TDD:P11N: VPoM (Virtual Pool of Memory)

Contents

Abstract

Background of Invention

- VPoM (Virtual Pool of Memory) system is proposed to solve the stranded memory problem which can be happened even when host uses CXL Memory Module (CMM) to expand the memory capacity. VPoM system is targeted to be used in the large scale high performance computing system, such as inference/training systems for large language model, and/or mixture of expert systems. The first invention describes overall system architecture of VPoM, and essential operations to create and to use (read/write) VPoM instance, which includes methods to build address space, and methods to route the memory transaction request packets.
- VPoM also has special features which minimize the possible performance bottleneck of VPoM sytems. Each special feature will be covered in a separate invention.
  - The second invention will cover VPoM coherence management scheme which helps scale-out the VPoM systems by minimizing the overhead of transaction monitoring and signaling for coherence management.
  - The third invention will cover VPoM scheduled prefetch mechanism to minimize the latency penalty of remote memory read in VPoM systems.
  - The fourth invention will cover VPoM stream processing architecture to reduce the traffic overhead in the interconnect network for memory pool (VPoM).

Summary of Invention

- VPoM (Virtual Pool of Memory) is a shared memory system virtually constructed with a set of donated memory regions from individual CXL Memory Modules (CMMs). VPoM provides more consistent memory bandwidth and lower latency compared to the centralized and physical CXL memory pool, thanks to VPoM architecture itself, coherence management scheme, and VPoM's performance enhancement features such as prefetch and stream processing.
- This invention series describes architecture and key operations of VPoM (Virtual Pool of Memory) systems, and special features which minimize the performance bottleneck.
- This invention series discloses
  - <<Invention A>> architecture and key operations of VPoM (Virtual Pool of Memory) systems.
  - <<Invention B>> coherence management scheme of VPoM systems.
  - <<Invention C>> performance enhancement features designed for VPoM (scheduled prefetch).
  - <<Invention D>> performance enhancement features designed for VPoM (stream processing).

New Features of Invention

- What each invention claims:
  - <<Invention A>> Overall system architecture of VPoM (Virtual Pool of Memory), which is characterized by having VPoM.GM (VPoM Global Manager, global coordinator for VPoM instance creation and other management) and CMM-VPoM (CXL Memory Module for VPoM), which has VPAG (Virtual Pool Access Gateway, which performs roles of VPoM memory transaction request router, coherence manager, remote data prefetcher, and stream processor) and VPoM.LM (Local VPoM Manager for each VPAG), (--> Figure A1, A4, A5)
    - including a method to build the VPoM instance and its address space (--> Figure A2, A3, A6)
    - method to route VPoM access requests based on this address space information (i.e., map table) (--> Table A1, A2)

- <<Invention B>> Coherence management method and system architecture for VPoM systems, to help scale-out the VPoM systems by minimizing the overhead of transaction monitoring and signaling for coherence management.
  - VPoM Coherence Manager architecture (--> Figure B1)
  - VPoM coherence management concept and table structure (--> Figure B2, Table B1)
  - VPoM coherence management related operations (--> Figure B3, B4, B5)
- <<Invention C>> Scheduled prefetch method and system architecture for VPoM systems, to minimize the latency penalty of remote memory read in VPoM systems,
  - VPoM's scheduled prefetch system architecture, which is characterized by having VPoM Remote Data Prefetch Manager (which supports prefetch triggering by prefetch-before-this-time and prefetch-after-this-transaction), Remote Memory Transaction Controller (which has dedicated prefetch queue in addition to the normal command queue, to guarantee performance isolation each other queues). (--> Figure C1)
  - Prefetch processing flows among the components (--> Figure C2)
  - VPoM prefetch related data structure (--> Table C1)
- <<Invention D>> Stream processing method and system architecture for VPoM systems, to reduce the traffic overhead in the interconnect network for memory pool (VPoM)
  - VPoM Stream Processor architecture, which is characterized by having Header Processor and Data Stream Processor, and their key operations (--> Figure D1)
  - VPoM stream processing related data structure for VPoM memory transaction request header, and stream processing result header (--> Figure D1)
  - A method to choose the best location to execute stream processing (source, destination, and interconnect switch) (--> Figure D1, D2, D3)

(P11N:A) VPoM System Architecture: Details of Invention

Architecture of VPoM (Virtual Pool of Memory) system

VPoM System Architecture

- VPoM (Virtual Pool of Memory) system consists of VPoM.GM (VPoM Global Manager) and CMM-VPoM (CXL Memory Module for VPoM) which consists of VPAG (Virtual Pool Access Gateway) and VPoM.LM (VPoM Local Manager).
  - VPAG is characterized by having the following modules: (1) Memory Transaction Packet Dispatcher (Router), (2) VPoM Address Map Table with Per-DMR Performance Attributes, (3) Remote Memory Transaction Controller, (4) Local Memory Controller, (5) VPoM Coherency Manager, (6) VPoM Remote Data Prefetch Manager, (7) VPoM Stream Processor.
  - VPoM.LM is characterized by having the following modules: (1) DMR Resource Manager (Create, Delete, Change), (2) DMR Performance Attribute Training Agent, (3) VPoM Instance Information Caching Agent.
  - VPoM.GM is characterized by having the following modules: (1) VPoM Instance Manager (Create, Delete, Change), (2) DMR Performance Attribute Training Coordinator, (3) VPoM Instance Information Server.
- Figure A1 depicts exemplary VPoM systems architecture.
- Each host has its local memory (e.g., 'Mem.1' in case of 'Host.1') equipped on local DIMM slots, and also has at least one CMM-VPoM (CXL Memory Module for VPoM).
  - CMM-VPoM has VPoM.LM (VPoM Local Manager), VPAG (Virtual Pool Access Gateway), memory (e.g., 'Mem.2' in case of 'Host.1'), and (at least) two CXL ports (one of the two CXL ports of CMM-VPoM is connected to the local host, and the other is connected to the CXL switch).
  - VPoM.LM manages the partition and donation of CMM-VPoM local memory ('Mem.2' in case of 'Host.1'). VPoM.LM creates/deletes/changes the partition of CMM-VPoM local memory based on the administrator's configuration information or predefined master default configuration (e.g., "50% of capacity (lower half of CMM-VPoM local memory's address range) will be used for local host use, 50% of capacity (higher half of CMM-VPoM local memory's address range) will be donated for VPoM"). VPoM.LM can decide one or more partitions of CMM-VPoM local memory to be donated for VPoM instance. A CMM-VPoM memory partition donated is called a Donated Memory Region (DMR). A VPoM instance can be viewed as virtual set of DMRs from participating CMM-VPOMs. VPoM.LM interacts with VPoM.GM to provide information related to the DMR, so that VPoM.GM can create/delete/change the VPoM instance based on it. VPoM.LM collects DMR Performance Attribute information during the VPoM instance training phase, as directed by VPoM.GM.
  - VPAG performs two crucial roles in VPoM systems. Firstly, VPAG performs a memory access gateway role. Based on the target memory address of the CXL memory transaction packet, VPAG forwards the CXL memory transaction packet to the CMM-VPoM local memory (e.g., 'Mem.2' in case of 'Host.1'), or forwards it to the remote memory (e.g., 'Mem.4' or 'Mem.6'). VPAG processes the CXL memory read or write requests received from VPAGs of other hosts. Secondly, VPAG performs a DMR Performance Attribute information provider role. VPAG provides host's Operating System (OS) with the expected (trained) latency and bandwidth information of each local or remote memory module which constitutes VPoM instance, so that OS can use the VPoM instance properly considering performance characterisitcs of memory module . To perform these two crucial roles, VPAG constructs and maintains (1) memory request/response routing table (MRRT), and (2) memory proximity domain table (MPDT).
- There should be VPoM.GM who coordinates all the VPoM instance related management actions. These management actions include (but not limited to) creating/deleting/changing VPoM instances, directing VPoM.LM to perform DMR Performance Attribute training, configuring CXL switch for packet forwarding based on the memory address ranges and corresponding device map, and monitoring health status of VPoM instances.

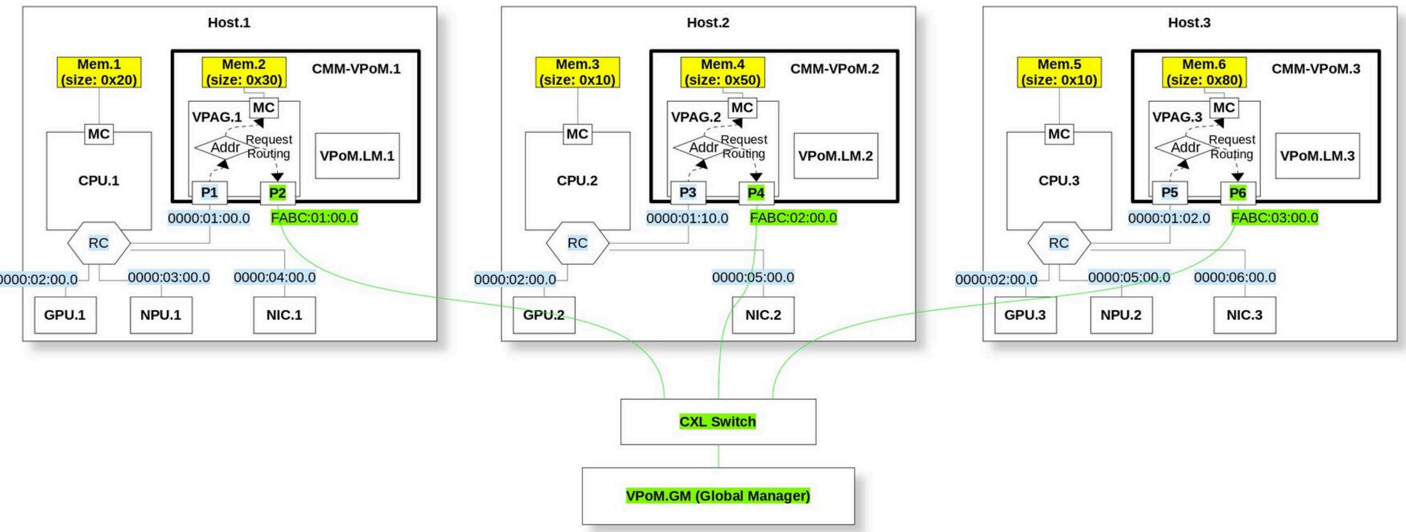


Figure A1. VPoM System Architecture

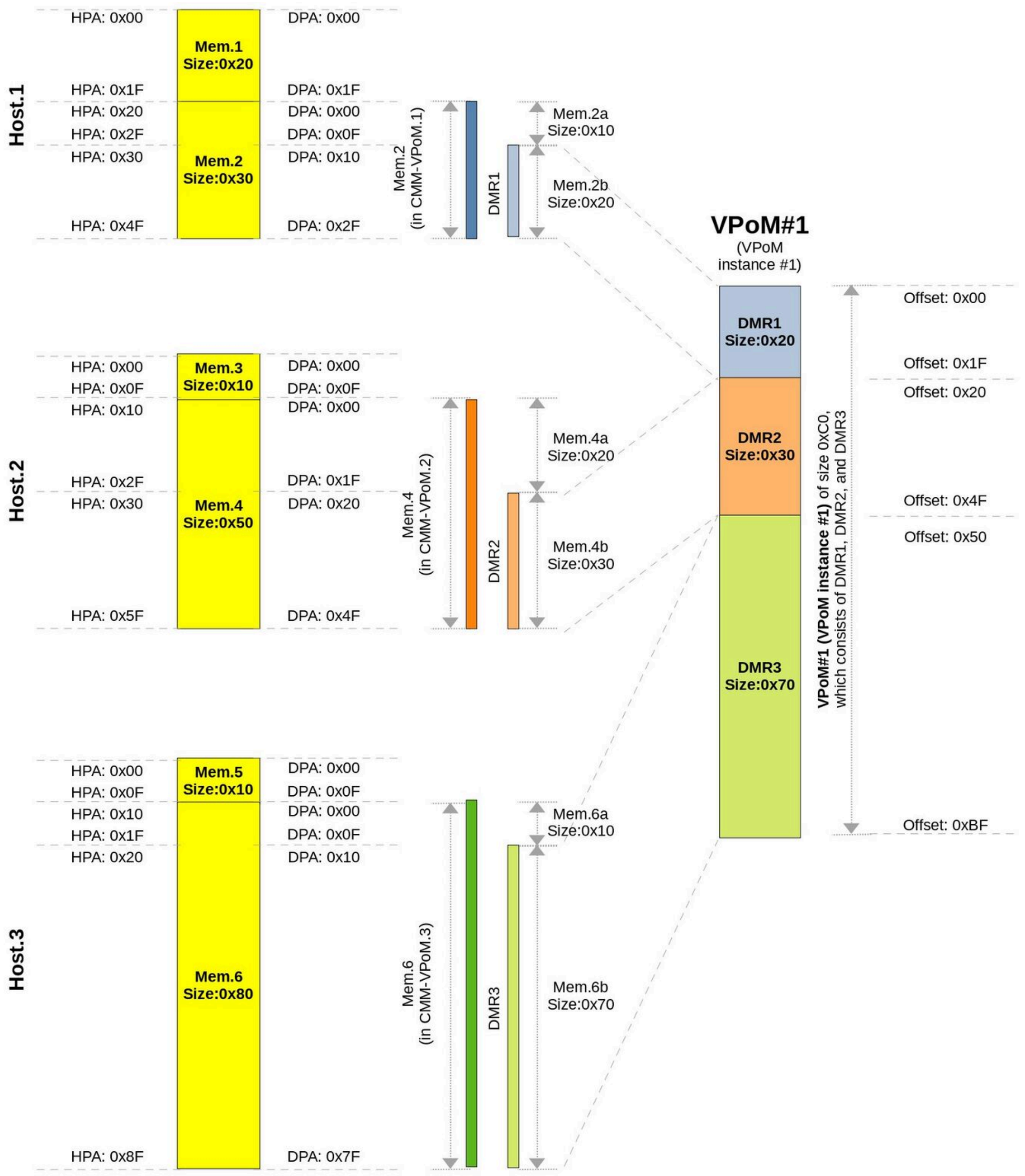


Figure A2. VPoM Instance (VPoM#1) created with three donated memory regions (DMRs)

VPoM Address Spaces

- This section describes the structure of the address space used by the VPoM system, especially how the address space that the host looks at varies before and after the VPoM instance is created.

