Parallel Processing

Lecture : Roofline Model

邵恩
高性能计算机研究中心

主要内容来自：

塞缪尔•威廉姆斯（Samuel Williams）
计算研究部，劳伦斯伯克利国家实验室
Computational Research Division, Lawrence Berkeley National Lab

为什么要使用性能模型或工具?

- 识别性能瓶颈
- 激发软件优化方法
- **确定我们什么时候完成优化，确定还能够优化的上限在哪里。**
  - 评估与机器特定功能能够带来的性能效果
  - 激发对并行算法更改和优化的需求
- 预测未来机器（或体系结构）的性能
  - 对未来计算系统或计算节点的采购计划，设定一个比较现实的预期或期望
  - 用于硬件/软件协同设计，以确保未来的体系结构能够适合未来应用程序的计算需求.

计算复杂度

- 假设运行时间与操作数相关（例如FP ops）
- 用户确定了并行算法、解法器、核函数的各项参数
- 根据这些参数计算操作次数
- 证明运行时间与这些参数相关

```
#pragma omp parallel for
for(i=0;i<N;i++){
  Z[i] = alpha*X[i] + Y[i];
}
```

双精度线性代数运算 DAXPY: O(N)复杂度，其中N是元素的[illegible]量。

```
#pragma omp parallel for
for(i=0;i<N;i++){
for(j=0;j<N;j++){
  double Cij=0;
  for(k=0;k<N;k++){
    Cij += A[i][k] * B[k][j];
  }
  [illegible] = sum;
```

[illegible]阵乘 DGEMM: $O(N^3)$ [illegible]其中N是行数。

为什么我们不能达到理想缩放比例？

(ideal scaling)

快速傅[illegible]量相关

共轭梯度[illegible]

多重网格法 M[illegible]数相关

N体问题 N-body: $O(N^2)$ 与粒子数量相关（时间步长）

理想缩放（ideal scaling）

- 在并行算法中，“理想缩放”指的是当我们增加处理器数量时，算法的运行时间会成比例地减少。
  - 例如，如果我们将处理器数量增加一倍，那么理想情况下算法的运行时间应该减半。
- 但是，在实际应用中，我们通常无法达到理想缩放。
  - 这可能是由于多种原因
    - 通信开销
    - 负载不均衡
    - 内存限制

数据移动的复杂性

- 假设应用的执行时间与访问（或移动）的数据量相关
- 可以很容易的得出计算过程中，所需要访问的数据量……计算数组访问次数
- 移动的数据更复杂，因为它需要了解缓存行为……
  - 需要考虑**有限缓存容量**的对数据移动的影响（移动次数、竞争等）

Operation	计算复杂度 Flop’s	搬移数据的复杂度 Data
DAXPY	O(N)	O(N)
DGEMV	$O(N^2)$	$O(N^2)$
DGEMM	$O(N^3)$	$O(N^2)$
FFTs	O(NlogN)	O(N)
CG	$O(N^{1.33})$	
MG		
N-body		

哪个代价更高呢……

执行 Flop，或从内存中移动数据?

[1]Hill et al, “Evaluating Associativity in CPU Caches”, IEEE Trans. Comput., 1989.

机器平衡(Machine Balance)和运算强度(Arithmetic Intensity)

- 数据移动和计算可以以不同的速率进行
- 机器平衡（Machine Balance）定义为如下的比率：

$$\text{Balance} = \frac{\text{Peak DP Flop/s}}{\text{Peak Bandwidth}}$$

← 浮点运算次数

← 内存操作次数

Operation	Flop’s	Data	AI (ideal)
DAXPY	O(N)	O(N)	O(1)
DGEMV	O(N^2)	O(N^2)	O(1)
DGEMM	O(N^3)	O(	O(N)
FFTs	O(	O(	O(logN)
CG	O(	O(	O(
MG	O(	O(	O(
N-body	O(	O(	O(N)

