MDF Runtime

1. 简介

MDF运行时支持在Intel GEN硬件中执行MDF内核。 最简单的使用模型包括一个用C / C ++编写并在CPU中运行的主机程序和一个用MDF语言编写并在GEN硬件中运行的内核。 主程序创建MDF设备（CmDevice），一个或多个输入 MDF surface（CmBuffer / CmSurface2D），一个或多个输出MDFsurface，以及一个或多个MDF内核（CmKernel）。 输入surface的内容可以根据主程序存储器中的内容进行初始化，或者输入surface可以是IDirect3DSurface9 surface（Windows中的DX9），ID3D11Texture2D surface（Windows中的DX11）或VA surface（Linux中）。 MDF内核获取输入surface，在GEN硬件中运行，并将结果写入输出surface。 主机程序可以获取输出surface的内容，也可以使用DirectX，MDF运行时或MediaSDK提供的接口进行进一步处理。

MDF运行时API定义了用于编写主机程序和处理主机程序与MDF内核之间的通信的类和函数。除了上面提到的CmDevice，CmBuffer / CmSurface2D和CmKernel，它还包括CmQueue，CmEvent和CmSampler。 CmQueue是一个以GPU为目标的有序任务队列，任务根据它们排队的顺序执行，在第一个任务完成执行之前，第二个任务不会开始执行。每个任务都有一个或多个同时运行的CmKernel。每个CmKernel通常有多个并行运行的线程。每个CmKernel都可以调用SetKernelArg来设置该内核的所有线程的常量参数值。它还可以调用SetThreadArg来设置该内核的不同线程可能不同的参数值。可以创建二维CmThreadSpace。 CmKernel的线程可以与CmThreadSpace中的单元相关联，并且可以在单元之间定义依赖模式。可以在Enqueue（）中使用此CmThreadSpace来定义线程依赖性。当任务（一个或多个并发运行的内核）入队时生成CmEvent。主程序可以查询CmEvent以确定相关任务是否已经完成（即该任务的所有内核都已完成）。

