is configured as described below.

- Each bank consists of a 480KB data-array organized as 120 logical ways
- Up to 104 ways representing 416KB, tagged for L3$, remaining ways treated as dedicated URB.
- The minimum URB size per L3 Bank is 64KB, but this can be programmed up to 128KB.
- 64B Cacheline storage per cell.
- The total L3 array bandwidth is 64B for each core cycle. The bandwidth can be utilized for either a 64B read cycle or a 64B write cycle
- Data protection via ECC.
- 48b virtual addressing support in TAG.
- 1b LRU implementation for selecting the line to be replaced.

L3 Cache Theory of Operation

Following are the L3/URB clients:

L3 Cache Clients	RW/RO
Data Cluster (i.e., spill/fills, load/stores, Global memory accesses)	RW
Sampler (L2$)	RO
IME (Motion Estimation)	RO
Instruction Cache (I$)	RO
State Arbiter	RO
Constant Cache	RO

URB Clients	RW/RO
Local Thread Dispatcher	RO
SF Backend	RO
Stream Out	RO
Clipper	RO
Geometry shader	RO
Tesselator	RO
VF	RW
Data Cluster	RW

The L3 and the URB are separate address spaces with clients capable of accessing only one or the other except for the data port. L3 access/cacheability is determined via a parameter, part of the surface state or base address programming of L3 clients.

Size of L3 Bank and Allocations

General notes

The L3 Cache has been divided into following client pools:

- URB: Local memory space
- DC: Data Cluster Data type
- Color: Color cache allocation
- Z: Depth cache allocation

In addition to these sub-groups, a collection of groups bundle multiple clients under the same allocation set:

- Read-Only (RO) Clients: Inst/State, Constants & Textures (I/S/C/T)
- All L3 Clients (a.k.a "Rest of L3"): DC, Inst/State, Constants & Textures
- Unified Tile Cache: Color/Z combined

Multi-Bank Allocation Options with Tile Cache and Command buffer support

The L3 cache no longer supports the highly banked SLM mode of operations. Out of the total L3 space per bank, a minimum of 64Kbytes will be earmarked for URB storage. The rest can be used as tagged L3 cache. The L3 Cache allocation is done on a per way basis. The Bank programming options allow a varied set of configurations to be programmed in the L3. Broadly, the number of ways allocated to the various sections (URB, DC, RO, Z, C and the command buffer) is selectable. Each of these sections can be allocated a subset of ways out of the total ways. Apart from that, in lieu of the DC and RO sections a single unified “Rest of L3” section can be used. Similarly, in lieu of separate Z and C partitions in the L3, a single Unified Tile Cache programming allows all Z and C streams to share a common section of the L3.

L3 Allocation programming (KBytes per bank)

URB	Rest (DC+RO)	DC	RO (I/S/C/T)	Z	Color	Unified Tile Cache	Command Buffer/ State	Sum
64KB or 128KB	0 to 416 in increments of 8KB	0 to 416 in increments of 8KB	0 to 416 in increments of 8KB	0 to 416 in increments of 8KB	0 to 416 in increments of 8KB	0 to 416 in increments of 8KB	0 to 416 in increments of 8KB	480

Note: The granularity of 8KB delta for each section arises from the fact that the L3 is implemented as a sectored cache with 2 ways per sector. Due to this, the number of ways programmed for each section of the L3 cache should be an even number. The programming of the **L3ALLOCREG** to configure the sections should be determined based on the size of the cache per way. The total number of ways implemented in the design is available in the **L3 Parameter Information register**.

However, several restrictions apply.

- It is not allowed to allocate the entire cache to DC (Data space) with 0KB for reads.
- It is not allowed to have Rest and DC to be "0KB" at the same time. Cache requires either Rest or DC to have at least non-0KB allocation.
- It is not allowed to have Rest and RO to be "0KB" at the same time. Cache requires either Rest or RO to have at least non-0KB allocation.

State is now stored as part of Command Streamer allocation. If specific allocation is 0KB, it will be placed in the Rest section.

L3 Allocation programming (KBytes per bank)									
Config	URB	Rest (DC+RO)	DC	RO (I/S/C/T)	Z	Color	Unified Tile Cache	Command Buffer	Sum
0 (def)	64	416	0	0	0	0	0	0	480
1	128	240	0	0	48	48	0	16	480
2	64	0	32	240	64	64	0	16	480
3	64	144	0	0	128	128	0	16	480
4	64	48	0	0	0	0	352	16	480
5	64	272	0	0	0	0	128	16	480
6	64	400	0	0	0	0	0	16	480
7	128	352	0	0	0	0	0	0	480
8	128	336	0	0	0	0	0	16	480

The number of L3 Banks will vary for different products and SKUs. The number of banks supported for each product is defined in the Configurations section of this document. The total amount of L3$ and URB supported by a product can be calculated by multiplying the number of banks by the values in the above tables.