Description of how each module works goes here.

Table A1. VPoM Address Spaces in terms of HPA (Host Physical Address), before and after creating a VPoM Instance (VPoM#1 depicted in Figure A2)

Case	Host.1's HPA View	Host.2's HPA View	Host.3's HPA View
Before creating a VPoM instance (VPoM#1)	Memory size total: 0x50 <Address Range> : <Target Memory> 0x0000 ~ 0x001F : Mem.1 0x0020 ~ 0x004F : Mem.2	Memory size total: 0x60 <Address Range> : <Target Memory> 0x0000 ~ 0x000F : Mem.3 0x0010 ~ 0x005F : Mem.4	Memory size total: 0x90 <Address Range> : <Target Memory> 0x0000 ~ 0x000F : Mem.5 0x0010 ~ 0x008F : Mem.6
After creating a VPoM instance (VPoM#1)	Memory size total: 0xF0 <Address Range> : <Target Memory> 0x0000 ~ 0x001F : Mem.1 0x0020 ~ 0x002F : Mem.2a 0x0030 ~ 0x0FFF: Unused 0x1000 ~ 0x101F : VPoM#1.DMR1 (Mem.2b) 0x1020 ~ 0x104F : VPoM#1.DMR2 (Mem.4b) 0x1050 ~ 0x10BF : VPoM#1.DMR3 (Mem.6b)	Memory size total: 0xF0 <Address Range> : <Target Memory> 0x0000 ~ 0x000F : Mem.3 0x0010 ~ 0x002F : Mem.4a 0x0030 ~ 0x0FFF: Unused 0x1000 ~ 0x101F : VPoM#1.DMR1 (Mem.2b) 0x1020 ~ 0x104F : VPoM#1.DMR2 (Mem.4b) 0x1050 ~ 0x10BF : VPoM#1.DMR3 (Mem.6b)	Memory size total: 0xE0 <Address Range> : <Target Memory> 0x0000 ~ 0x000F : Mem.5 0x0010 ~ 0x001F : Mem.6a 0x0020 ~ 0x0FFF: Unused 0x1000 ~ 0x101F : VPoM#1.DMR1 (Mem.2b) 0x1020 ~ 0x104F : VPoM#1.DMR2 (Mem.4b) 0x1050 ~ 0x10BF : VPoM#1.DMR3 (Mem.6b)

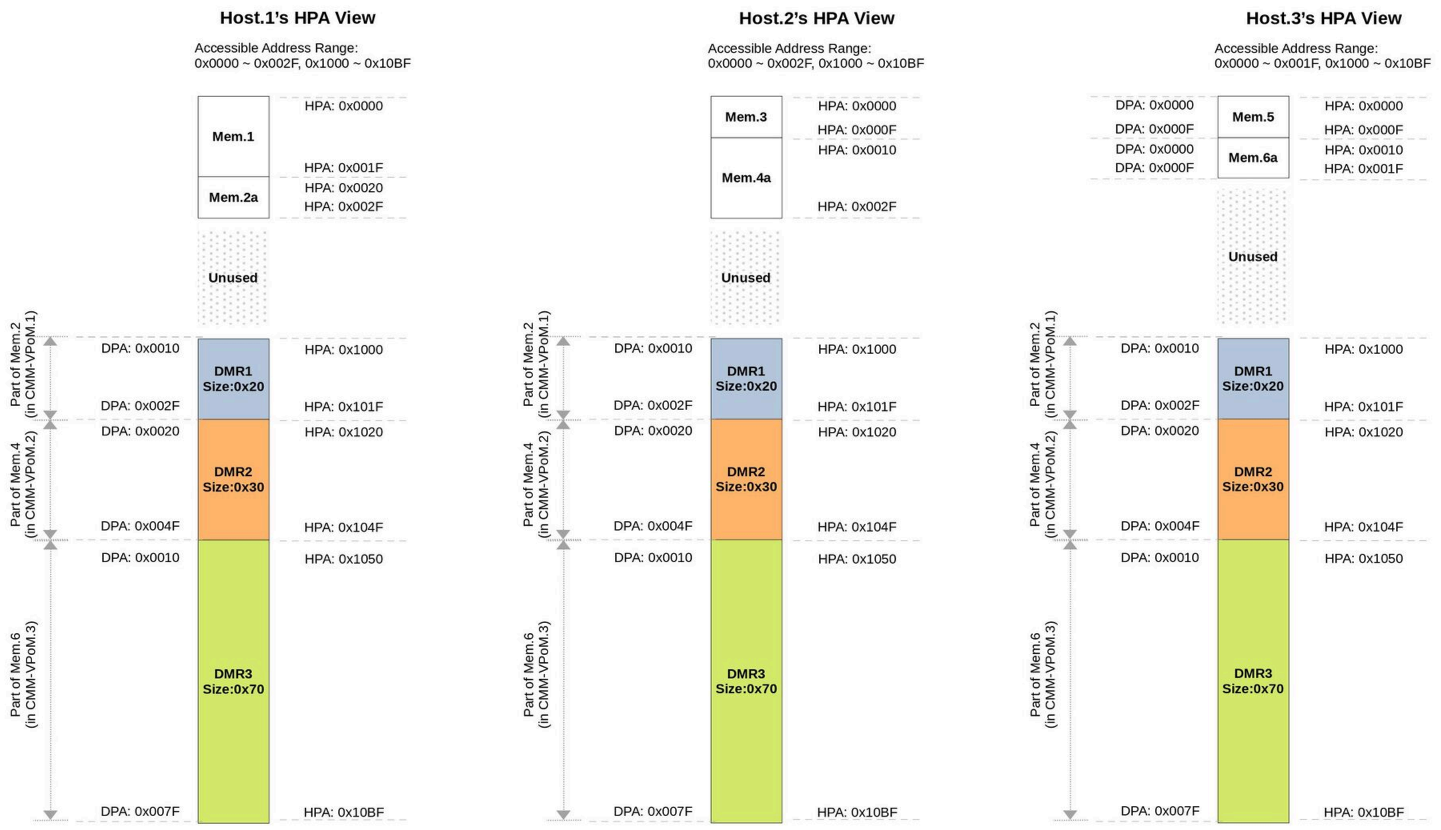


Figure A3. HPA that each host CPU sees after creation and attachment of VPoM instance

- In the example of Figure A1., Host.1 has two memory modules, the first one is host CPU memory marked as Mem.1 whose size is 0x20, and the second one is CMM-VPoM memory marked as Mem.2 whose size is 0x30. So the Host.1 has local HPA (Host Physical Address) ranges from 0x00~0x4F (Mem.1 0x00~0x1F, Mem.2 0x20~0x4F).
- Depending on the system, the CXL memory address may start immediately after the end of the host CPU memory in the DIMM, or may start from a specified address value. In the former case, a continuous address space encompassing the host CPU memory and the CXL memory is created, and in the latter case, a discontinuous

address space in which an unused address range exists between the host CPU memory and the CXL memory address is created. In this example, the former situation is assumed.

- VPoM.LM in CMM-VPoM.1 manages partition of Mem.2. VPoM.LM creates two memory partitions of size 0x10 (Mem.2a) and size 0x20 (Mem.2b). VPoM.LM decides to donate the second memory partition Mem.2b as DMR (Donated Memory Region) for VPoM instance, so that partition Mem.2b can be shared to the other hosts. Mem.2b is marked as DMR1.
- Likewise, VPoM.LM of Host.2 and that of Host.3 do the same thing. Therefore, for Host.2, Mem.4b which is the second partition of Mem.4 (size 0x50) is donated to the VPoM instance as DMR2. And for Host.3, Mem.6b which is the second partition of Mem.6 (size 0x80) is donated to the VPoM instance as DMR3.
- VPoM.GM creates a VPoM instance of size 0xC0 called VPoM#1 using donated memory regions such as DMR1 (Mem.2b, size: 0x20), DMR2 (Mem.4b, size: 0x30), and DMR3 (Mem.6b, size: 0x70). Then VPoM.GM provides the information needed to use the VPoM instance to all VPoM.LMs who participated in creating the VPoM#1.
- VPoM.LM in each host provides updated CXL memory information to the host, so that host CPU can use VPoM instance (VPoM#1 in this example).
- The differences in address space that the host CPU sees before and after creating a VPoM instance (VPoM#1) are described in Table 1.
- HPA Range for VPoM is reserved address range for VPoM instances. It means there may exist one or multiple number of VPoM instances in use, then start addresses of all the VPoM instances should be equal or larger than the lower bound of HPA Range for VPoM, and end addresses of all the VPoM instances should be equal or smaller than the upper bound of HPA Range for VPoM. It is strongly recommended that each host does not use this HPA Range for VPoM for other uses than VPoM. This makes the management of VPoM system easier. Every new VPoM instance will have start HPA and end HPA allocated within this HPA Range for VPoM. VPoM.LM and VPoM.GM collaborate to decide the proper start HPA and end HPA of newly created VPoM instance when they create new VPoM instance, to avoid any address conflict.
- If there is no predefined HPA Range for VPoM definition in each host, then VPoM.GM (VPoM Global Manager) will provide HPA Range for VPoM information to VPoM.LMs in all hosts, as a default configuration.
- In the example illustrated in Figure 3, 0x1000 is the lower bound of HPA Range for VPoM. Thus every host uses 0x1000 as a HPA Base to access VPoM (In this example, we assume that the HPA Base for VPoM 0x1000 is large enough to prevent address conflicts between any local memory devices (including DIMMs and CXL memory devices) and VPoM instances). If there is a need to create additional VPoM instance to use, then each VPoM.LM checks the available HPA range for new VPoM instance, and lets VPoM.GM knows this information. VPoM.GM collects this information that each VPoM.LM shared. After successfully creating the new VPoM, VPoM.GM determines the proper address as a start address
- In our example case, there may be unused memory address range which host CPU can not access. For example, CPU.1 can not access HPA 0x0030~0x0FFF, CPU.2 can not access HPA 0x0030~0x0FFF, and CPU.3 can not access HPA 0x0020~0x0FFF.
- Due to the resource virtualization provided by VPAG, there are resources where entities in the opposite resource area cannot see. For example, in Figure 1, host CPUs such as CPU.1, CPU.2, and CPU.3 can not see the resources marked with light green color, such as P2, P4, P6, VPoM.GM, and CXL Switch. What they can see are resources marked with light blue color. In the case of Host.1, CPU.1 can see CXL port P1 (0000:01:00.0), GPU.1 (0000:02:00.0), NPU.1 (0000:03:00.0), and NIC.1 (0000:04:00.0). In the case of Host.2, CPU.2 can see CXL port P3 (0000:01:10.0), GPU.2 (0000:02:00.0), and NIC.1 (0000:05:00.0). In the case of Host.3, CPU.3 can see CXL port P5 (0000:01:02.0), GPU.3 (0000:02:00.0), NPU.2 (0000:05:00.0), and NIC.3 (0000:06:00.0). This resource isolation makes it easier to manage failures of the DMR (Donated Memory Region) that make up the VPoM instance. However, VPAG which has two CXL ports (one belongs to the light blue resource area, the other one belongs to the light green resource area) can see both resource areas to perform gateway role.

VPAG

- VPAG is characterized by having the following modules: (1) Memory Transaction Packet Dispatcher (Router), (2) VPoM Address Map Table with Per-DMR Performance Attributes, (3) Remote Memory Transaction Controller, (4) Local Memory Controller, (5) VPoM Coherence Manager, (6) VPoM Remote Data Prefetch Manager, (7) VPoM Stream Processor.
- This invention covers (1) Memory Transaction Packet Dispatcher (Router), (2) VPoM Address Map Table with Per-DMR Performance Attributes, (3) Remote Memory Transaction Controller, (4) Local Memory Controller.
- Other inventions will cover (5) VPoM Coherence Manager, (6) VPoM Remote Data Prefetch Manager, (7) VPoM Stream Processor.