1. 名词解释

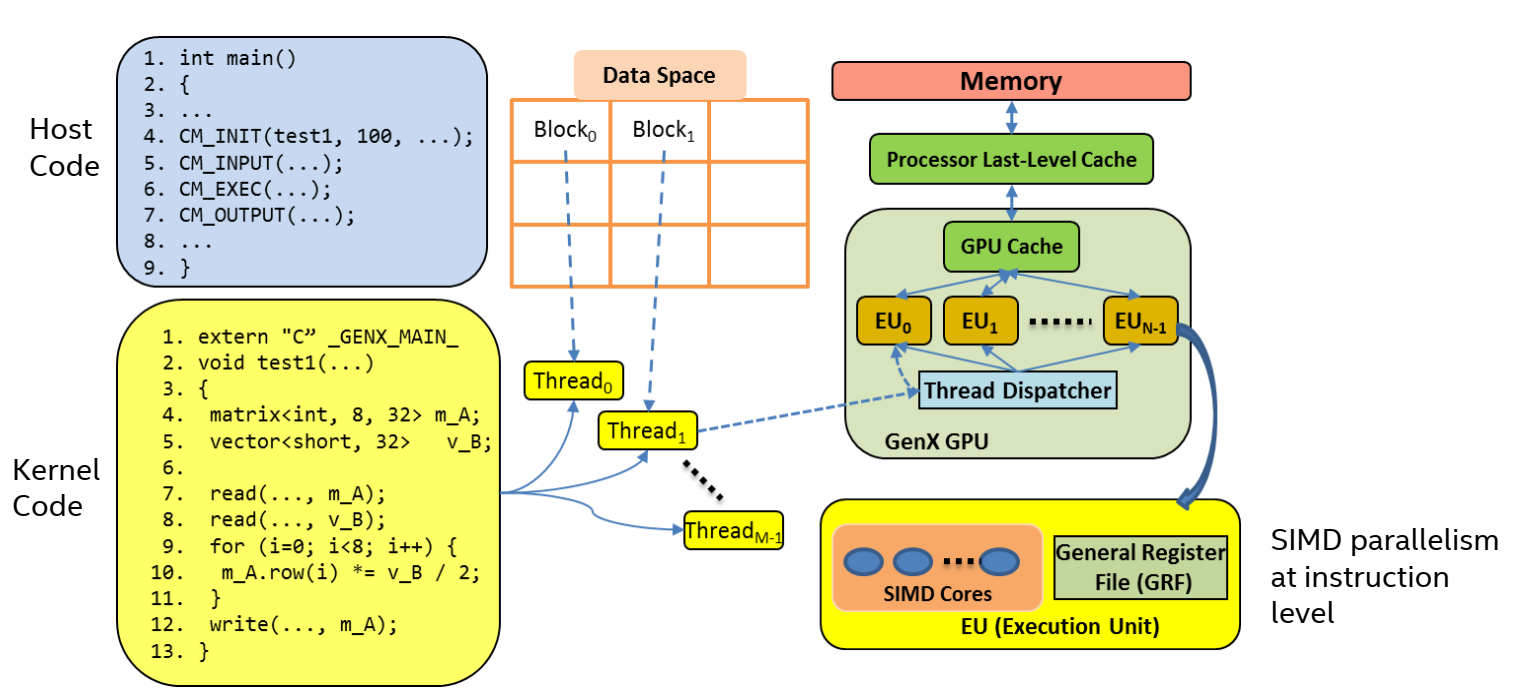
CM运行时：CM工具链的运行时部分，也称为CMRT。

内核：用CM语言编写的函数，由CM编译器编译，通过CM运行时在GPU中运行。

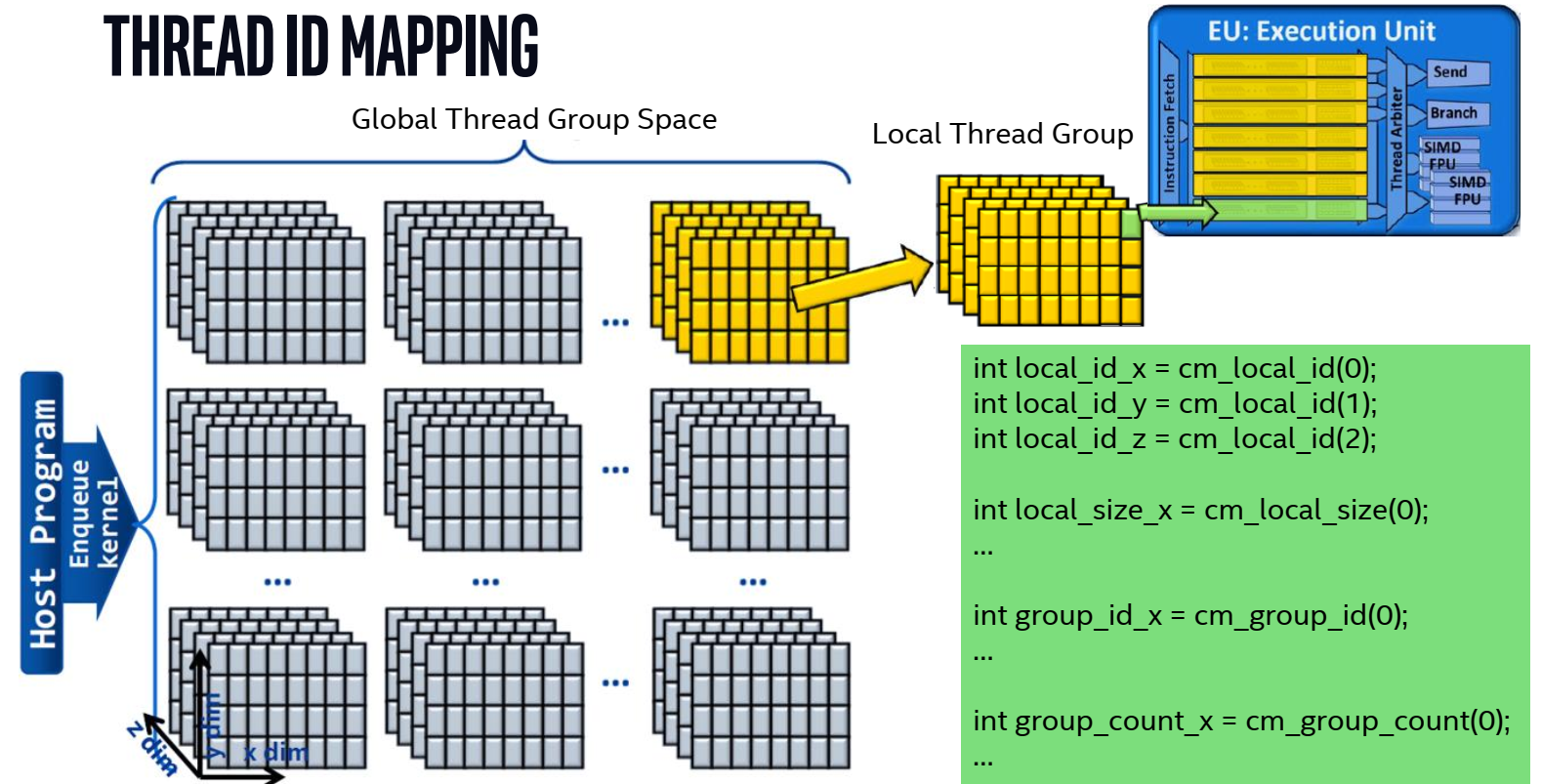
线程：每个内核可以并行运行多个实例。 每个实例都是一个线程。 例如，不同的线程可以处理图像的不同宏块。