Note that any allocation for <64KB will limit the bank allocation and proper set distribution. Such programming will lead to set allocation to be less efficient and will not be sufficient to contain the surface unless it is programmed to be natively aligned.

Performance configurations are listed as #5 (deferred rendering) and #6 (immediate and deferred rendering).

L3 Ordering Restrictions

The Super Q will enforce specific ordering requirements on the accesses to the L3/URB but will still allow out-of-order accesses where possible.

Basic Ordering Requirements

The primary purpose of ordering of transactions is to ensure coherency and causality for software accesses to memory. For example, if a thread writes to a memory address, it can expect that any subsequent read issued by the same thread to the same address should read back what it wrote. Conversely, if the read is before the write, it shall receive the value prior to the write.

L3 Cache Allocation Policy

The L3 cache allocation policy is "Allocate on fill". i.e., a line in the cache storage is allocated only upon receipt of the data from the external memory and not at the time that the cache detects a miss. The "Allocate on Fill" policy eliminates many boundary cases by regulating the entry invalidation at the last phase of the data servicing.

If the data already residing in the allocated entry is not modified, then the incoming Fill will overwrite the location. If the allocated entry carries a dirty data, then an eviction is generated, and the dirty data will be moved to the super queue for writing out to memory.

Cache replacement algorithm

The L3 cache uses a 1b LRU algorithm for cache replacement. An N-way 1b LRU has an N-bit vector for each set. The vector is initialized to all 0's. As each Fill request arrives, the first bit in the vector with a 0 is selected, that way is used, and the bit is flipped. Each subsequent fill will search for the first 0 and replace that line and flip its bit. Eventually, when all N bits are 1 and there are no candidates, all bits are cleared.

Any hits to lines present in the cache will update the LRU vector to set the bit corresponding to the way. In this manner, after the first set of allocations are done, any hits will mark the way as "recently used" and avoid replacement. If during a hit operation, a set becomes fully "recent" i.e., all ways have a '1' in the LRU vector, the vector is cleared making way for fresh marking for recent uses.

Memory Object Control State on Cacheability

This 7-bit field is used in various state commands and indirect state objects to define L3/LLC cacheability, memory type, and graphics data type for memory objects.

Note that memory type information from state is used for non-IA compatible paging structures (legacy context). For new context definition where IA compatible (IA32e) paging structures are used, memory typing follows the IOMMU defined structures.

MOCS[6:1] in L3 is used as an index to a set of programmable tables starting with address xB020h. GFX Software can set up the tables as part of the h/w context, and program various index values in surfaces to point to a table that best suits for that particular surface.

L3 Coherency

Coherency is one of the crucial topics within L3, there are multiple levels of coherency that are checked and ensured via L3. The list of domains and flows is dependent on the usage models however basic premise is always the same.

The coherency levels:

1. Thread Level Coherency within a Thread Group
2. Thread Group Coherency between multiple domains
3. GPU/IA level coherency

Besides these special domains basic producer/consumer models are followed which are listed as:

1. Fixed function is producing
2. Data Port is producing

When textures are stored as compressed in L3, the expectation is to ensure that the compressed RO-content is invalidated before same surface can produced as R/W.

Thread Level Coherency

A given thread group is contained within a sub-slice, where its writes and reads target the L3 for global memory and SLM for shared local memory. Given the shared local memory view is the same for all sub-slice accesses, coherency or data sharing is guaranteed within the thread group. Local syncs are executed up to Data Port boundary and not exposed to L3.

Thread Group Coherency

Thread groups can be distributed to multiple sub-slices that are physically far from each other. The coherency between thread groups can only be maintained for their global memory accesses. There are two implications of this coherency depending on the mode we are operating at:

1. Non IA-Coherent L3 mode: This is the same methodology that was introduced with the addition of GT4 support. The cross-thread group coherency is maintained via Sync Global which is processed by L3 as well as introducing WT mode to be able to update global memory with latest data. This mechanism is not expected to be used.
2. IA-Coherent L3 mode: For the new mode of operation, there is no need to have WT behavior. The data in L3 is already visible to all consumers (i.e. other thread groups of GPU or IA cores). Sync'global has no affect given the data in L3 is already globally visible to all consumers.

GPUIA Level Coherency

In non-coherent L3 mode, data sharing between GPU and IA happens via a s/w controlled flow which requires the internal GPU caches to be flushed in order to make the data visible, similarly same caches need to be invalidated when IA produces data. This mode of operation is still available. Within the L3, non-coherent accesses use "virtual" addresses and are tagged in the L3 Tag array using the Virtual/Physical bit.

Coherent memory is also supported where L3 content is visible to IA via snoops. This allows certain streams (i.e. data port) to be GO (Globally Observable) for IA once posted to L3, allowing shorter loop for completions and eliminating the need to do an entire pipeline flush over L3 to get it sync'ed to IA coherent domain. Coherency enhances the general programmability of GPU when cooperating with IA. Within the L3, coherent accesses use a full 48-bit physical address and are tagged using the Virtual/Physical bit in the l3 Tag array.