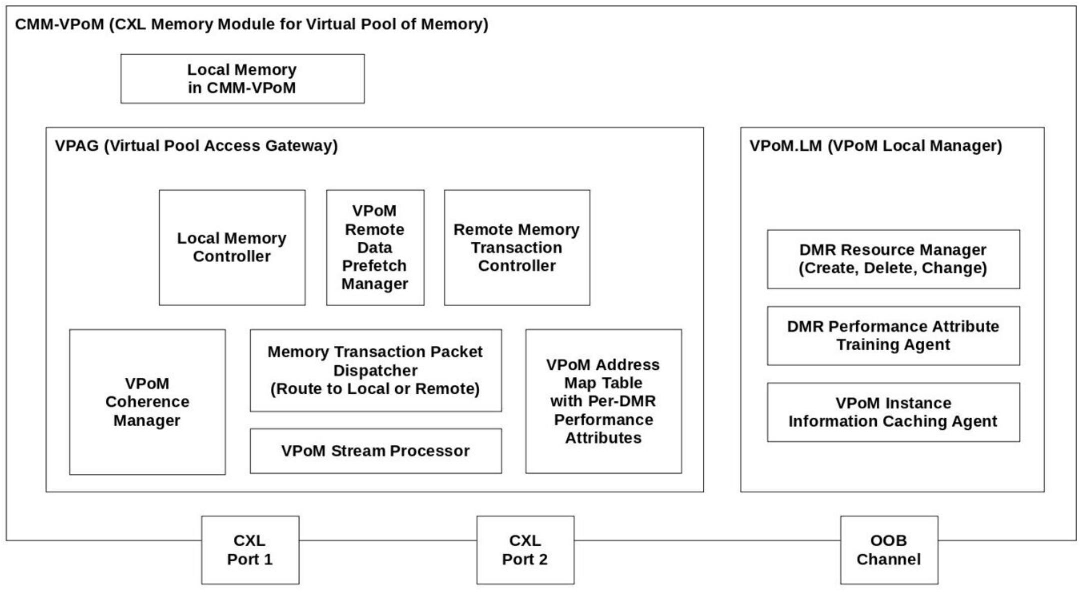

CMM-VPoM (CXL Memory Module for Virtual Pool of Memory)
Local Memory in CMM-VPoM
VPAG (Virtual Pool Access Gateway)
Local Memory Controller
VPoM Remote Data Prefetch Manager
Remote Memory Transaction Controller
VPoM Coherence Manager
Memory Transaction Packet Dispatcher (Route to Local or Remote)
VPoM Stream Processor
VPoM Address Map Table with Per-DMR Performance Attributes
VPoM.LM (VPoM Local Manager)
DMR Resource Manager (Create, Delete, Change)
DMR Performance Attribute Training Agent
VPoM Instance Information Caching Agent
CXL Port 1
CXL Port 2
OOB Channel

Figure A4. CMM-VPoM internal architecture including VPAG and VPoM.LM

VPAG: Memory Transaction Packet Router with VPoM Address Map Table with Per-DMR Performance Attributes

- VPAG has VPoM Address Map Table which consists of a couple of columns such as Address Range (e.g., 0x1000 ~ 0x101F), Target Interface (e.g., to be forwarded to the local memory controller, or to be forwarded to the egress CXL port), DMR ID (Donated Memory Region ID), and DMR Performance Attribute which provides latency information (e.g., minimum, average, standard deviation, maximum, and some selected tail latencies), and bandwidth information (e.g., minimum, average, maximum) for selected memory IO sizes such as 64B, 4KiB, 24KiB, and so on.
- VPAG can route memory transaction packets to the proper destination using Address Range and Target Interface information, and VPAG can determine the optimal flow rate considering the latency and bandwidth information given by DMR Performance Attribute in this table. Details on how to determine the optimal flow rate will be covered in other inventions.
- DMR Performance Attribute information can be collected by performing DMR Performance Parameter Training during the phase of VPoM creation.
- VPAG provides an interface for host CPU to access this table so that the operating system running on the host CPU can use the DMR Performance Attribute information for optimal job scheduling.
- VPAG constructs VPoM Address Map Table with Per-DMR Performance Attributes, based on the information given by VPoM.LM (especially from VPoM Instance Information Caching Agent)

Table A2. Exemplary VPoM Address Map Table with Per-DMR Performance Attributes (VPAG.1 case)

Address Range	Target Interface	DMR ID	DMR Performance Attributes				
			Memory IO Size: 64B		Memory IO Size: 4KiB		
			latency	bandwidth	latency	bandwidth	
0x1000~ 0x101F	CMM-VPoM.1's local memory (Mem.2b)	VPoM#1.DMR1	min: __ns avg: __ns stdev: __ns max: __ns 99%: __ns 99.9%: __ns	min: __GB/s avg: __GB/s max: __GB/s	min: __ns avg: __ns stdev: __ns max: __ns 99%: __ns 99.9%: __ns	min: __GB/s avg: __GB/s max: __GB/s	min: __n avg: __ stdev: _ max: __ 99%: _ 99.9%:
0x1020~ 0x104F	CXL Port P2 (Mem.4b)	VPoM#1.DMR2	min: __ns avg: __ns stdev: __ns max: __ns 99%: __ns 99.9%: __ns	min: __GB/s avg: __GB/s max: __GB/s	min: __ns avg: __ns stdev: __ns max: __ns 99%: __ns 99.9%: __ns	min: __GB/s avg: __GB/s max: __GB/s	min: __n avg: __ stdev: _ max: __ 99%: _ 99.9%:
0x1050~ 0x10BF	CXL Port P2 (Mem.6b)	VPoM#1.DMR3	min: __ns avg: __ns stdev: __ns max: __ns 99%: __ns 99.9%: __ns	min: __GB/s avg: __GB/s max: __GB/s	min: __ns avg: __ns stdev: __ns max: __ns 99%: __ns 99.9%: __ns	min: __GB/s avg: __GB/s max: __GB/s	min: __n avg: __ stdev: _ max: __ 99%: _ 99.9%:

VPAG: Remote Memory Transaction Controller

- VPAG has special type of memory controller to manage remote memory transactions. For efficient processing of memory transactions, VPAG may handle the memory transactions separately based on the transaction types such as bulk memory transaction, atomic memory transaction, and normal memory transaction. Memory transaction type may be specified by the host explicitly, or may be classified automatically by VPAG.
- Memory access requests can be classified as bulk memory transaction type when the they are related with accessing large area of memory spaces at a time. It can be one memory request to access the large amount of consecutive memory region, or they can be a couple of memory requests to access multiple memory regions. The criteria to classify as bulk memory transaction can be specified by the on-site human system administrator or provided as a default policy of VPoM.GM. Or, there may be explicit tag which denotes the request can be handled as a bulk transaction. In case of bulk memory transaction, VPAG's Bulk Memory Transaction Manager can split the request into a couple of sub requests so that it can get benefit of parallelism. Per-DMR performance parameter information also can be considered to determine the proper size of stripe for parallelism.
- Memory access requests can be classified as atomic memory transaction type when there is a memory barrier (or fence) to enforce an ordering constraint on memory operations. Or, there may be explicit tag which denotes the group of requests should be handled as an atomic transaction. In case of atomic memory transaction, VPAG's Atomic Memory Transaction Manager may create separate queue for each atomic transaction if the address space range does not overlap among them (atomic transaction request), to avoid head-of-lock issue.
- Memory access requests which were not classified as bulk memory transaction type or atomic memory transaction type will be handled as a normal memory transaction type, and these memory requests will be handled separately from the bulk memory transaction type requests and the atomic memory transaction type requests. Normal memory transaction manager may create the submission queue - completion queue pairs as many as the number of target DMRs, to avoid any potential head-of-lock issue, if the remaining space for queue permits.

VPAG: Local Memory Controller

- This is memory controller for the CMM-VPoM local memory such as DDR or LPDDR.

VPoM.LM

VPoM.LM is characterized by having the following modules: (1) DMR Resource Manager (Create, Delete, Change), (2) DMR Performance Attribute Training Agent, (3) VPoM Instance Information Caching Agent. Description of how each module works goes here.

VPoM.LM: DMR Resource Manager (Create, Delete, Change)

- DMR Resource Manager creates/deletes/changes the partition of the local memory (e.g., in the case of CMM-VPoM.1, Mem.2 can be partitioned into Mem.2a and Mem.2b). DMR Resource Manager manages donation of the local memory partition to create VPoM instance, also manages withdrawal of the memory donated. If VPoM.LM decides to donate some partition of the memory (e.g., Mem.2b), then this partition will be registered to the DMR (Donated Memory Region) list of the DMR Resource Manager, and then VPoM.LM inform VPoM.GM about this. When there are any changes regarding the DMR, DMR Resource Manager notifies this to VPoM.GM so that VPoM.GM can re-configure the related VPoM instances.

VPoM.LM: DMR Performance Attribute Training Agent

- DMR Performance Attribute Training Agent performs the test to evaluate the latency and throughput characteristics of remote memory (remote DMR of the target VPoM instance). This maintains Test Case DataBase (TCDB) for DMR Performance Attribute training. The test cases are defined in prescribed data format. DMR Performance Attribute Training Agent fetches and parses the test cases, and executes them in the order specified. DMR Performance Attribute Training Agent converts the action specified in the test case to the CXL memory requests. DMR Performance Attribute Training Agent generates the memory access requests, and handles the responses of them, to evaluate the performance characteristics and also to evaluate the error rates and error modes. These remote memory access requests and responses are transmitted over CXL interconnect.
- Each DMR Performance Attribute Training Agent's operation sequence is coordinated by VPoM.GM.

VPoM.LM: VPoM Instance Information Caching Agent

- VPoM Instance Information Caching Agent manages the copy of information maintained by VPoM.GM (VPoM Global Manager) for quick access by VPAG, such as current list of VPoM instances, memory transaction request routing table, per-DMR performance attributes, and so on. Thanks to this VPoM Information Cache, there is no need for VPAG to travel across the network to the VPoM.GM whenever VPAG needs this information. If there are any changes in local DMR configuration, or there are any faults ocurred in the local DMR, then VPoM.LM's VPoM Information Caching Agent informs the VPoM.GM about these events.

VPoM.GM

VPoM.GM is characterized by having the following modules: (1) VPoM Instance Manager (Create, Delete, Change), (2) DMR Performance Attribute Training Coordinator, (3) VPoM Instance Information Server. Before going into the description of how each module works, basic assumptions are described. Figure 5 depicts the architecture of VPoM.GM.

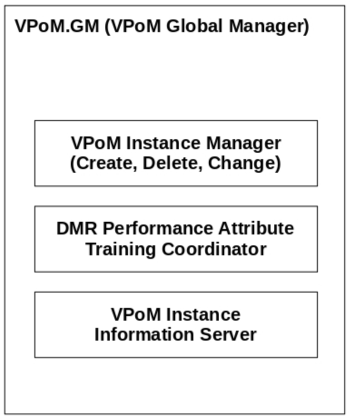

VPoM.GM (VPoM Global Manager)
VPoM Instance Manager (Create, Delete, Change)
DMR Performance Attribute Training Coordinator
VPoM Instance Information Server

Figure A5. VPoM.GM (VPoM Global Manager)

VPoM.GM: VPoM Instance Manager (Create, Delete, Change)

- VPoM.GM's VPoM Instance Manager cooperates with VPoM.LMs to create/delete/change VPoM instances.
- Figure B3 illustrates a procedure to create a VPoM instance, done by VPoM.GM and VPoM.LMs. The followings are brief description about it.
- VPoM.GM can delete the VPoM instance or can change VPoM instance when there are no outstanding memory transactions on that VPoM instance. When this happens VPoM.GM notifies this event to the related VPoM.LM. Then VPoM.LM performs the corresponding actions guided by VPoM.GM.