任务：并发运行的多个线程。 这些线程可以来自不同的内核。 层次结构是一个包含多个内核的任务，每个内核可以在多个线程中运行，任务的所有线程同时运行

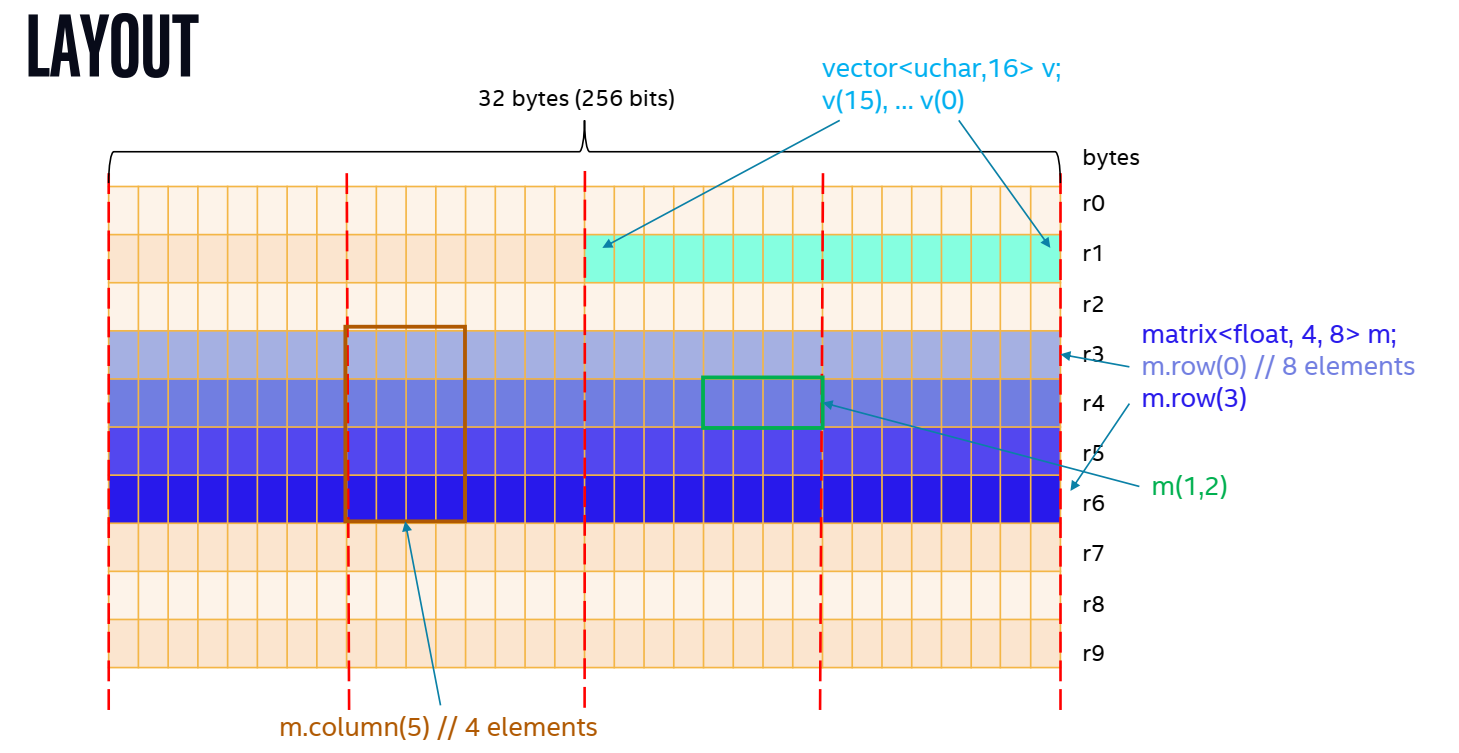
1. 编程模型



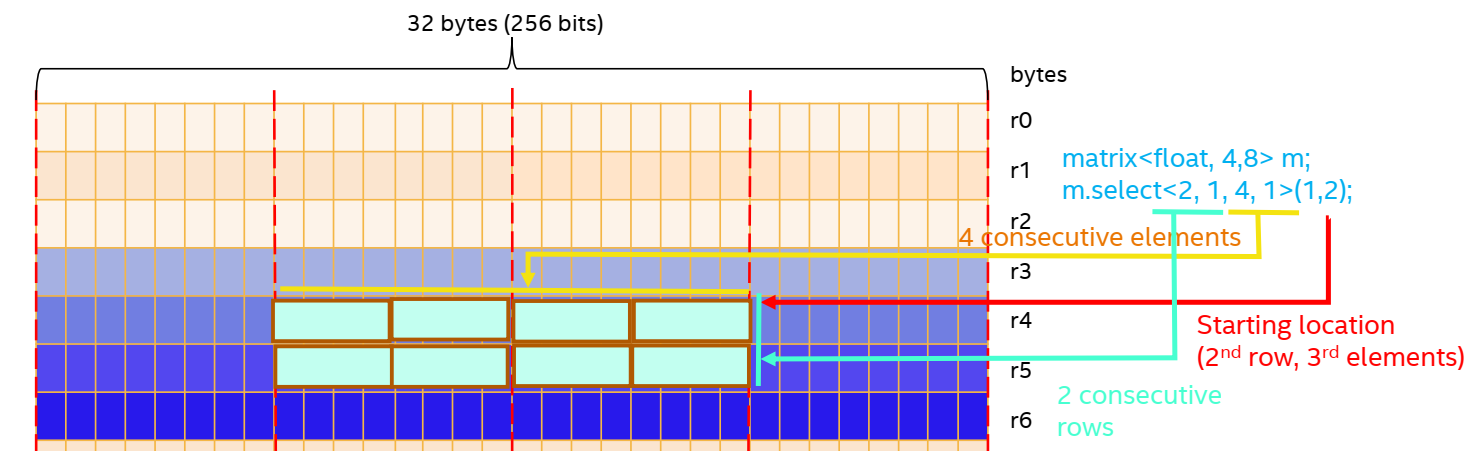
1. CM 线程映射关系



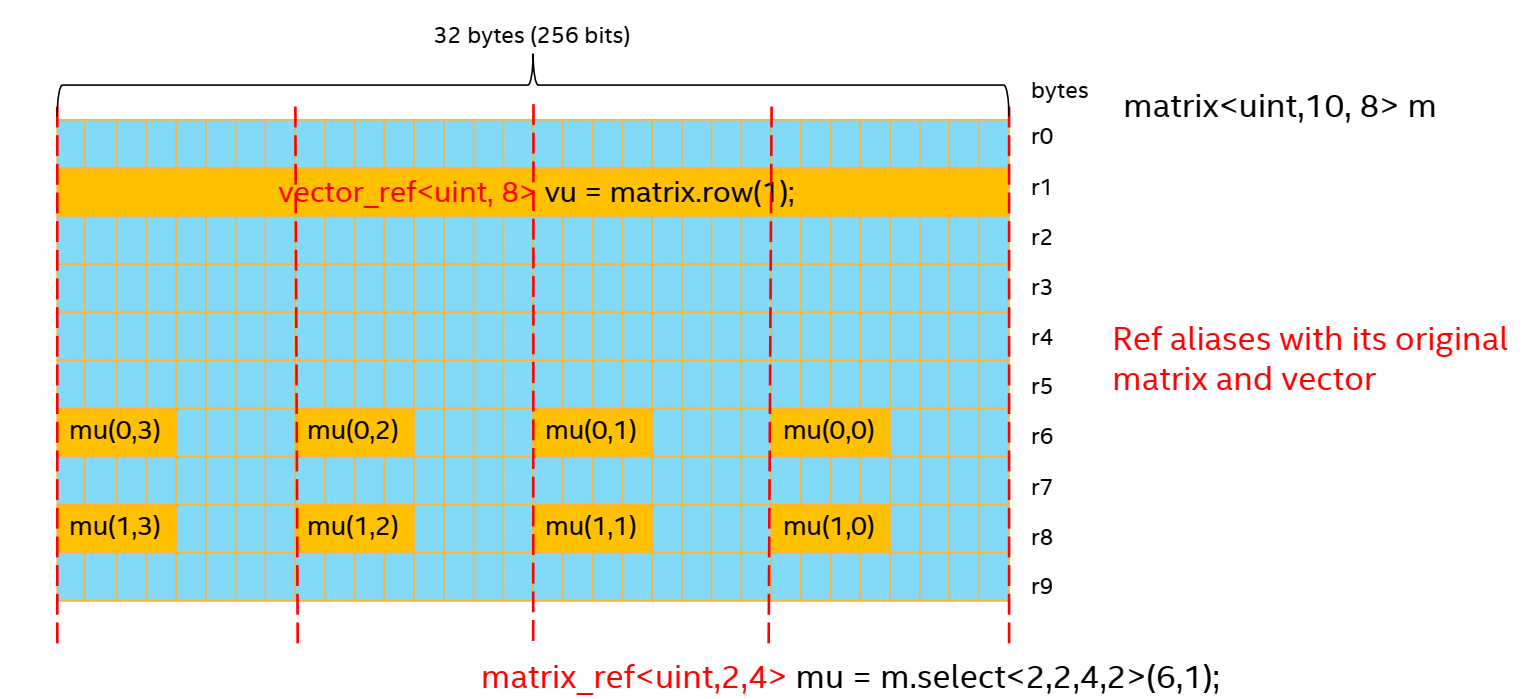
1. CM 数据结构 ，vector<type.N>: N elements of type , matrix<type,rows,columns> ,会映射到GRF寄存器文件上



