Intel® Arc™ A-Series Graphics and Intel Data Center GPU Flex Series

Open-Source Programmer's Reference Manual

For the discrete GPUs code named "Alchemist" and "Arctic Sound-M"

Volume 7: Memory Cache

March 2023, Revision 1.0

Notices and Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks

Customer is responsible for safety of the overall system, including compliance with applicable safety-related requirements or standards.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exceptions that a) you may publish an unmodified copy and b) code included in this document is licensed subject to Zero-Clause BSD open source license (0BSD). You may create software implementations based on this document and in compliance with the foregoing that are intended to execute on the Intel product(s) referenced in this document. No rights are granted to create modifications or derivatives of this document.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

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.

L3 Cache

The L3 cache is a multiple input multiple output system that operates at high bandwidths through sub-banking.

The features and functionality of the L3 cache are described in greater detail in the following sections of this document.

Overview

In order to cater to the bandwidth demands, the L3 cache is organized as multiple independent banks each of which can perform one operation (read or write) every clock. The banks in the design are address hashed. i.e., a single address is stored in only one of the banks and that is determined by a hash value generated out of various bits of the address. The L3 cache thus forms a single contiguous memory space across all the banks and sub banks in the design. This gives rise to a 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 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

L3 Bank Configuration

Each bank of the L3 cache is implemented as a set associative cache with varying number of sets and ways as demanded by the overall size of the cache. Various SKUs of the IP are equipped with different number of banks potentially differing in sizes too. The size of the cache for each SKU is specified in the configuration section of this document.

The following aspects of the configuration are, however, common to all the implementations.

- The cacheline size is fixed at 64B. However, access granularities down to a byte in width are supported in the design. The cache subsystem will perform appropriate read-merge operations to support partial write accesses.
- The cache is also often sectored with 2-4 cachelines forming a sector in order to support reduced tag storage and to ais the co-location of certain cachelines for allowing locality of access. Details about sectoring can be seen in subsequent sections in this chapter.
- The L3 cache also allows sectioning of the cache to allow clients to be allocated a certain reserved space within the cache. Details can be seen in subsequent sections in this chapter.
- All L3 cache data is protected by ECC.

Bandwidth and Throughput Capability

The table below shows the throughput capability for the L3$ per bank. Note that the URB and SLM have been separated from the L3$ and their throughput is no longer dependent on the L3$ bandwidth. The total system throughput is derived by multiplying the throughput numbers below by the total number of L3 Banks supported by a product. It is to be noted that there are additional limitations on throughput based the source of the transactions. These limitations can, sometimes, be more restrictive than the L3$. For example, the typed data port can generate only 32B of write per clock. These numbers are to be used in conjunction with the source limitations to arrive at the final throughput.

Note also, that the read and write bandwidths are not exclusive i.e., the throughput is limited to one operation per clock.

Memory Type	Read Throughput (Bytes/Clock)	Write Throughput (Bytes/Clock)	Atomic Throughput (32b Ops/Clock)
L3$	1*64	1*64	8

L3 Access Ordering

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. The primary purpose of ordering transactions is to ensure coherency and causality for software accesses to memory. 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 read back 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 and pipeline moves on. 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 scheme for cacheline 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 replaced, 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, and the first way is selected. Post this, any access to the cacheline that results in a hit will cause the bit to be set. This will deselect that way from getting re-allocated until all the other ways are also accessed.

L3 Cache clients

The following are the L3 cache clients:

L3 Cache Clients	RW/RO
Data Cluster (i.e., spill/fills, load/stores, Global memory accesses)	RW
Sampler (L2$)	RO
Instruction Cache (I$)	RO
State	RO
Constant Cache	RO
Copy Engine	RW
Command stream	RW

Note: The above internal clients have an access path to the L3 cache. However, the actual cacheability settings of surfaces and their page tables will determine whether the accesses get allocated into the L3 cache.

Cacheability control for accesses

This 7-bit field is used in various state commands and indirect state objects to define L3/LLC/eDRAM 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.

The L3 cacheability is controlled by a combination of the factors listed below

1. Cacheability request from the instruction (Valid only for EU accesses through the data port)
2. Legacy state programming based cachebility (Memory Object Control State -- MOCS)

At each level, an uncacheable or cacheable behavior can be requested. The cacheable state is considered the higher state. A line will be allowed to cache in the L3 only if both "allow" cacheability. On the other hand, lowering the cacheability state to uncacheable can be done by any level.

L3 Bank Allocations

The following section describes the L3 cache bank allocation programming.

Size of L3 Bank and Allocations for Virtual Cache

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

The L3 does not support the highly banked SLM mode of operations. The URB storage has also been moved out of the L3$. With this change, the entire storage of L3 space can be used only as a tagged L3 cache.

The Bank programming options allow a very varied set of configurations to be programmed in the L3. Broadly, the number of ways allocated to the various sections (DC, RO, Z, C and the command buffer) is selectable. Each of these sections can be allocated any number 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. The following table reflects the programmability of the configuration.

L3 Allocation programming (KBytes per bank)							
Rest (DC+RO)	**DC**	**RO (I/S/C/T)**	**Z**	**Color**	**Unified Tile Cache**	**Command Buffer/ State**	**Sum**
0 to 320/512 in increments of 8KB	0 to 320/512 in increments of 8KB	0 to 320/512 in increments of 8KB	0 to 320/512 in increments of 8KB	0 to 320/512 in increments of 8KB	0 to 320/512 in increments of 8KB	0 to 320/512 in increments of 8KB	320/512

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 (with each way representing 4KB). 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 the CS specific allocation is 0KB, state will be placed in the Rest section.
- Though the theoretical minimum number of ways for a given section is 2, due to the set address calculations, there is a potential hotspotting possibility for sections that are narrower than 8ways. For performance reasons, allocating lower than 8 ways to any section should be avoided.

The following table reflects programming options:

L3 Allocation programming (KBytes per bank)						
Config	Rest (DC+RO)	R/W	RO (I/S/C/T)	Unified Tile Cache	Command Buffer	Sum
0 (def)	512	0	0	0	0	512
1	384	0	0	128	0	512
2	480	0	0	0	32	512
3	256	0	0	256	0	512
4	0	128	352	0	32	512
5	256	0	0	224	32	512
6	504	0	0	0	8	512

Generic note: 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. The total amount of L3$ supported by a product can be calculated by multiplying the number of banks by the values in the above tables.

L3 Cache Error Protection

L3 cache error protection is covered via ECC (SECDED.) All transactions coming into and going out of the L3 cache subsystem are also covered through parity bits. The complete coverage of error detection, correction and reporting is covered in a separate section on RAS.