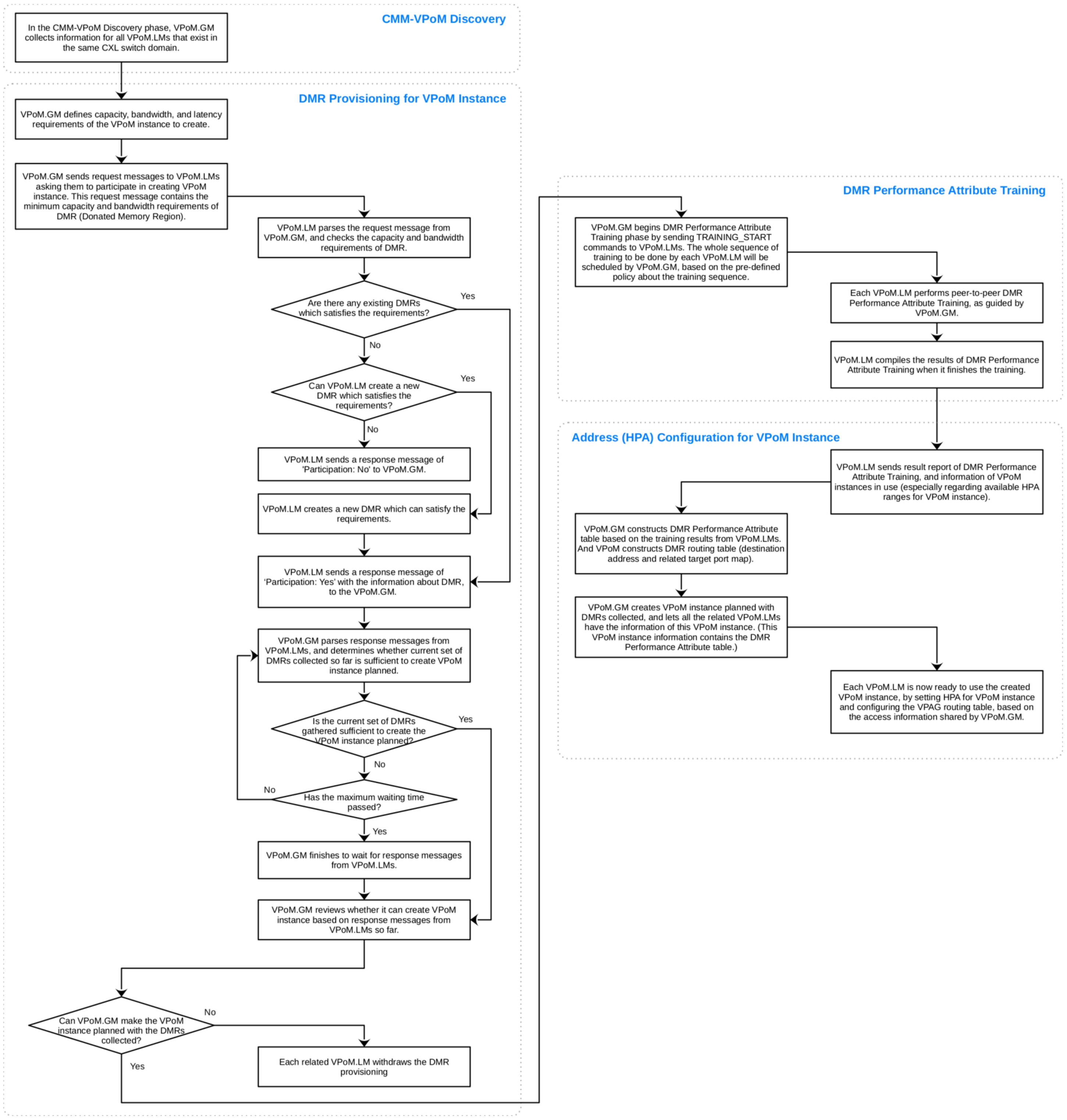

CMM-VPoM Discovery
In the CMM-VPoM Discovery phase, VPoM.GM collects information for all VPoM.LMs that exist in the same CXL switch domain.
DMR Provisioning for VPoM Instance
VPoM.GM defines capacity, bandwidth, and latency requirements of the VPoM instance to create.
VPoM.GM sends request messages to VPoM.LMs asking them to participate in creating VPoM instance. This request message contains the minimum capacity and bandwidth requirements of DMR (Donated Memory Region).
VPoM.LM parses the request message from VPoM.GM, and checks the capacity and bandwidth requirements of DMR.
Are there any existing DMRs which satisfies the requirements?
Yes
No
Can VPoM.LM create a new DMR which satisfies the requirements?
Yes
No
VPoM.LM sends a response message of 'Participation: No' to VPoM.GM.
VPoM.LM creates a new DMR which can satisfy the requirements.
VPoM.LM sends a response message of 'Participation: Yes' with the information about DMR, to the VPoM.GM.
VPoM.GM parses response messages from VPoM.LMs, and determines whether current set of DMRs collected so far is sufficient to create VPoM instance planned.
Is the current set of DMRs gathered sufficient to create the VPoM instance planned?
Yes
No
No
Has the maximum waiting time passed?
Yes
VPoM.GM finishes to wait for response messages from VPoM.LMs.
VPoM.GM reviews whether it can create VPoM instance based on response messages from VPoM.LMs so far.
Can VPoM.GM make the VPoM instance planned with the DMRs collected?
No
Yes
Each related VPoM.LM withdraws the DMR provisioning
DMR Performance Attribute Training
VPoM.GM begins DMR Performance Attribute Training phase by sending TRAINING_START commands to VPoM.LMs. The whole sequence of training to be done by each VPoM.LM will be scheduled by VPoM.GM, based on the pre-defined policy about the training sequence.
Each VPoM.LM performs peer-to-peer DMR Performance Attribute Training, as guided by VPoM.GM.
VPoM.LM compiles the results of DMR Performance Attribute Training when it finishes the training.
Address (HPA) Configuration for VPoM Instance
VPoM.LM sends result report of DMR Performance Attribute Training, and information of VPoM instances in use (especially regarding available HPA ranges for VPoM instance).
VPoM.GM constructs DMR Performance Attribute table based on the training results from VPoM.LMs. And VPoM constructs DMR routing table (destination address and related target port map).
VPoM.GM creates VPoM instance planned with DMRs collected, and lets all the related VPoM.LMs have the information of this VPoM instance. (This VPoM instance information contains the DMR Performance Attribute table.)
Each VPoM.LM is now ready to use the created VPoM instance, by setting HPA for VPoM instance and configuring the VPAG routing table, based on the access information shared by VPoM.GM.

Figure A6. Procedure to create a VPoM instance

1. In the CMM-VPoM Discovery phase, VPoM.GM collects information for all VPoM.LMs that exist in the same CXL switch domain.
2. VPoM.GM defines capacity, bandwidth, and latency requirements of the VPoM instance to create.
3. VPoM.GM sends request messages to VPoM.LMs asking them to participate in creating VPoM instance. This request message contains the minimum capacity and bandwidth requirements of DMR (Donated Memory Region).
4. VPoM.LM parses the request message from VPoM.GM, and checks the capacity and bandwidth requirements of DMR. If there is any pre-configured DMR which satisfies the requirements, then VPoM.LM selects that DMR. If there is no such DMR but if the current memory resource status permits, then VPoM.LM creates a new DMR which can satisfy the requirements. VPoM.LM sends a response message of 'Participation: Yes' with the information about DMR, to the VPoM.GM. If it is not possible to participate with the proper DMR, then VPoM.LM sends a response message of 'Participation: No' to VPoM.GM.
5. For each response message from VPoM.LM, VPoM.GM parses it, and determines whether current set of DMRs collected so far is sufficient to create VPoM instance planned. If the current set of DMRs gathered sufficient to create the VPoM instance planned, or if the maximum waiting time has passed, then VPoM.GM finishes to wait for response messages from VPoM.LMs.
6. VPoM.GM reviews whether it can create VPoM instance based on response messages from VPoM.LMs so far. If VPoM.GM can make the VPoM instance planned with the DMRs collected, then VPoM.GM begins DMR Performance Attribute Training, based on the pre-defined policy about training sequence.

7. Each VPoM.LM performs peer-to-peer DMR Performance Attribute Training, as guided by VPoM.GM. When the all the training sequences are completed, then each VPoM.LM reports the DMR Performance Attribute Training results to VPoM.GM.
8. VPoM.GM constructs DMR Performance Attribute table based on the training results from VPoM.LMs. VPoM.GM creates VPoM instance planned with DMRs collected, and lets all the related VPoM.LMs have the information of this VPoM instance. (This VPoM instance information contains the DMR Performance Attribute table.)
9. Each VPoM.LM is now ready to use the created VPoM instance, based on the access information shared by VPoM.GM.

VPoM.GM: DMR Performance Attribute Training Coordinator

- VPoM.GM coordinates all DMR Performance Attribute Training Agent operations based on the DMR Performance Attribute Training Procedure, which can be defined by on-site system administrator, or which can be provided as a default configuration. Purpose of this coordination is to avoid any unwanted interference caused by the unplanned training traffic.

VPoM.GM: VPoM Instance Information Server

- VPoM Instance Information server communicates with VPoM.LM to provide VPoM instance access information to each host.

(P11N:B) Coherence Management for VPoM systems: Details of Invention

Summary of Invention

- VPoM (Virtual Pool of Memory) is a shared memory system virtually constructed with a set of donated memory regions from individual CXL Memory Modules (CMMs). VPoM provides more consistent memory bandwidth and lower latency compared to the centralized and physical CXL memory pool, thanks to VPoM architecture itself, coherence management scheme, and VPoM's performance enhancement features such as prefetch and stream processing.
- This invention series describes architecture and key operations of VPoM (Virtual Pool of Memory) systems, and special features which minimize the performance bottleneck.
- This invention discloses
  - ~~architecture and key operations of VPoM (Virtual Pool of Memory) systems.~~
  - coherence management scheme of VPoM systems.
  - ~~performance enhancement features designed for VPoM (scheduled prefetch).~~
  - ~~performance enhancement features designed for VPoM (stream processing).~~

VPAG: VPoM Coherence Manager

- VPAG monitors the changes of specified memory area, and informs relevant host CPUs so that they can invalidate the cache of the corresponding range of memory.
- Figure B1 depicts the architecture of VPAG's **VPoM Coherence Manager**. VPAG always knows if there are any data changes on the DMR (local DMR that VPAG manages), thanks to the **Memory Transaction Monitor** in the **Memory Transaction Packet Dispatcher** in VPAG. When there are data changes in the memory chunks registered in this **VPoM Coherence Management Table**, then VPAG check the current time and 'Coherence::Data-Deadline' information, so if the current time does not exceed the 'Coherence::Data-Deadline' time, then VPAG Coherence Manager's **Data Change Notifier** sends data change notification message(s) for the corresponding memory chunk(s) to the corresponding host(s), so that the host can invalidate the cache for that data retrieved. But if the current time already has passed 'Coherence::Data-Deadline' time, then VPAG deletes the corresponding record in the table, and does not send any data change notification. **Table Maintenance Loop** in the **VPoM Coherence Manager** performs this actions based on the information provided by **Per-Memory-Chunk Runtime Status Bitmap** in the **Memory Transaction Packet Dispatcher**.

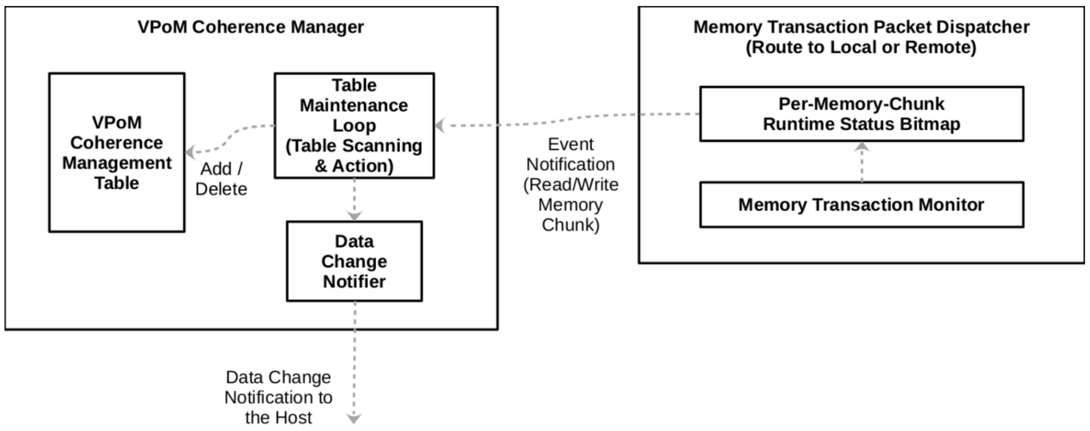
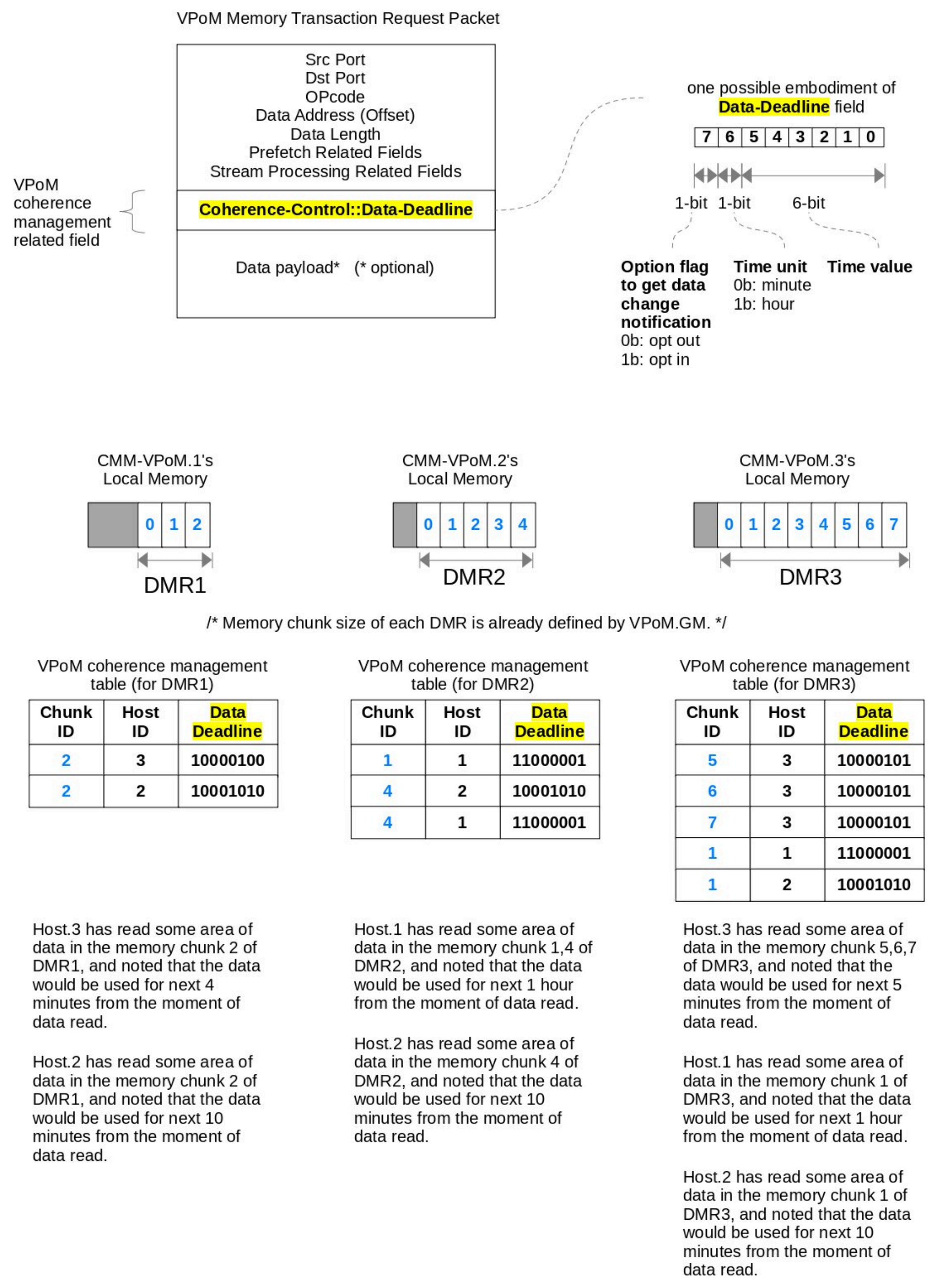