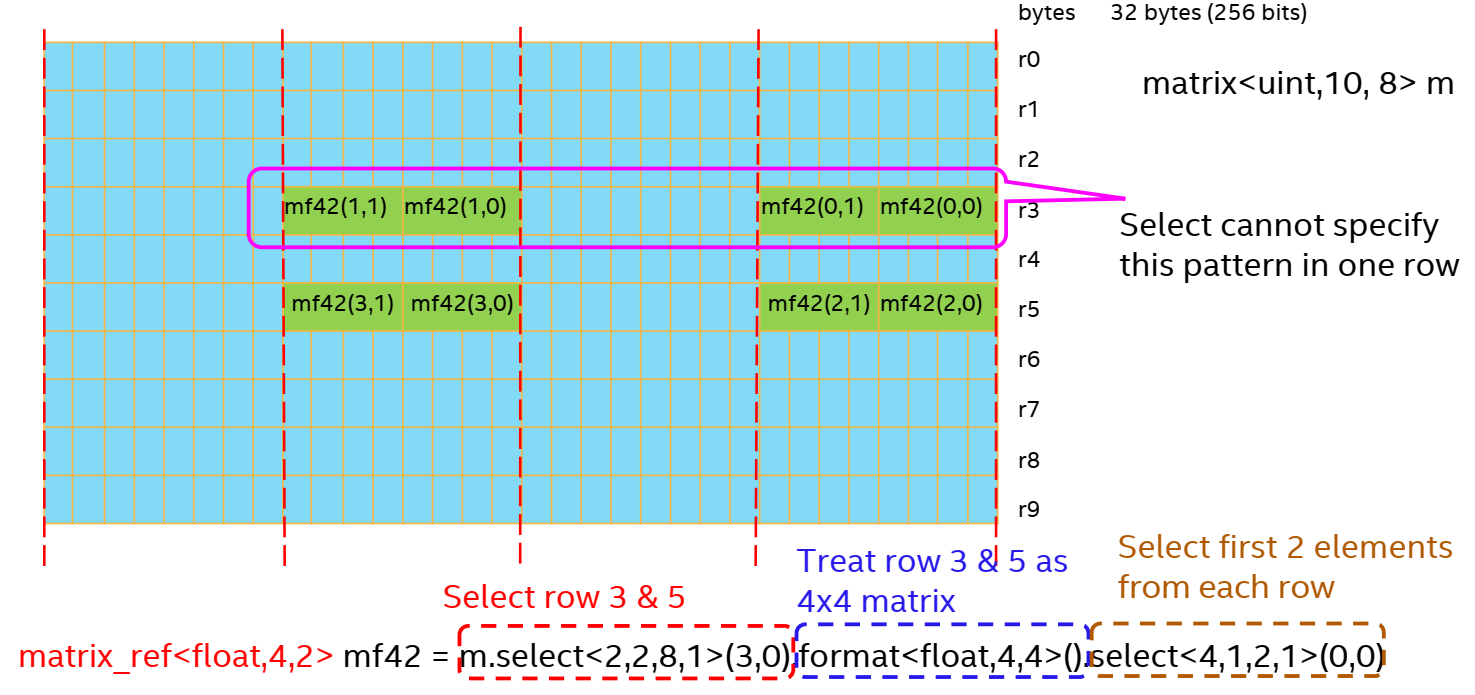
寄存器文件的regioning操作 ，select



寄存器的reference不会占用额外的寄存器资源，相当于别名



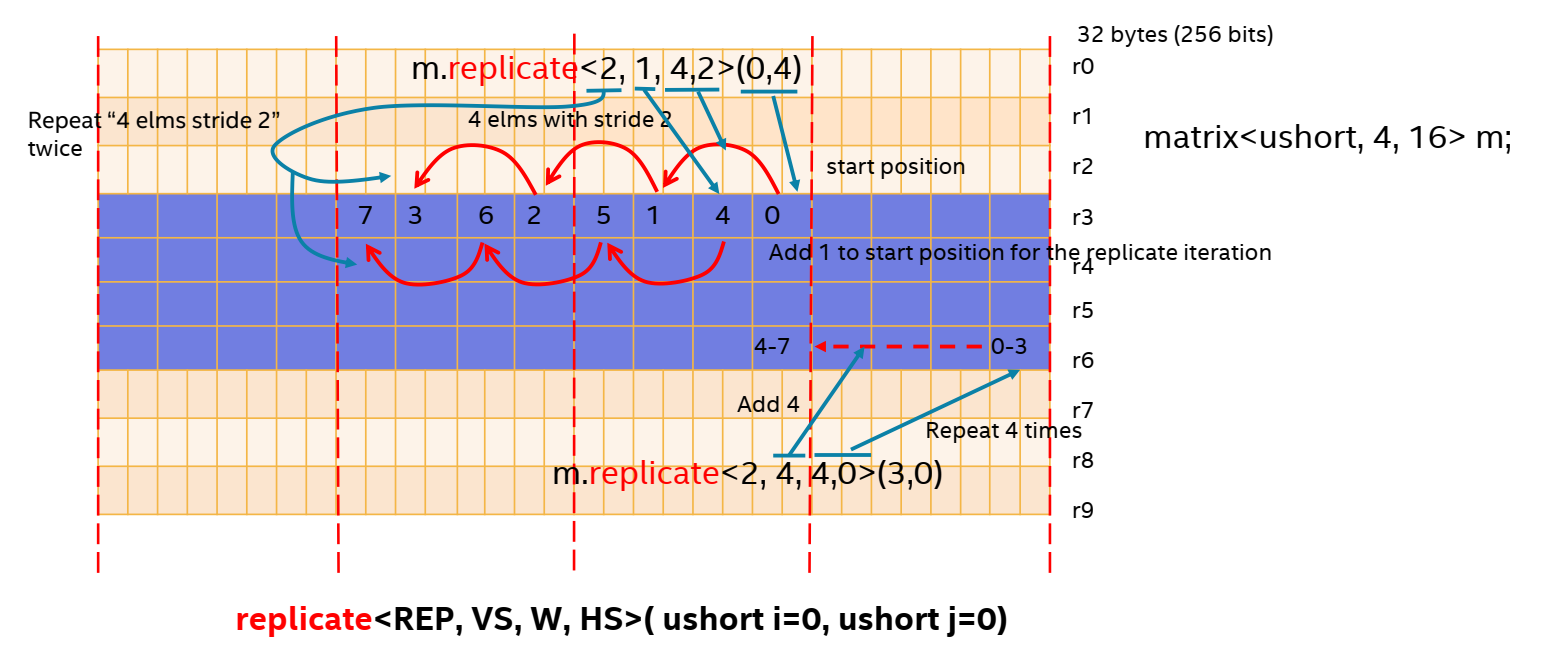
数据的格式转换，format



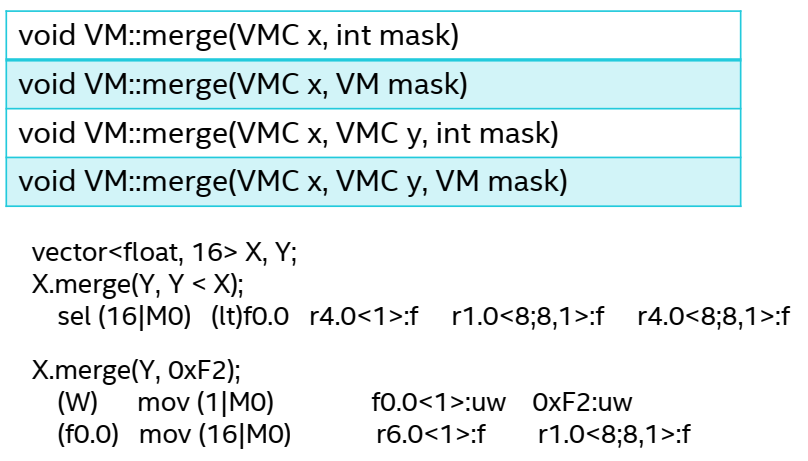
Replicate操作可以重复选中的一维线性elements, 构成一个新的vector或者matrix

* **replicate<REP, VS, W, HS>( ushort i=0, ushort j=0)**