算术强度和机器平衡更高的核函数，更受计算能力限制（扩展性更好）

- 运算强度（Arithmetic Intensity）定义为如下的比率：

$$\text{AI} = \frac{\text{Flop’s Performed}}{\text{Data Moved}}$$

← 浮点运算（FLOPs）次数

← 内存访问（Bytes）数

每字节浮点运算次数（F/B）

计算深度(Computational Depth)

- 顺序执行的CPU 会在内存不连续性和函数调用上产生延迟和开销
- 并行机器在同步（共享内存）、点对点通信、缩减和广播方面产生类似的开销
- 因此，我们可以按深度（算法依赖链的最大深度）对算法进行分类
  - 侧面体现一个核函数的非计算开销

Operation	Flop’s	Data	AI (ideal)	Depth
DAXPY	O(N)	O(N)		O(1)
DGEMV	$O(N^2)$	O(	)	O(logN)
DGEMM	O(	O	N)	O(logN)
FFTs	O(Nlo			O(logN)
CG	O($N^{1.3}$			
MG				O(logN)
N-body				O(logN)

分布式内存的性能建模

- 在分布式内存中，通过发送消息来完成处理器间的通信.

- 消息传递时间可能受到多方面因素影响....
  - 开销（指发送/接收消息过程中耗费的 CPU 时间片）
  - 延迟（指消息在网络中的通信延迟）
  - 消息吞吐量（指以小信息单位，发送小消息的速度……消息/秒）
  - 带宽（指发送大消息过程中，有效数据的传输吞吐量……GBytes/s）
- 带宽和延迟，主要受到互连网络的体系结构和通信拥塞的限制

- 根据 N（问题规模）和 P（处理器个数）值的不同，以上这些因素可能会对我们算法的分布式内存造成不同程度的影响

性能模型

- 许多不同的因素，可以影响核函数的执行时间.
  - 有些是应用程序的特性，
  - 有些是机器的特性，
  - 有些两者都是（内存访问模式+缓存）

#FP操作的次数	Flop/s
Cache数据移动的速率	Cache GB/s
DRAM 数据移动的速率	DRAM GB/s
PCIe 数据移动的速率	PCIe bandwidth
计算深度	OMP Overhead
MPI 消息大小（size）	Network Bandwidth
MPI 发送/等待时间的比率	Network Gap
#MPI 等待时间	Network Latency