VPoM Coherence Manager
VPoM Coherence Management Table
Add / Delete
Table Maintenance Loop (Table Scanning & Action)
Data Change Notifier
Data Change Notification to the Host
Event Notification (Read/Write Memory Chunk)
Memory Transaction Packet Dispatcher (Route to Local or Remote)
Per-Memory-Chunk Runtime Status Bitmap
Memory Transaction Monitor

Figure B1. VPoM Coherence Manager Architecture and Related Operations

- Figure B2 illustrates one possible embodiment of VPoM coherence management table for DMR1, DMR2, and DMR3 cases.
  - For DMR1, Host.3 has read some area of data in the memory chunk 2 of DMR1, and noted that the data would be used for next 4 minutes from the moment of data read. Host.2 has read some area of data in the memory chunk 2 of DMR1, and noted that the data would be used for next 10 minutes from the moment of data read.
  - For DMR2, Host.1 has read some area of data in the memory chunk 1,4 of DMR2, and noted that the data would be used for next 1 hour from the moment of data read. Host.2 has read some area of data in the memory chunk 4 of DMR2, and noted that the data would be used for next 10 minutes from the moment of data read.
  - For DMR3, Host.3 has read some area of data in the memory chunk 5,6,7 of DMR3, and noted that the data would be used for next 5 minutes from the moment of data read. Host.1 has read some area of data in the memory chunk 1 of DMR3, and noted that the data would be used for next 1 hour from the moment of data read. Host.2 has read some area of data in the memory chunk 1 of DMR3, and noted that the data would be used for next 10 minutes from the moment of data read.


VPoM Memory Transaction Request Packet
Src Port
Dst Port
OPcode
Data Address (Offset)
Data Length
Prefetch Related Fields
Stream Processing Related Fields
Coherence-Control::Data-Deadline
Data payload* (* optional)
VPoM coherence management related field
one possible embodiment of Data-Deadline field
7 6 5 4 3 2 1 0
1-bit 1-bit 6-bit
Option flag to get data change notification
0b: opt out
1b: opt in
Time unit
0b: minute
1b: hour
Time value
CMM-VPoM.1's Local Memory
0 1 2
DMR1
CMM-VPoM.2's Local Memory
0 1 2 3 4
DMR2
CMM-VPoM.3's Local Memory
0 1 2 3 4 5 6 7
DMR3
/* Memory chunk size of each DMR is already defined by VPoM.GM. */

VPoM coherence management table (for DMR1)

Chunk ID	Host ID	Data Deadline
2	3	10000100
2	2	10001010

Host.3 has read some area of data in the memory chunk 2 of DMR1, and noted that the data would be used for next 4 minutes from the moment of data read.

Host.2 has read some area of data in the memory chunk 2 of DMR1, and noted that the data would be used for next 10 minutes from the moment of data read.

VPoM coherence management table (for DMR2)

Chunk ID	Host ID	Data Deadline
1	1	11000001
4	2	10001010
4	1	11000001

Host.1 has read some area of data in the memory chunk 1,4 of DMR2, and noted that the data would be used for next 1 hour from the moment of data read.

Host.2 has read some area of data in the memory chunk 4 of DMR2, and noted that the data would be used for next 10 minutes from the moment of data read.

VPoM coherence management table (for DMR3)

Chunk ID	Host ID	Data Deadline
5	3	10000101
6	3	10000101
7	3	10000101
1	1	11000001
1	2	10001010

Host.3 has read some area of data in the memory chunk 5,6,7 of DMR3, and noted that the data would be used for next 5 minutes from the moment of data read.

Host.1 has read some area of data in the memory chunk 1 of DMR3, and noted that the data would be used for next 1 hour from the moment of data read.

Host.2 has read some area of data in the memory chunk 1 of DMR3, and noted that the data would be used for next 10 minutes from the moment of data read.

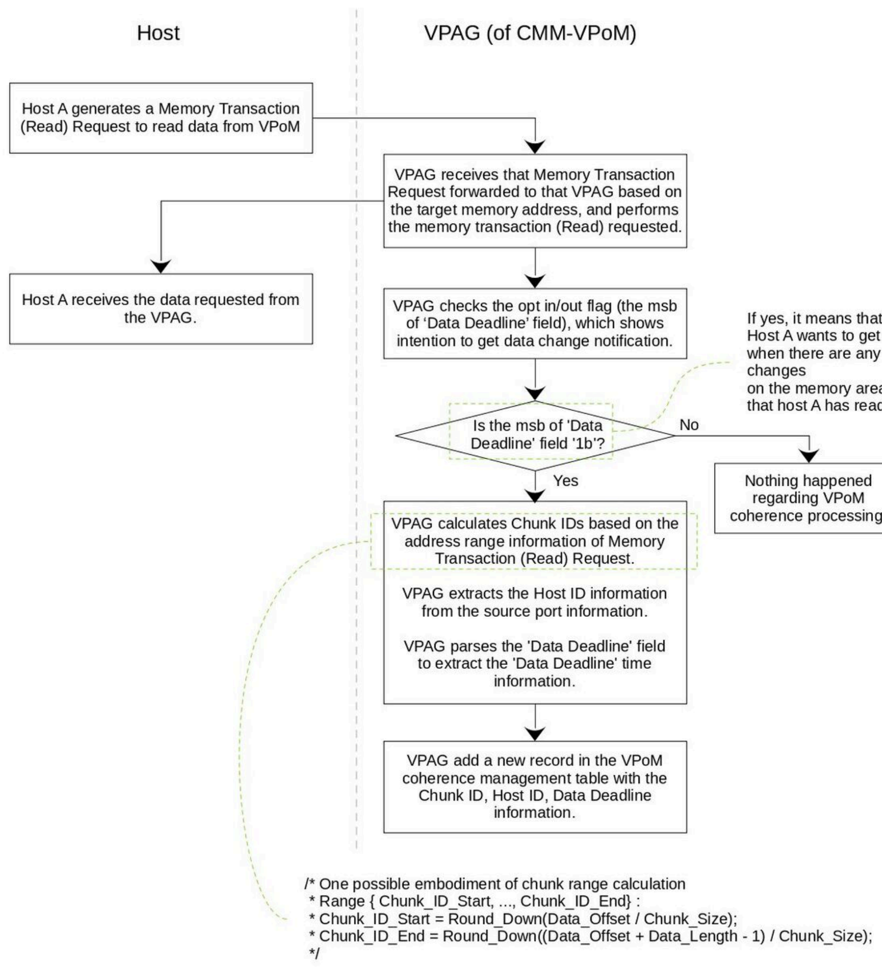
Figure B2. VPoM Coherence Management Table.

- Table B1 describes details of Coherence::Data-Deadline field.

Table B1. VPoM Coherence Control related fields in the CXL Memory Transaction Request Packet.

field name	field value type
Coherence::Data-Deadline	▪ if msb of this field value is 0b, then it means opt out of the data change notification service. ▪ if msb of this field value is 1b, then it means opt in of the data change notification service. ▪ if 2nd msb of this field value is 0b, then time unit is minute. ▪ if 2nd msb of this field value is 1b, then time unit is hour. ▪ the remaining value of this field (from 3rd msb to the lsb) is 'Data-Deadline' time value.

- Figure B3 and B4 depict VPoM Coherence Management Operations when there are VPoM Memory Reads, Writes. Figure B5 depict housekeeping loop operations which periodically scan the VPoM coherence management table to find records of which 'Data-Deadline' already passed. When there are any expired records, then VPAG deletes them.


Host
VPAG (of CMM-VPoM)
Host A generates a Memory Transaction (Read) Request to read data from VPoM
VPAG receives that Memory Transaction Request forwarded to that VPAG based on the target memory address, and performs the memory transaction (Read) requested.
Host A receives the data requested from the VPAG.
VPAG checks the opt in/out flag (the msb of 'Data Deadline' field), which shows intention to get data change notification.
If yes, it means that Host A wants to get notified when there are any changes on the memory area that host A has read
Is the msb of 'Data Deadline' field '1b'?
No
Yes
Nothing happened regarding VPoM coherence processing.
VPAG calculates Chunk IDs based on the address range information of Memory Transaction (Read) Request.
VPAG extracts the Host ID information from the source port information.
VPAG parses the 'Data Deadline' field to extract the 'Data Deadline' time information.
VPAG add a new record in the VPoM coherence management table with the Chunk ID, Host ID, Data Deadline information.

```
/* One possible embodiment of chunk range calculation
 * Range { Chunk_ID_Start, ..., Chunk_ID_End} :
 * Chunk_ID_Start = Round_Down(Data_Offset / Chunk_Size);
 * Chunk_ID_End = Round_Down((Data_Offset + Data_Length - 1) / Chunk_Size);
 */
```

Figure B3. VPoM Coherence Management Operation when there is VPoM Memory Read

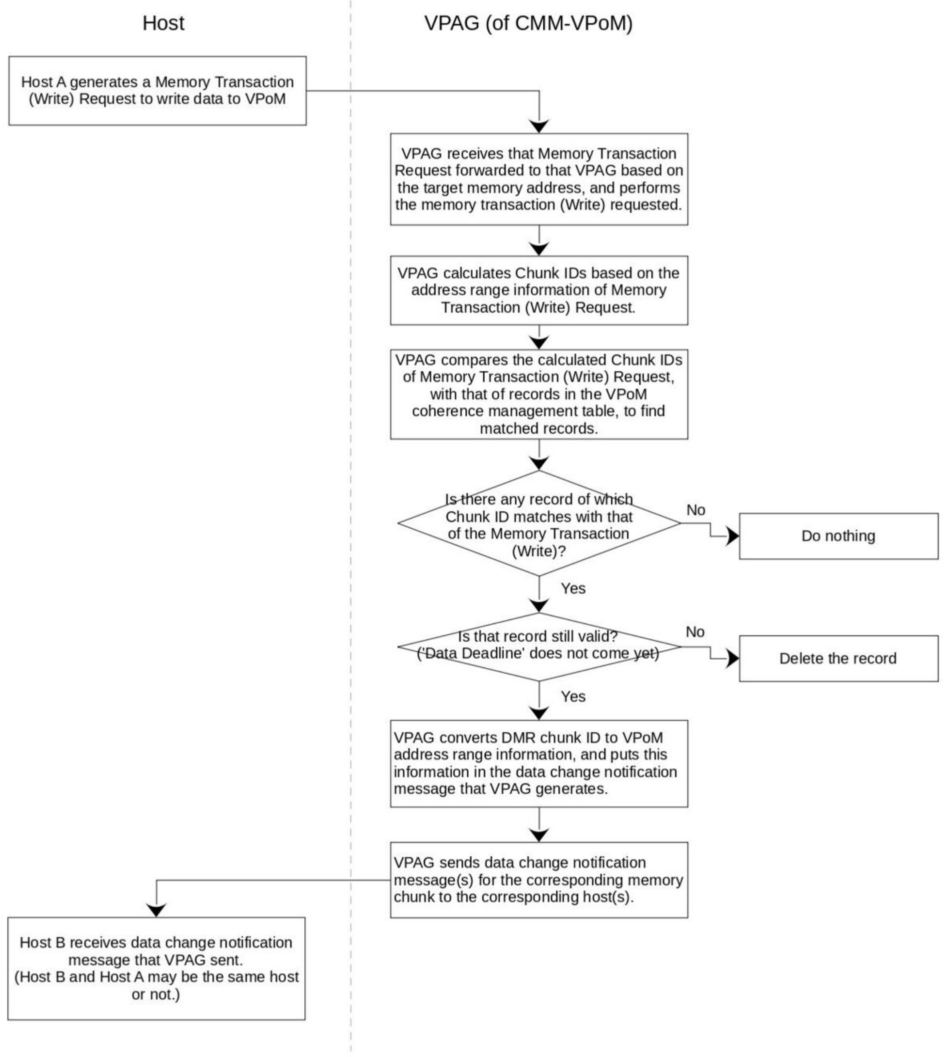

Host
VPAG (of CMM-VPoM)
Host A generates a Memory Transaction (Write) Request to write data to VPoM
VPAG receives that Memory Transaction Request forwarded to that VPAG based on the target memory address, and performs the memory transaction (Write) requested.
VPAG calculates Chunk IDs based on the address range information of Memory Transaction (Write) Request.
VPAG compares the calculated Chunk IDs of Memory Transaction (Write) Request, with that of records in the VPoM coherence management table, to find matched records.
Is there any record of which Chunk ID matches with that of the Memory Transaction (Write)?
No
Do nothing
Yes
Is that record still valid? ('Data Deadline' does not come yet)
No
Delete the record
Yes
VPAG converts DMR chunk ID to VPoM address range information, and puts this information in the data change notification message that VPAG generates.
VPAG sends data change notification message(s) for the corresponding memory chunk to the corresponding host(s).
Host B receives data change notification message that VPAG sent. (Host B and Host A may be the same host or not.)