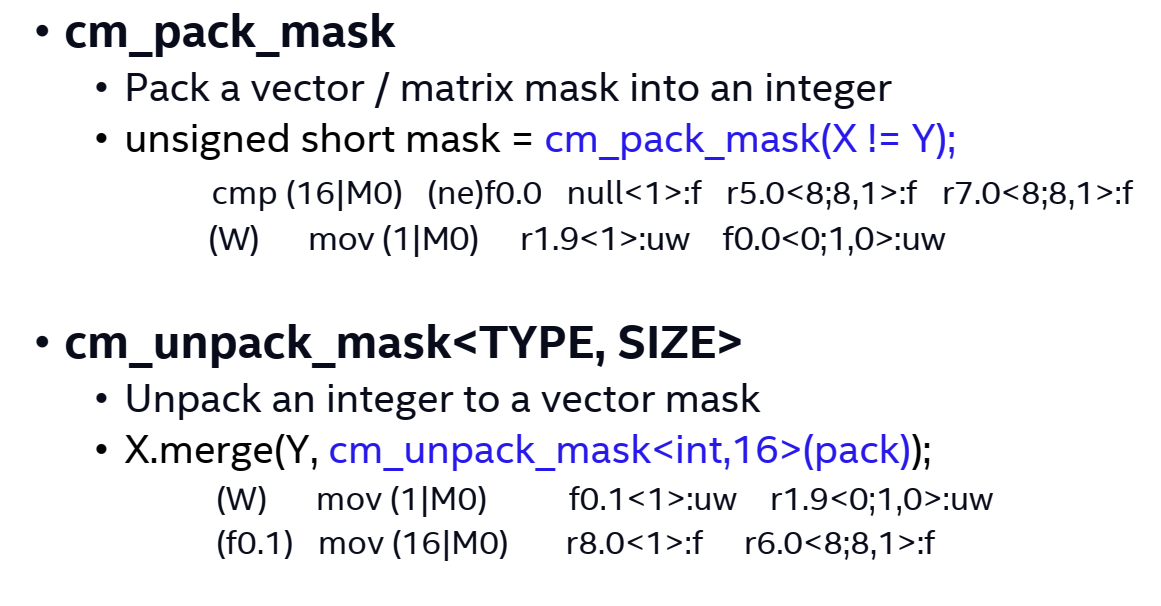
The following can be used in CM to select/replicate “REP” blocks of “W” sequential elements with a stride of “HS” starting at (i,j)/(i) from the matrix/vector object with each block strided by “VS” elements, and return a new vector of “REP” \* “W” length