Coherency Usage Models

This section is to give some examples of usage models and high-level handling within L3. They are specific to L3 flows and not meant to represent over coherency usage models on the system level.

Fixed Func Producing (Push Constants)

Push constants are also similarly processed; in fact, the target is the same: URB. However, URB accesses are processed with their URB offsets via TAG, their original address is based on virtual memory pointing to buffer location in memory. The delivery mechanism makes the CAMs not possible hence the completions are managed once access is deemed towards SuperQ of the corresponding L3 Bank.

There are additional constraints independent of L3 handling of push constants which are defined as part of the Global Arbitration fabric.

EUs Producing via HDC

Data port is the producer for Global and Shared Local memory types. The shared local memory is specific to a data port, hence L3 does not have to do anything special to maintain coherency but to keep accesses to SLM in-order.

For Global memory coherency is maintained via combination of many mechanisms.

1. Completion tracking: Each Data Port tracks their writes towards L3 and wait for them to reach to GO. GO message is given by L3

a. For Non-coherent/virtual addressed Write: Once the ingress queue retires the write in the corresponding node to targeted bank.

b. For Coherent/physical addressed Write: Once the ownership is obtained for the write (i.e. read-for-ownership or invalid-to-modified) which means write is GO with respect to IA cores.

GO information is used within data port to make sure handle releases are gated until the producer’s updates are globally visible.

2. Thread Level Flush: EU threads have the capability to push globally tagged data from L3 to next level caches. Usage is not recommended.

Invalidation and Flushes

There are two different sources to initiate flushes and invalidations:

- Command Streamer initiated
- EU/thread initiated

Both cases have two modes of operation

- Non-IA coherent L3
- IA coherent L3

In addition, there are side flows that back up the various flows and they do require invalidation/flush sequences to be executed for their purpose. Their behavior has no impact between two modes of L3 operation.

- Global Invalidation
- Power Management Invalidation

Command Streamer Invalidation Flows

The Command streamer can initiate the following flush/invalidation flows.

1. **Top of the pipe invalidations**: These are direct, asynchronous commands from the command streamer unit to L3 in respective slices.
2. **Pipeline flush**: These commands flow through the pipe and are issued through the data port to the L3 cache in each slice.

Non-IA Coherent Flows

This is the traditional flow where the content of L3 needs to be invalidated or flushed.

Top of the Pipe Invalidations

For this case, the invalidation causes the cache to drain all FIFOs and pipelines, and the node performs all invalidations. Only after completion in all slices will new transactions be sent to the L3.

The Top of the Pipe invalidation is intended for cases where invalidation and completion need to be coordinated between slices. Therefore, each slice performs invalidation, but will not proceed until all slices have completed the invalidation.

Pipeline Flush

Pipeline invalidation is much simpler and optimized for performance. In this case there is no need to coordinate between slices and the operation is more like a "sync" point. Transactions arriving after the invalidation request will still be queued in the SQ but will only be processed after the invalidation of all candidate lines in the cache is complete. The invalidation acts like a fence.

The pipeline flush send via the dataport will only flush dataport generated allocations. In previous generations this was not optimized and used flush all content of L3 cache.

IA-Coherent Flows

In principle IA-coherent flow is same from the L3 cache perspective, the only difference is within the banks given the data that we are trying to flush or invalidate is already coherent with IA. Such case eliminates the need to push any bank content explicitly out to LLC.

The L3 bank will ensure that the transactions in flight are all globally observable when a flush/invalidation is received.

EUThread Flows

There is continued support for the EUs to perform flush/invalidation events that are not coordinated with other EUs. The data cluster will relate the request to the corresponding L3 and require the entire content of this buffer to be invalidated or flushed to relative coherency domain.

The IA-coherent vs non-IA coherent treatment is same as command streamer flows and still applicable for EU/Thread Flows for invalidation/flush.

Power Management Invalidation

Given we have a IA-coherent mode where a standard pipeline flush does not push the modified lines to outside of GT and we are progressing to deferred invalidations, we need a way to ensure L3 content is cleared and deferred events are complete. This is where PM interaction will have to be introduced.

Corresponding PM will message to L3 config agent to request a flush when needed. Main usage mode is prior to entering RC6. The rest of the treatment in the nodes is the same. Once flush/invalidate event is complete, message(s) will be sent back to PM with the completion status.

L3 Cache Error Protection

L3 cache error protection is covered via ECC (SECDED). All accesses are subject to ECC protection where single bit errors are fixed silently. Double bit errors are reported via a register structure and communicated by an interrupt to GFX driver. L3 cache HW is additionally capable of stalling execution upon a double bit error.