Figure B4. VPoM Coherence Management Operation when there is VPoM Memory Write

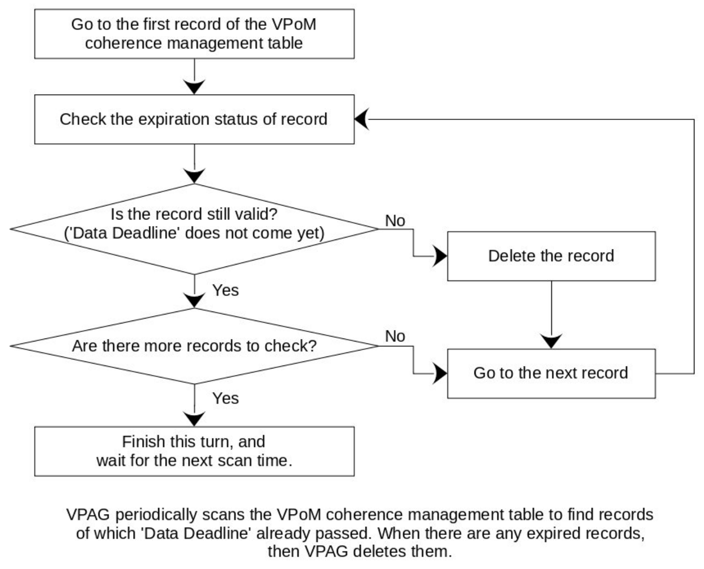

Go to the first record of the VPoM coherence management table
Check the expiration status of record
Is the record still valid? ('Data Deadline' does not come yet)
No
Delete the record
Yes
Are there more records to check?
No
Go to the next record
Yes
Finish this turn, and wait for the next scan time.
VPAG periodically scans the VPoM coherence management table to find records of which 'Data Deadline' already passed. When there are any expired records, then VPAG deletes them.

Figure B5. VPoM Coherence Management Operation as Periodic Housekeeping Task Loop

(P11N:C) Scheduled Prefetch Mechanism for VPoM systems: Details of Invention

Summary of Invention

- VPoM (Virtual Pool of Memory) is a shared memory system virtually constructed with a set of donated memory regions from individual CXL Memory Modules (CMMs). VPoM provides more consistent memory bandwidth and lower latency compared to the centralized and physical CXL memory pool, thanks to VPoM architecture itself, coherence management scheme, and VPoM's performance enhancement features such as prefetch and stream processing.
- This invention series describes architecture and key operations of VPoM (Virtual Pool of Memory) systems, and special features which minimize the performance bottleneck.
- This invention discloses
  - ~~architecture and key operations of VPoM (Virtual Pool of Memory) systems.~~
  - ~~coherence management scheme of VPoM systems.~~
  - performance enhancement features designed for VPoM (scheduled prefetch).
  - ~~performance enhancement features designed for VPoM (stream processing).~~

VPAG: VPoM Remote Data Prefetch Manager

- VPAG prefetches data when there is an explicit intention flag in the memory transaction (read) request. If '1b' is the value of 'Prefetch::Prefetch-Needed' field in the CXL Memory Transaction Packet (Read), then VPAG checks additional prefetch related fields for processing, such as 'Prefetch::Before-This-Time', 'Prefetch::After-This-Time', 'Prefetch::Data-Store-Location', 'Prefetch::Prefetch-Done-Notification-Channel' fields.
- VPAG calculates the proper time to actually prefetch the data considering the amount of data to read and the target DMR's latency and bandwidth information. Absolutely this actual prefetch time should be sufficiently earlier than 'Prefetch::Before-This-Time' specified. For example, if the amount of data to read is 100GB, and target DMR's max bandwidth is 10GB/s, then it means it will take at least 10 seconds to complete the prefetch operation. VPAG also considers the performance degradation effect caused by the other concurrent memory read and write transactions. For example, if there will be additional scheduled 100GB prefetch operation from the same DMR at the same time, Then VPAG schedules the prefetch operation at least 20 seconds before the specified 'Prefetch::Before-This-Time'.
- VPAG also provides a way to trigger data prefetch only after some specified memory transaction is done. This feature is useful when there is a need to read (prefetch) the data at specified memory location, only after some operation is done (e.g., some data is scheduled to be written to that memory area, and the user wants to read (prefetch) that data). Prefetch::After-This-Transaction field will be used to control this feature.
- To support the 'Prefetch::Before-This-Time' and 'Prefetch::After-This-Transaction' features, VPoM Remote Data Prefetch Manager has Timer and Transaction Completion Monitor. When both 'Prefetch::Before-This-Time' and 'Prefetch::After-This-Transaction' are set in the VPoM memory transaction request, then VPoM Remote Data Prefetch Manager schedules prefetch action to satisfy both of the conditions (i.e., "AND" condition). If there is a case that both of the condition can not be met simultaneously, then VPoM Remote Data Prefetch Manager raises error exception so that requester can handle that.

- VPAG provides data prefetch notification service to the host CPU (or other devices in the host). A method to request the data prefetch service is described as the following. When host CPU generates a memory transaction request packet, it sets a flag in the 'Prefetch::Prefetch-Needed' field, specifies an expected time in the 'Prefetch::Before-This-Time' field, specifies preferred location to store the prefetched data in the 'Prefetch::Data-Store-Location' field, specifies the interrupt type and ID in the 'Prefetch::Prefetch-Done-Notification-Channel' field. These fields can be implemented in the reserved area of the CXL memory request packet.
- Figure C1 depicts VPoM Remote Data Prefetch Manager architecture and related operations described above.

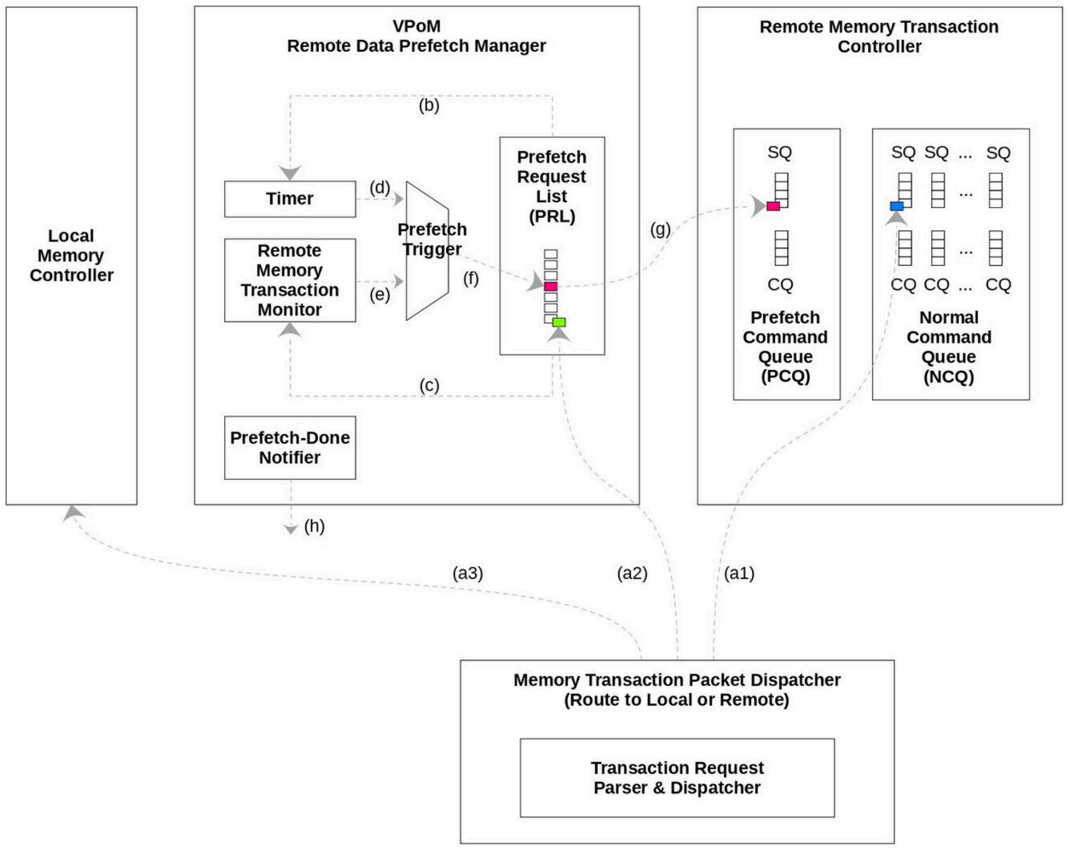

Local Memory Controller
VPoM Remote Data Prefetch Manager
(b)
Timer
(d)
Remote Memory Transaction Monitor
(e)
Prefetch Trigger
(f)
Prefetch Request List (PRL)
(c)
Prefetch-Done Notifier
(h)
(g)
Remote Memory Transaction Controller
SQ
CQ
Prefetch Command Queue (PCQ)
SQ SQ ... SQ
CQ CQ ... CQ
Normal Command Queue (NCQ)
(a3)
(a2)
(a1)
Memory Transaction Packet Dispatcher (Route to Local or Remote)
Transaction Request Parser & Dispatcher

Figure C1. VPoM Data Prefetch Manager Architecture and Related Operations

VPoM Remote Data Prefetch Manager architecture and related operations are depicted in the Figure 15. Detailed description of operations are following.

- (**a1**) For a normal remote memory transaction request (prefetch: no), **Memory Transaction Packet Dispatcher** enqueues a command to the **Normal Command Queue (NCQ)** in the **Remote Memory Transaction Controller**.
- (**a2**) For a remote memory transaction request (prefetch: yes), **Memory Transaction Packet Dispatcher** adds the item to the **Prefetch Request List (PRL)**.
- (**a3**) For a normal local memory transaction request, **Memory Transaction Packet Dispatcher** enqueues a command to the **Local Memory Controller**.
- (**b**) **Timer** check the time to trigger prefetch action based on the 'Prefetch::Before-This-Time' information stored in the **Prefetch Request List (PRL)**.
- (**c**) **Remote Memory Transaction Monitor** checks the status of **Normal Command Queue (NCQ)** and also **Prefetch Command Queue (PCQ)** in the **Remote Memory Transaction Controller** to meet the condition specified in the 'Prefetch::After-This-Transaction' information stored in the **Prefetch Request List (PRL)**.
- (**d**) When the 'Prefetch::Before-This-Time' condition is met, then **Timer** notifies the **Prefetch Trigger** of this information.
- (**e**) When the 'Prefetch::After-This-Transaction' condition is met, then **Remote Memory Transaction Monitor** notifies **Prefetch Trigger** of this information.
- (**f**) **Prefetch Trigger** module determines prefetch requests to be enqueued to the **Prefetch Command Queue (PCQ)** in the **Remote Memory Transaction Controller**, based on the information given by **Timer** and **Remote Memory Transaction Monitor**.
- (**g**) **Prefetch Request List (PRL)** makes a command for the selected prefetch request, and enqueues the command to the **Prefetch Command Queue (PCQ)** in the **Remote Memory Transaction Controller**. **Remote Memory Transaction Controller** will execute the command in the SQ (Submission Queue) of **Prefetch Command Queue (PCQ)**, and will update the CQ (Completion Queue) of **Prefetch Command Queue (PCQ)** when the prefetch action is completed.
- (**h**) When a prefetch action is completed, then **Prefetch-Done Notifier** sends notification to the requester, based on the 'Prefetch::Prefetch-Done-Notification-Channel' information.

Figure C2 depicts the remote data prefetch related processing flows among the **Memory Transaction Packet Dispatcher**, **VPoM Remote Data Prefetch Manager**, **Remote Memory Transaction Controller**, and **Local Memory Controller**.