性能模型

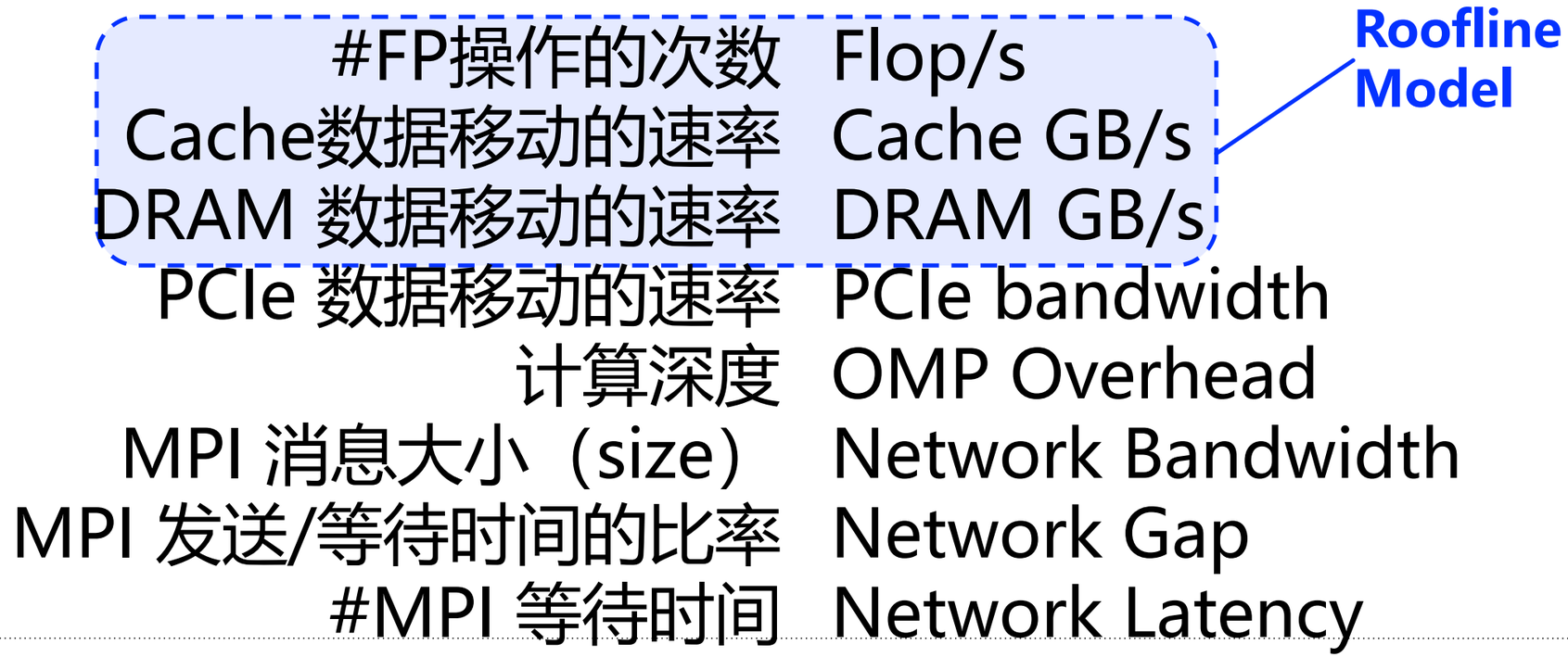
- 不能对每个应用程序把以下所有这些术语都考虑一遍

计算复杂度

#FP操作的次数	Flop/s
Cache数据移动的速率	Cache GB/s
DRAM 数据移动的速率	DRAM GB/s
PCIe 数据移动的速率	PCIe bandwidth
计算深度	OMP Overhead
MPI 消息大小（size）	Network Bandwidth
MPI 发送/等待时间的比率	Network Gap
#MPI 等待时间	Network Latency

性能模型

- 因为要考虑的因素太多，性能模型通常只考虑其中部分因素。



Williams et al, "Roofline: An Insightful Visual Performance Model For Multicore Architectures", CACM, 2009.

性能模型

- 因为要考虑的因素太多，性能模型通常只考虑其中部分因素。

#FP操作的次数	Flop/s
Cache数据移动的速率	Cache GB/s
DRAM 数据移动的速率	DRAM GB/s
PCIe 数据移动的速率	PCIe bandwidth
计算深度	OMP Overhead
MPI 消息大小（size）	Network Bandwidth
MPI 发送/等待时间的比率	Network Gap
#MPI 等待时间	Network Latency

LogP

Culler, et al, "LogP: a practical model of parallel computation", CACM, 1996.

性能模型

- 因为要考虑的因素太多，性能模型通常只考虑其中部分因素。

#FP操作的次数	Flop/s
Cache数据移动的速率	Cache GB/s
DRAM 数据移动的速率	DRAM GB/s
PCIe 数据移动的速率	PCIe bandwidth
计算深度	OMP Overhead
MPI 消息大小（size）	Network Bandwidth
MPI 发送/等待时间的比率	Network Gap
#MPI 等待时间	Network Latency

LogGP

Alexandrov, et al, "LogGP: incorporating long messages into the LogP model - one step closer towards a realistic model for parallel computation", SPAA, 1995.

性能模型

- 因为要考虑的因素太多，性能模型通常只考虑其中部分因素。

#FP操作的次数	Flop/s
Cache数据移动的速率	Cache GB/s
DRAM 数据移动的速率	DRAM GB/s
PCIe 数据移动的速率	PCIe band
计算深度	OMP C
MPI 消息大小（size）	Network
MPI 发送/等待时间的比率	Network Ga
#MPI 等待时间	Network Latency

LogCA

Bin Altaf et al, "LogCA: A High-Level Performance Model for Hardware Accelerators", ISCA, 2017.