根据mask来融合两个vector或者matrix

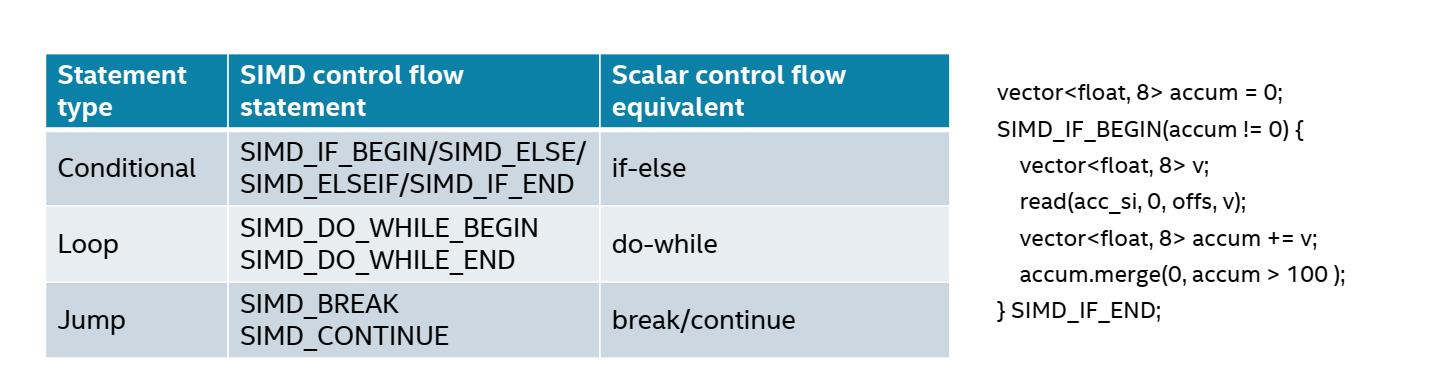


Mask的产生



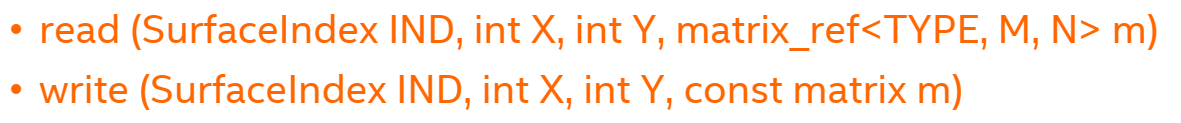
1. SIMD 控制流

Scalar operations are allowed and considered as replicated

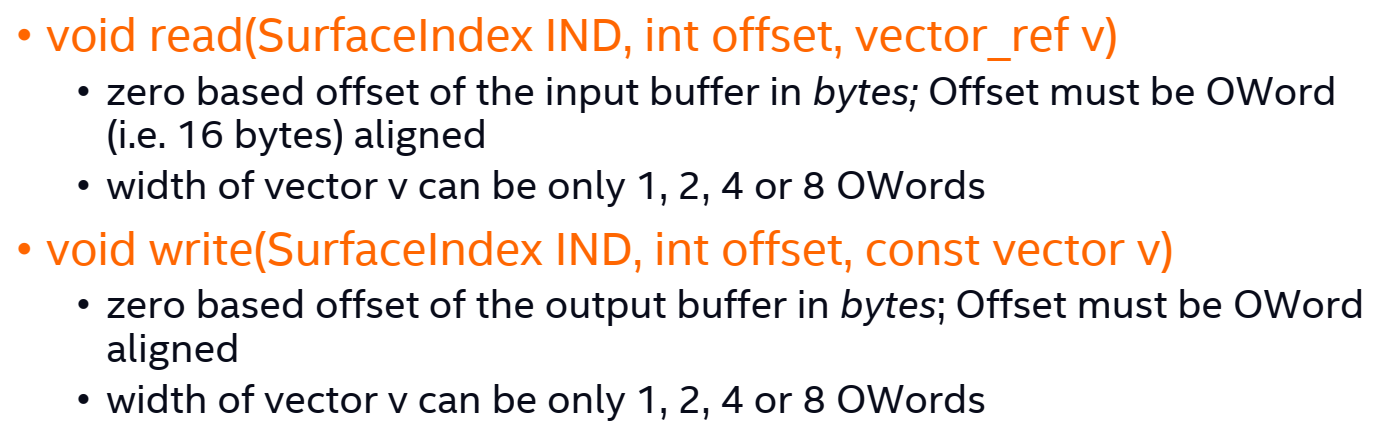


1. CM I/O

Media block R/W:



OWORD R/W:

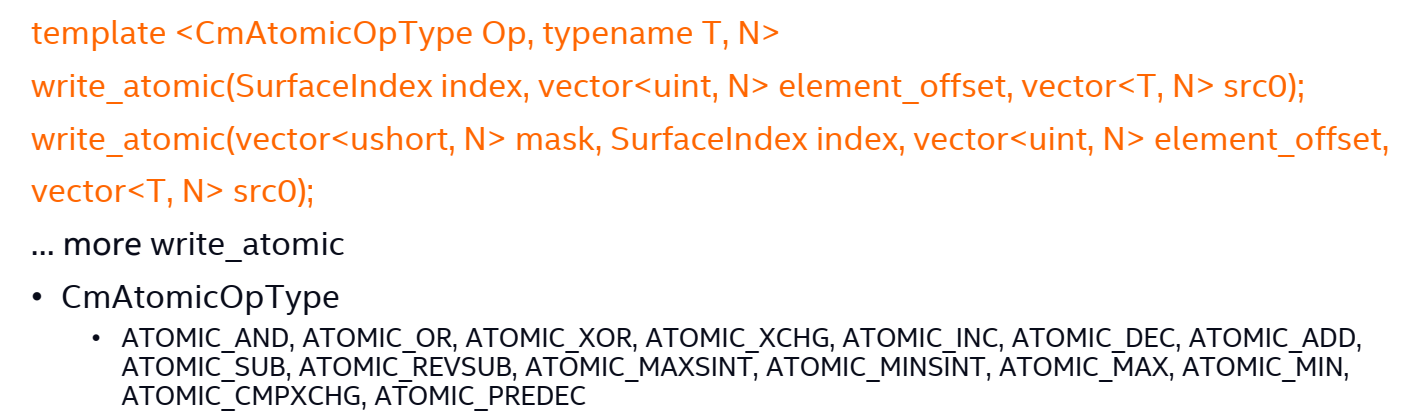


Scatter R/W

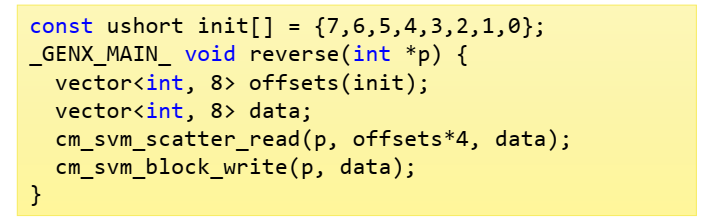
read(SurfaceIndex IND, uint global\_offset, vector<uint, N> element\_offset, vector\_ref<T, N> v);

write(SurfaceIndex IND, uint global\_offset, vector<uint, N> element\_offset, vector<T, N> v);

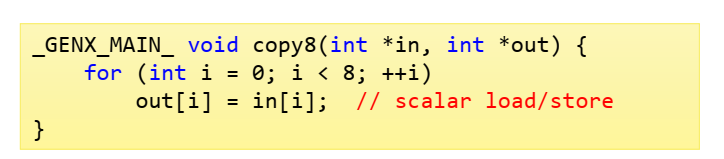
Atomic Write:



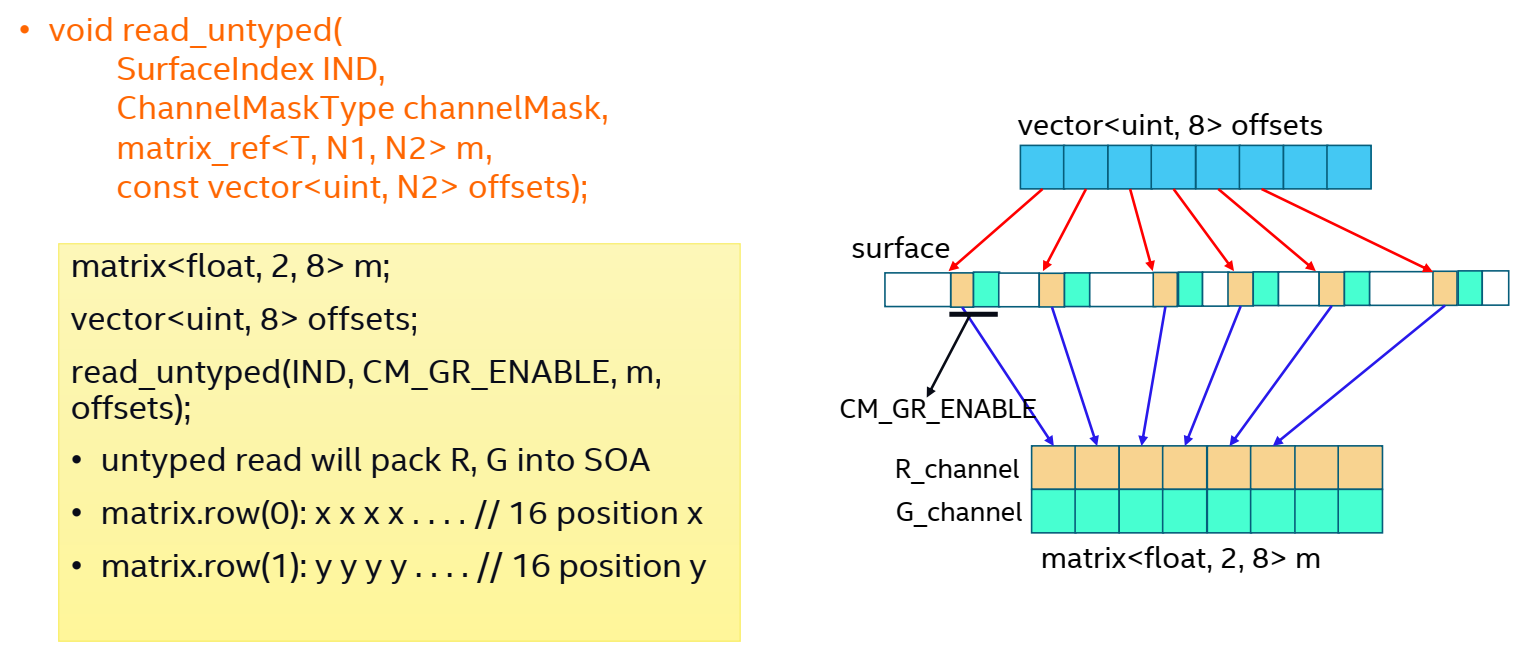
SVM R/W:

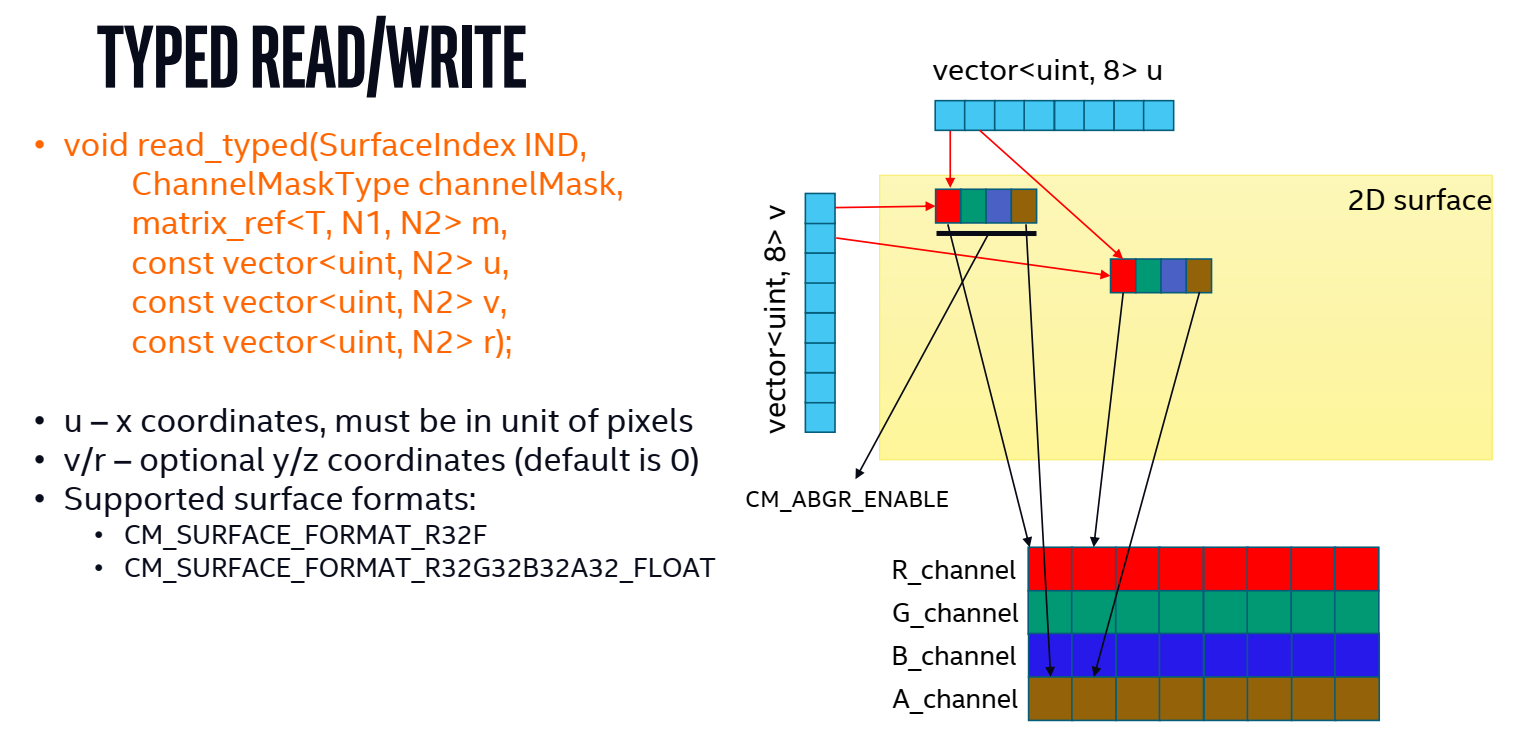


Scalar load/store

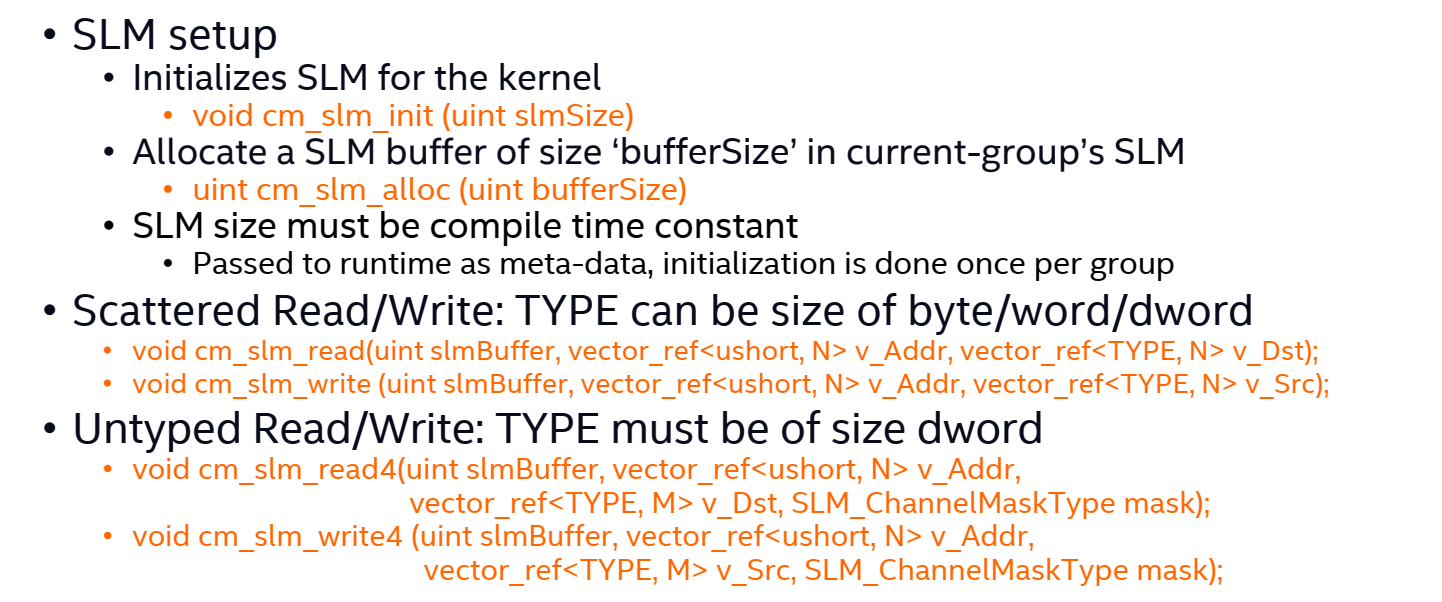


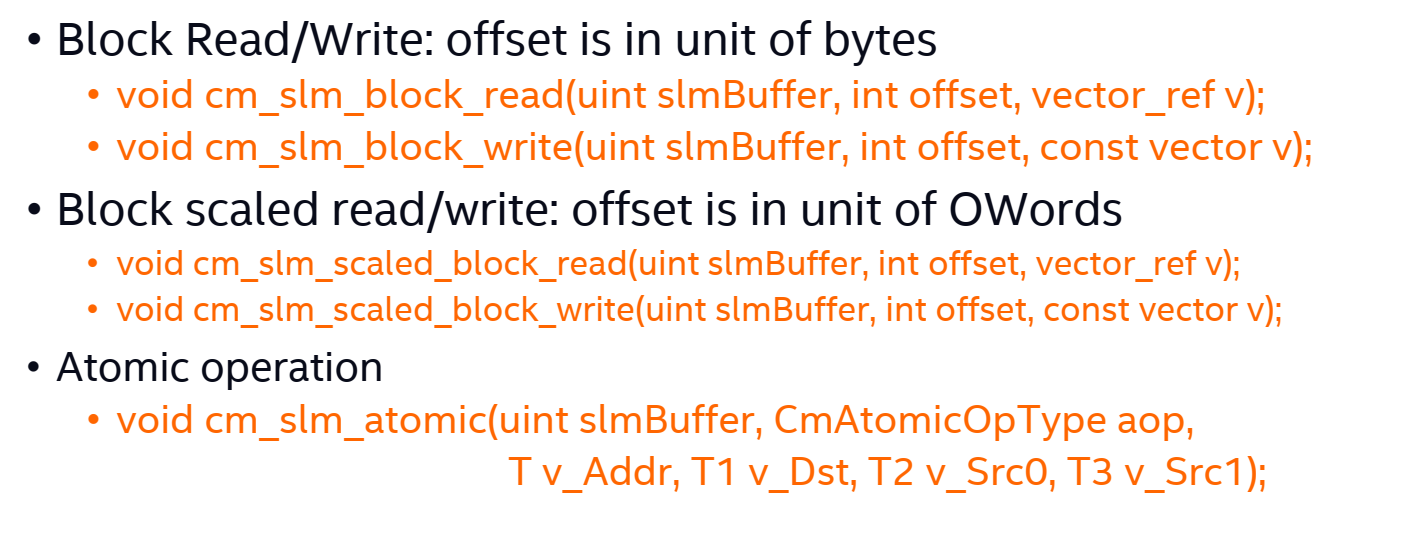
SOA R/W





SLM





1. 线程同步