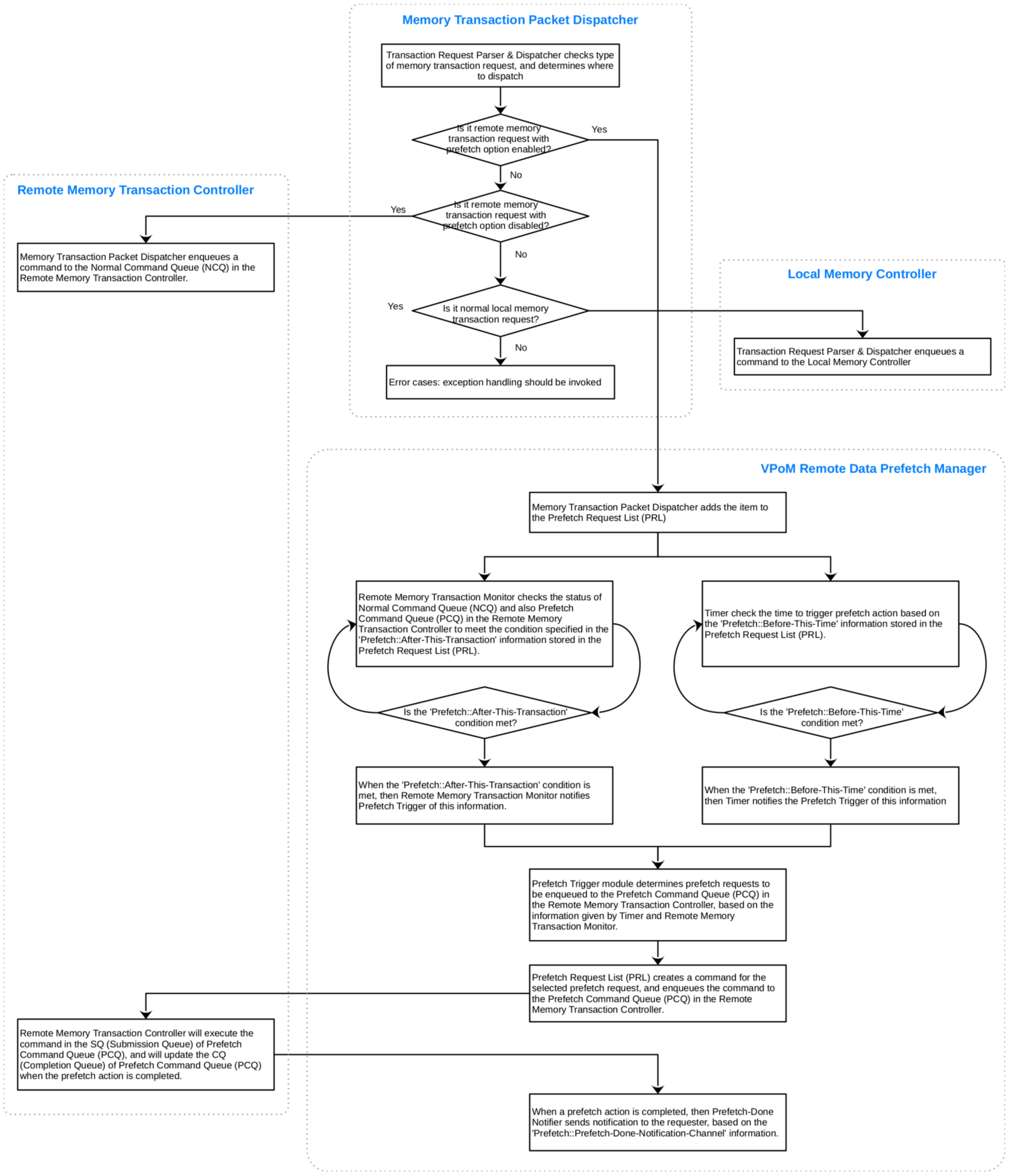

Memory Transaction Packet Dispatcher
Transaction Request Parser & Dispatcher checks type of memory transaction request, and determines where to dispatch
Is it remote memory transaction request with prefetch option enabled?
Yes
No
Is it remote memory transaction request with prefetch option disabled?
Yes
No
Is it normal local memory transaction request?
Yes
No
Error cases: exception handling should be invoked
Remote Memory Transaction Controller
Memory Transaction Packet Dispatcher enqueues a command to the Normal Command Queue (NCQ) in the Remote Memory Transaction Controller.
Local Memory Controller
Transaction Request Parser & Dispatcher enqueues a command to the Local Memory Controller
VPoM Remote Data Prefetch Manager
Memory Transaction Packet Dispatcher adds the item to the Prefetch Request List (PRL)
Remote Memory Transaction Monitor checks the status of Normal Command Queue (NCQ) and also Prefetch Command Queue (PCQ) in the Remote Memory Transaction Controller to meet the condition specified in the 'Prefetch::After-This-Transaction' information stored in the Prefetch Request List (PRL).
Timer check the time to trigger prefetch action based on the 'Prefetch::Before-This-Time' information stored in the Prefetch Request List (PRL).
Is the 'Prefetch::After-This-Transaction' condition met?
Is the 'Prefetch::Before-This-Time' condition met?
When the 'Prefetch::After-This-Transaction' condition is met, then Remote Memory Transaction Monitor notifies Prefetch Trigger of this information.
When the 'Prefetch::Before-This-Time' condition is met, then Timer notifies the Prefetch Trigger of this information
Prefetch Trigger module determines prefetch requests to be enqueued to the Prefetch Command Queue (PCQ) in the Remote Memory Transaction Controller, based on the information given by Timer and Remote Memory Transaction Monitor.
Prefetch Request List (PRL) creates a command for the selected prefetch request, and enqueues the command to the Prefetch Command Queue (PCQ) in the Remote Memory Transaction Controller.
Remote Memory Transaction Controller will execute the command in the SQ (Submission Queue) of Prefetch Command Queue (PCQ), and will update the CQ (Completion Queue) of Prefetch Command Queue (PCQ) when the prefetch action is completed.
When a prefetch action is completed, then Prefetch-Done Notifier sends notification to the requester, based on the 'Prefetch::Prefetch-Done-Notification-Channel' information.

Figure C2. VPoM remote data prefetch processing flows

Table C1. VPoM Prefetch related fields in the Memory Transaction Request Packet.

field name	field value type
Prefetch::Prefetch-Needed	▪ If this field value is 0b, then it means there is no need to do prefetch. ▪ If this field value is 1b, then it means that prefetch is needed (explicit intention to prefetch).
Prefetch::Before-This-Time	▪ If msb of this field value is 0b, then the remaining value of this field means relative time from this request. ▪ If msb of this field value is 1b, then the remaining value of this field means absolute time (UTC).
Prefetch::After-This-Transaction	▪ If msb of this field value is 0b, VPAG's data prefetch manager does not care about the execution order of VPoM memory transaction. The remaining value of this field will be ignored. ▪ If msb of this field value is 1b, VPAG's data prefetch manager guarantees that this prefetch should be performed only after specified VPoM memory transaction. The remaining value of this field is used to specify the VPoM memory transaction ID. In actual application use cases, for example, transaction ID will be returned when programmer calls API for VPoM memory read or write transaction. Then the programmer can fill the Prefetch::After-This-Transaction field with this transaction ID, when they create a VPoM memory read (prefetch) transaction with Prefetch::After-This-Transaction enabled.
Prefetch::Data-Store-Location	▪ If msb of this field value is 0b, then VPAG will place the prefetched data on to the CMM-VPoM local memory. The remaining value of this field means the starting address of the prefetched data in the CMM-VPoM local memory. ▪ If msb of this field value is 1b, then VPAG will place the prefetched data on to the host local memory. The remaining value of this field means the starting address of the prefetched data in the host CPU memory.
Prefetch::Prefetch-Done-Notification-Channel	▪ msb 2-bit of this field denotes notification method type. If msb 2-bit of this field is 00b, then there will be no prefetch-done notification to the requester, and the remaining value of this filed will be ignored. If msb 2-bit of this field has other values, then it denotes the the requester wants to get notified when the prefetch action is done, and the remaining value of this field will be used as a specification of notification channel preferred. ▪ 00b: no need to get notified when the prefetch is completed ▪ 01b: MSI ▪ 10b: MSI-X ▪ 11b: Other custom notification method/channel ▪ The remaining value of this field denotes the notification channel ID to be used for this request.

(P11N:D) Stream Processing Architecture for VPoM Systems: Details of Invention

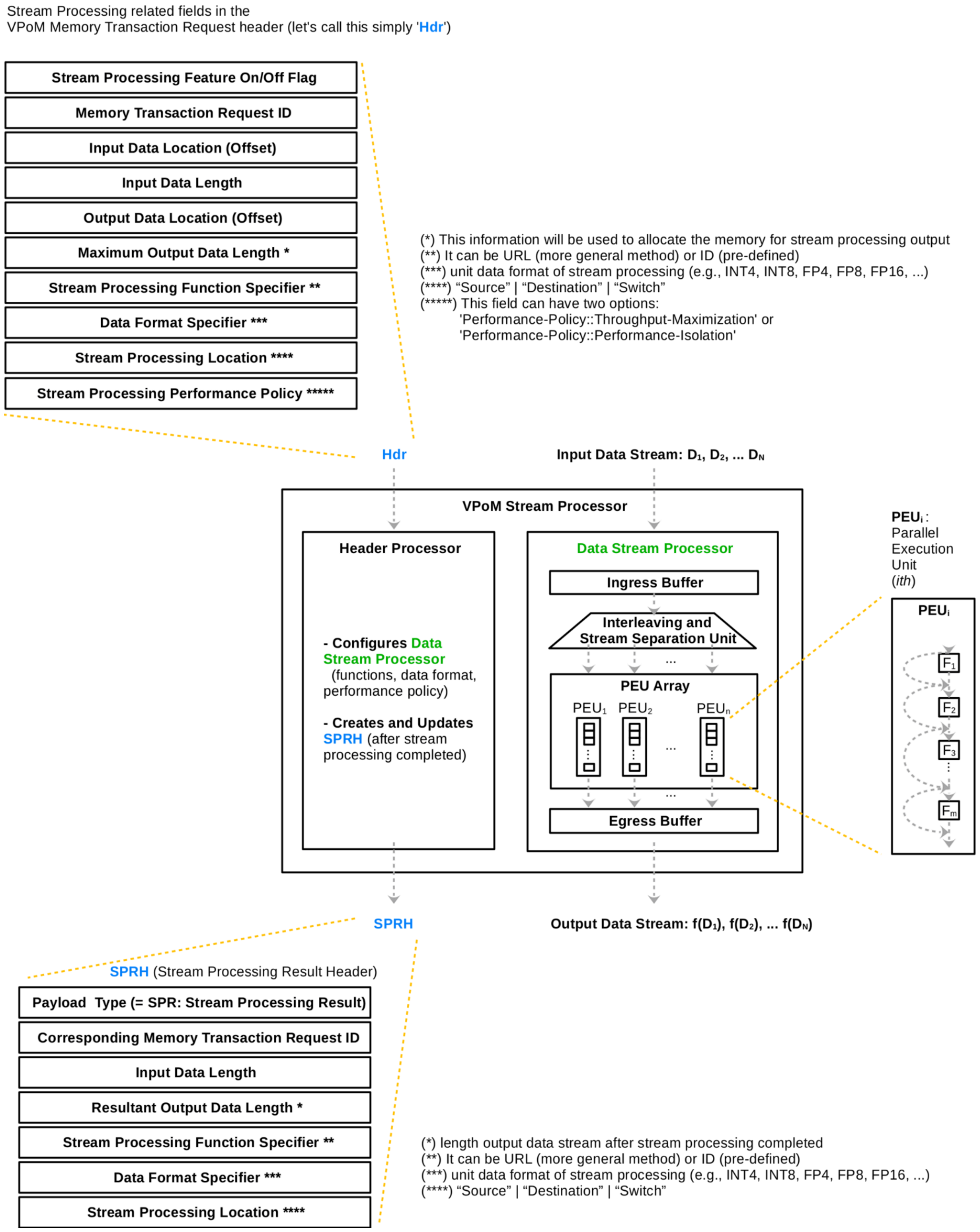
Summary of Invention

- VPoM (Virtual Pool of Memory) is a shared memory system virtually constructed with a set of donated memory regions from individual CXL Memory Modules (CMMs). VPoM provides more consistent memory bandwidth and lower latency compared to the centralized and physical CXL memory pool, thanks to VPoM architecture itself, coherence management scheme, and VPoM's performance enhancement features such as prefetch and stream processing.
- This invention series describes architecture and key operations of VPoM (Virtual Pool of Memory) systems, and special features which minimize the performance bottleneck.
- This invention discloses
  - ~~architecture and key operations of VPoM (Virtual Pool of Memory) systems.~~
  - ~~coherence management scheme of VPoM systems.~~
  - ~~performance enhancement features designed for VPoM (scheduled prefetch).~~
  - performance enhancement features designed for VPoM (stream processing).