Roofline Model简介

性能模型/模拟器

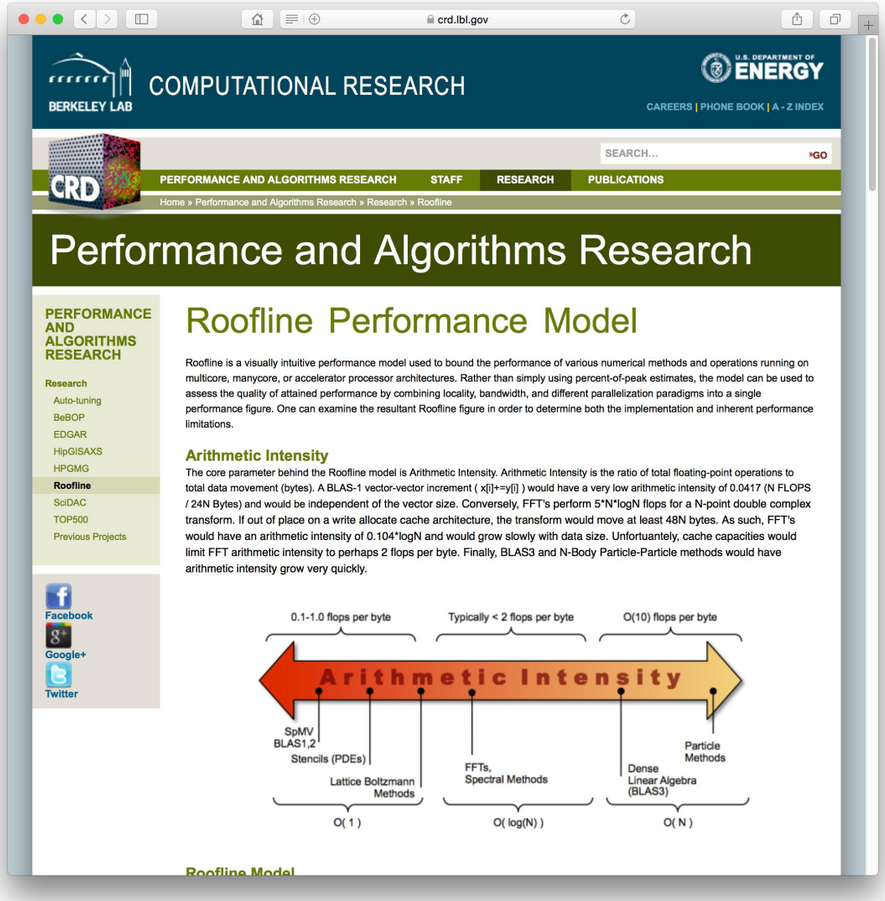
- 历史上，许多性能模型和模拟器，都通过统计和追踪计算过程的延迟，以预测性能（即：计数器的周期）

- 在过去的二十年里，出现了许多延迟隐藏技术……
  - 乱序执行（Out-of-order execution）：通过硬件提高并行性以隐藏延迟
  - 数据预取（prefetching ）：用硬件推测可能会加载数据，提前加载到快速缓存中
  - 大规模线程并行（Massive thread parallelism ）：(通过开启大量的独立线程，满足延迟-带宽条件)

- 有效的延迟隐藏技术，直接导致计算机从延迟受限，转变为**吞吐量受限（ throughput-limited computing)**

Roofline Model

- **Roofline Model**是一种面向吞吐量的性能模型……
  - 关注的操作的并发数量，即吞吐率。而不是单个操作或核函数的执行时间
  - 在Roofline Model中，计算平台的峰值性能代表了系统的最大并发能力
  - 而访存带宽限制则代表了系统在给定延迟下能够支持的最大并发量
  - 独立于 ISA 指令集和体系结构（适用于 CPU、GPU、Google TPUs1 等……）