VPAG: VPoM Stream Processor

- VPAG has stream processing feature designed for VPoM systems. The purpose of this feature is to reduce the traffic overhead in the interconnect network for memory pool (VPoM), or to enhance the level of security, or to reduce or hide the computation time cooperating with VPoM's remote data prefetch feature. VPoM stream processing is performed by VPAG's **VPoM Stream Processor**, of which architecture and related data structures are depicted in Figure D1.
- Possible examples of stream processing functions are (but not limited to): to find a maximum value or minimum value, to calculate average value, to select data which satisfy the arithmetic condition (equa to, larger than, less than, larger than or equal to, less than or equal to), to encrypt or decrypt data, to calculate the data integrity checksum, to calculate softmax, to quantize the data, to transform the data format, and so on.
- **VPoM Stream Processor** consists of **Header Processor** and **Data Stream Processor**.
- **Header Processor** configures the actions to be done by **Data Stream Processor** based on the information specifed in the Stream Processing related header, such as 'Stream Processing Feature On/Off Flag', 'Stream Processing Function Specifier', 'Data Format Specifier', 'Stream Processing Performance Policy'. If 'Stream Processing Feature On/Off Flag' is set to 'Stream-Processing-Feature-On', then VPAG's **VPoM Stream Processor** performs the required action for stream processing, based on the information specifed in the Stream Processing related fields. **Header Processor** creates and updates 'SPRH (Stream Processing Result Header)' fields after stream processing completed, especially 'Resultant Output Data Length'. Information described in the SPRH fields will be used (1) to match the stream processing resultant data and the corresponding VPoM Memory Transaction Request (based on the 'Corresponding Memory Transaction Request ID' information) and (2) to verify the processing result.
- **Data Stream Processor** consists of **Ingress Buffer**, **Interleaving and Stream Separation Unit**, **PEU (Parallel Execution Unit) Array**, and **Egress Buffer**.
- Based on the 'Stream Processing Performance Policy' information defined in the VPoM Memory Transaction Request header, **Data Stream Processor**'s **Interleaving and Stream Separation Unit** determines the way to use **PEU**s (Parallel Execution Units) in the **PEU Array**. If 'Stream Processing Performance Policy' is set to 'Performance-Policy::Throughput-Maximization', then **Interleaving and Stream Separation Unit** will operate in the 'Interleaving (across PEUs)' mode. If 'Stream Processing Performance Policy' is set to 'Performance-Policy::Performance-Isolation', then **Interleaving and Stream Separation Unit** will operate in the 'Stream Separation' mode.
- **PEU (Parallel Execution Unit)** implements one or multiple functions. **PEU** provides a way to select one or multiple functions to execute. When multiple functions are selected, **PEU** provides chained execution of selected functions, which is configured by 'Stream Processing Function Specifier' information in the Stream Processing related fields in the VPoM Memory Transaction Request header.
- 'Stream Processing Location' can have one of the values 'Source', 'Destination', and 'Switch'.
- VPoM.GM (and VPoM.LM) provides the list of supported stream processing functions, and list of data formats supported for stream processing, such as FP32, FP16, FP8, FP4, INT8, and INT4. This data is provided as a VPoM system capability information so that user (human programmer or automated system) of VPoM systems can choose proper functions to be executed as a stream processing.

- VPoM stream processing can be performed in the VPAG of the source node, or in the VPAG of the destination node, or even in the CXL Switch. 'Stream Processing Location' field in the VPoM Memory Transaction Request header specifies this.
- **VPoM Stream Processor** reads and processes data of 'Input Data Length', from the 'Input Data Location'. Output data processed by **VPoM Stream Processor** will be stored at the memory location specified by 'Output Data Location'. 'Maximum Output Data Length' information will be used by VPAG to allocate the memory to store the stream processing output data.


Stream Processing related fields in the
VPoM Memory Transaction Request header (let's call this simply 'Hdr')
Stream Processing Feature On/Off Flag
Memory Transaction Request ID
Input Data Location (Offset)
Input Data Length
Output Data Location (Offset)
Maximum Output Data Length *
Stream Processing Function Specifier **
Data Format Specifier ***
Stream Processing Location ****
Stream Processing Performance Policy *****
Hdr
Input Data Stream: D1, D2, ... DN
VPoM Stream Processor
Header Processor
- Configures Data Stream Processor (functions, data format, performance policy)
- Creates and Updates SPRH (after stream processing completed)
Data Stream Processor
Ingress Buffer
Interleaving and Stream Separation Unit
PEU Array
PEU1
PEU2
PEUn
Egress Buffer
PEUi : Parallel Execution Unit (ith)
PEUi
F1
F2
F3
Fm
SPRH
Output Data Stream: f(D1), f(D2), ... f(DN)
SPRH (Stream Processing Result Header)
Payload Type (= SPR: Stream Processing Result)
Corresponding Memory Transaction Request ID
Input Data Length
Resultant Output Data Length *
Stream Processing Function Specifier **
Data Format Specifier ***
Stream Processing Location ****

(*) This information will be used to allocate the memory for stream processing output
(**) It can be URL (more general method) or ID (pre-defined)
(***) unit data format of stream processing (e.g., INT4, INT8, FP4, FP8, FP16, ...)
(****) “Source” | “Destination” | “Switch”
(*****) This field can have two options:
'Performance-Policy::Throughput-Maximization' or
'Performance-Policy::Performance-Isolation'

(*) length output data stream after stream processing completed
(**) It can be URL (more general method) or ID (pre-defined)
(***) unit data format of stream processing (e.g., INT4, INT8, FP4, FP8, FP16, ...)
(****) “Source” | “Destination” | “Switch”

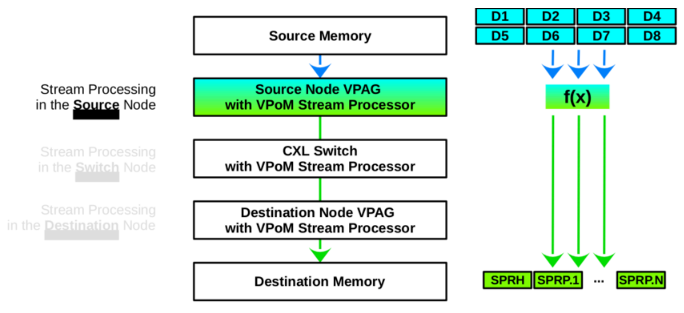
Figure D1. VPoM Stream Processor Architecture and related data structures

- Figure D2a, D2b, and D2c illustrates the possible different cases of stream processing, in terms of processing locations (source, destination, switch).
- Figure D3a, D3b, and D3c illustrates example cases of possible different cases of stream processing, based on the exemplary system which consists of three hosts and one CXL switch.
- When the "Stream-Processing-Location" flag has been set as 'Processing-in-the-Source-Node', Stream processing is being done in the source node VPAG's VPoM Stream Processor. We can use this operational model when the operator (function) satisfies associative property. We may choose this operational model when the size of output data of a stream processing function is smaller than that of input data, such as max(large amount of data), min(large amount of data), sum(large amount of data),

select(select condition, large amount of data), and so on. This may significantly reduce the data traffic on the CXL network. Operational model and example cases are illustrated in Figure D2a and Figure D3a.

- When the "Stream-Processing-Location" flag has been set as 'Processing-in-the-Destination-Node', Stream processing is being done in the destination node VPAG's VPoM Stream Processor. We need to use this operational model when the operator (function) does not satisfy associative property. We may choose this operational model when the operation makes the size of output data larger than that of input data, such as decompression. Operational model and example cases are illustrated in Figure D2b and Figure D3b.
- When the "Stream-Processing-Location" flag has been set as 'Processing-in-the-Switch', Stream processing is being done in the intermediate CXL Switch's VPoM Stream Processor. We can use this operational model when the operator (function) satisfies associative property. We may choose this operational model when the size of output data of a stream processing function is smaller than that of input data, such as max(large amount of data), min(large amount of data), sum(large amount of data), select(select condition, large amount of data), and so on. This may significantly reduce the data traffic on the CXL network. Operational model and example cases are illustrated in Figure D2c and Figure D3c.

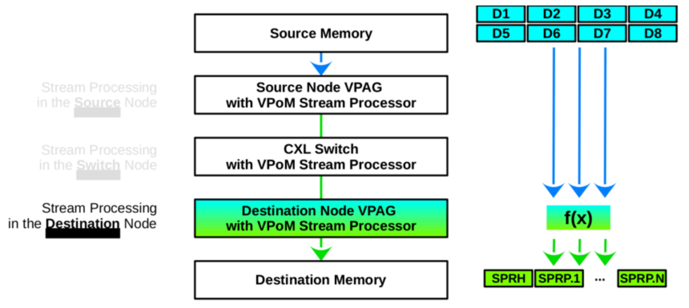
Figure D2. Stream Processing Type in terms of Processing Locations (Source, Destination, Switcl


Source Memory
D1
D2
D3
D4
D5
D6
D7
D8
Stream Processing in the Source Node
Source Node VPAG with VPoM Stream Processor
f(x)
Stream Processing in the Switch Node
CXL Switch with VPoM Stream Processor
Stream Processing in the Destination Node
Destination Node VPAG with VPoM Stream Processor
Destination Memory
SPRH
SPRP.1
...
SPRP.N

Operation model:
When the "Stream-Processing-Location" flag has been set as 'Processing-in-the-Source-Node', Stream processing is being done in the source node VPAG's VPoM Stream Processor.

Application cases:
We can use this operational model when the operator (function) satisfies associative property.
We may choose this operational model when the size of output data of a stream processing function is smaller than that of input data, such as max(large amount of data), min(large amount of data), sum(large amount of data), select(select condition, large amount of data), and so on. This may significantly reduce the data traffic on the CXL network.

Figure D2a. Stream Processing in the Source Node


Source Memory
D1
D2
D3
D4
D5
D6
D7
D8
Stream Processing in the Source Node
Source Node VPAG with VPoM Stream Processor
Stream Processing in the Switch Node
CXL Switch with VPoM Stream Processor
Stream Processing in the Destination Node
Destination Node VPAG with VPoM Stream Processor
f(x)
Destination Memory
SPRH
SPRP.1
...
SPRP.N

Operation model:
When the "Stream-Processing-Location" flag has been set as 'Processing-in-the-Destination-Node', Stream processing is being done in the destination node VPAG's VPoM Stream Processor.

Application cases:
We need to use this operational model when the operator (function) does not satisfy associative property.
We may choose this operational model when the operation makes the size of output data larger than that of inpu data, such as decompression.

Figure D2b. Stream Processing in the Destination Node

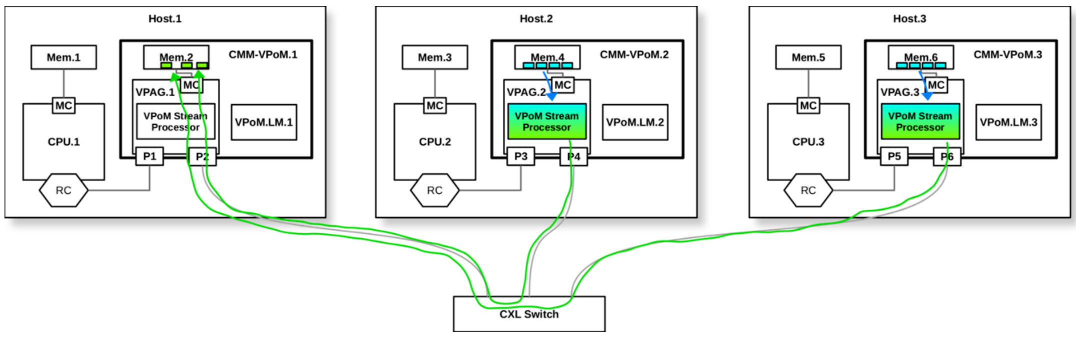


Figure D3a. An Example - Stream Processing in the Source Node

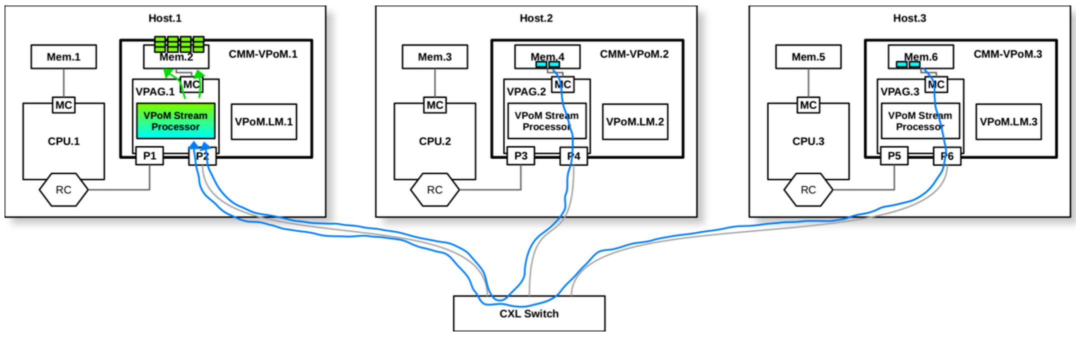


Figure D3b. An Example - Stream Processing in the Destination Node

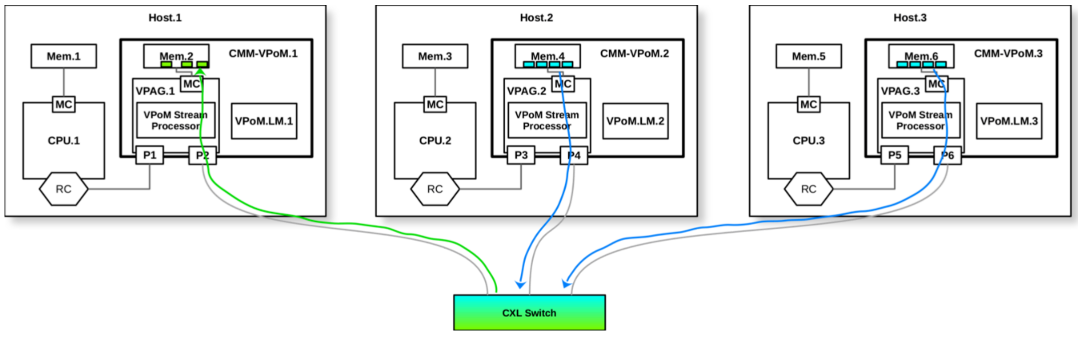


Figure D3c. An Example - Stream Processing in the Switch Node