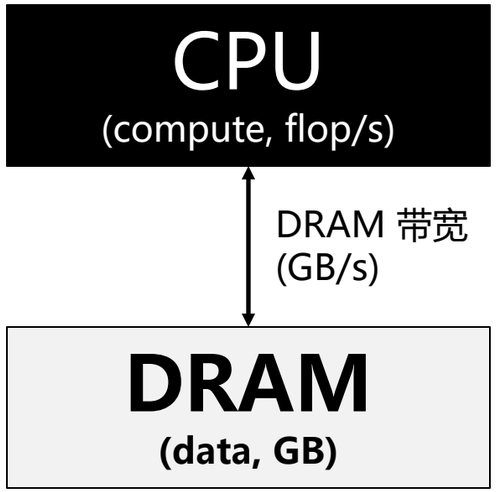
https://crd.lbl.gov/departments/computer-science/PAR/research/roofline

[1]Jouppi et al, "In-Datacenter Performance Analysis of a Tensor Processing Unit", ISCA, 2017.

(DRAM) Roofline

- 人们可能希望始终获得最佳计算性能 (即获得最大限度的Flop/s)
- 但是，有限的局部性（数据重用）和访存带宽的上限，限制了核函数能够达到的计算性能。

  .
- 假定：
  - 理想化的processor与caches
  - 冷启动时，想要的数据就在 DRAM中



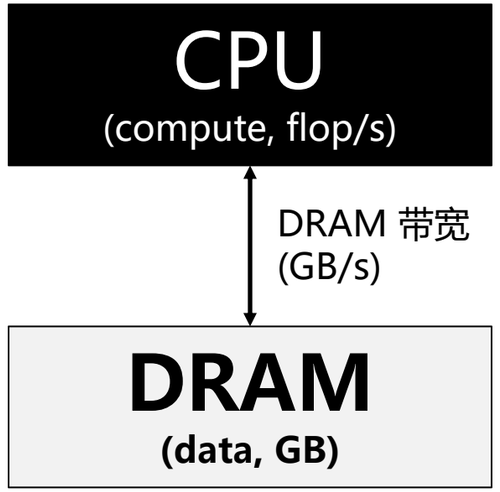
并发的浮点操作数　浮点数计算性能(峰值)

单个操作延迟时间（Time） = max { #FP ops / Peak GFlop/s ; #Bytes / Peak GB/s }

单个操作访存数据量　　访存带宽

(DRAM) Roofline

- 人们可能希望始终获得最佳计算性能 (即获得最大限度的Flop/s)
- 但是，有限的局部性（数据重用）和访存带宽的上限，限制了核函数能够达到的计算性能。
- 假定：
  - 理想化的processor与caches
  - 冷启动时，想要的数据就在 DRAM中



浮点数计算性能(峰值)

单个操作延迟时间

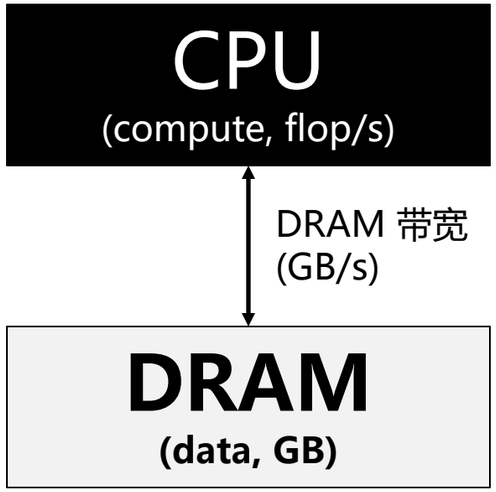
并发的浮点操作数

$$\frac{\text{Time}}{\#\text{FP ops}} = \max \begin{cases} 1 \,/\, \text{Peak GFlop/s} \\ (\#\text{Bytes} \,/\, \#\text{FP ops}) \,/\, \text{Peak GB/s} \end{cases}$$

单个操作访存数据量　并发的浮点操作数　访存带宽(峰值)

(DRAM) Roofline

- 人们可能希望始终获得最佳计算性能 (即获得最大限度的Flop/s)
- 但是，有限的局部性（数据重用）和访存带宽的上限，限制了核函数能够达到的计算性能。.
- 假定：
  - 理想化的processor与caches
  - 冷启动时，想要的数据就在 DRAM中



浮点数计算性能(峰值)

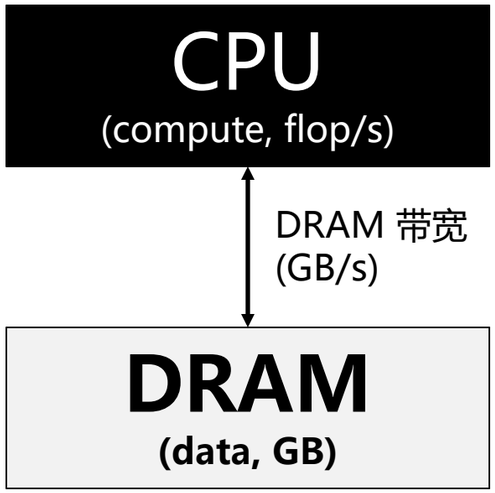
并发的浮点操作数 / 单个操作延迟时间 = #FP ops / Time = min { Peak GFlop/s ; (#FP ops / #Bytes) * Peak GB/s }

访存带宽(峰值)

并发的浮点操作数　　单个操作访存数据量

(DRAM) Roofline

- 人们可能希望始终获得最佳计算性能 (即获得最大限度的Flop/s)
- 但是，有限的局部性（数据重用）和访存带宽的上限，限制了核函数能够达到的计算性能。.
- 假定：
  - 理想化的processor与caches
  - 冷启动时，想要的数据就在 DRAM中



浮点数计算性能(实际)

GFlop/s = min

Peak GFlop/s

AI * Peak GB/s

浮点数计算性能(峰值)

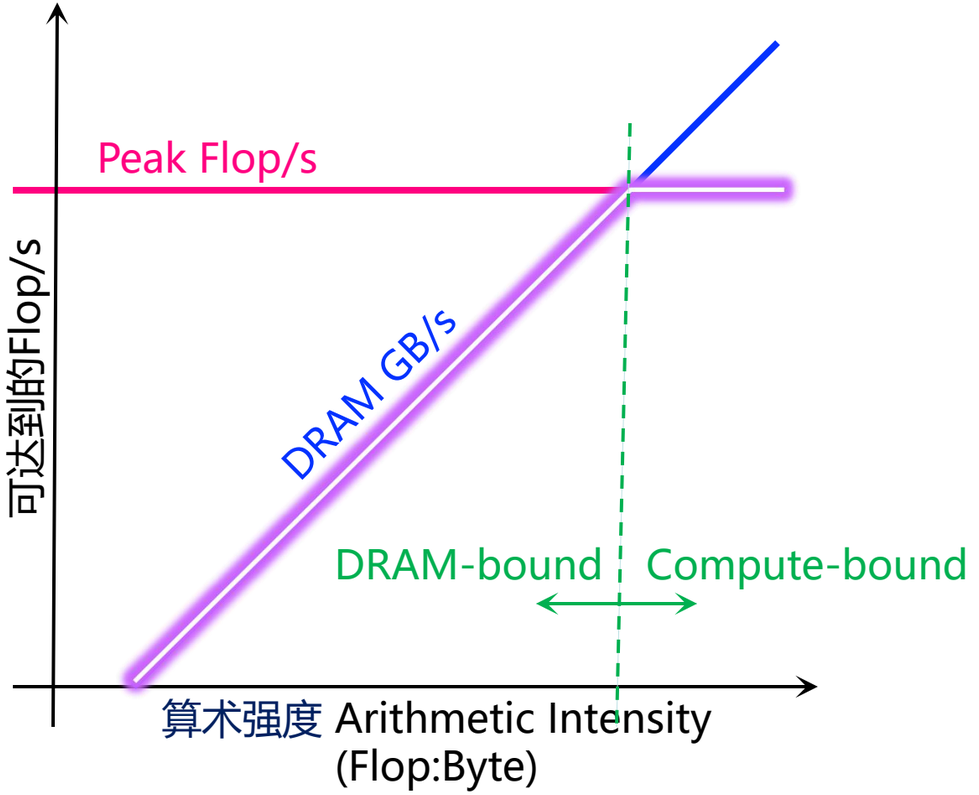
访存带宽(峰值)

算术强度

注： 算术强度 Arithmetic Intensity (AI) = Flops / Bytes (仅针对 DRAM)

(DRAM) Roofline

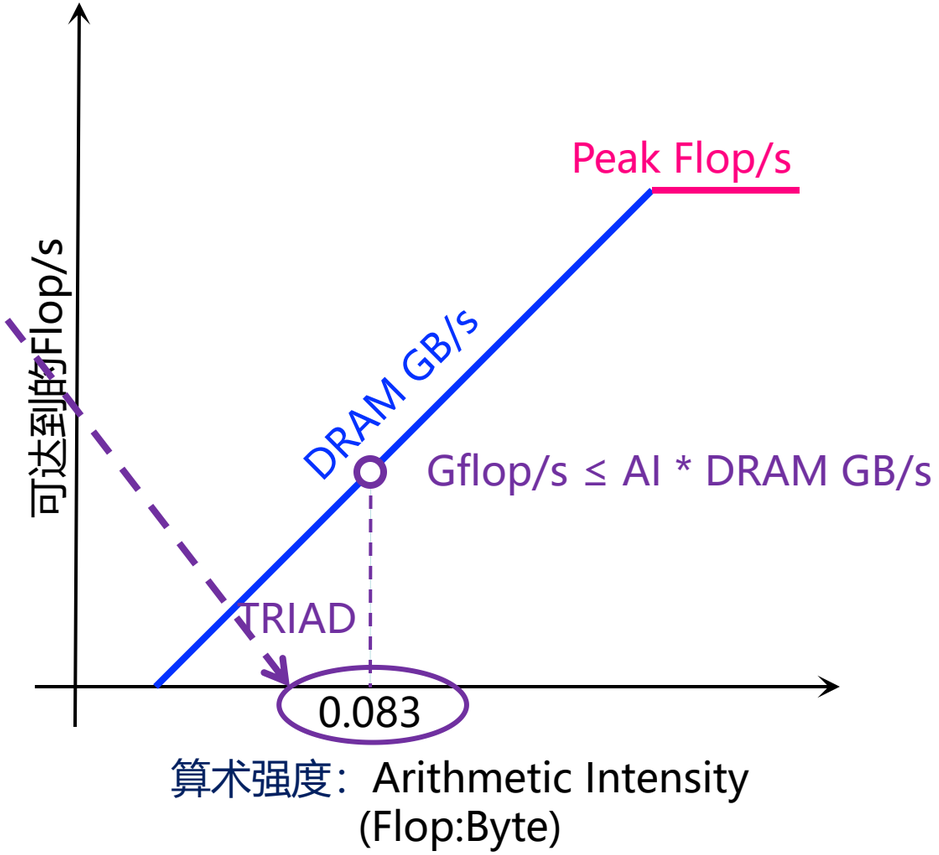
- 横轴：以算术强度（Arithmetic Intensity）作为 x 轴绘制 Roofline 边界
  - 即每字节内存访问所执行的浮点运算次数
- 纵轴:表示浮点性能
  - 即每秒浮点运算次数
- **对数-对数刻度（Log-log scale）** 使得不同数量级的数据能够在同一张图上清晰地显示
- 算术强度低于机器平衡的核函数，会受限于DRAM的访存带宽限制实际计算性能（我们稍后会对此进行细化）。



注：机器平衡（Machine Balance）是指计算平台的计算能力与内存带宽之间的比值。

Roofline 例子 #1——STREAM基准测试（Triad）

- 典型的机器平衡是每字节5-10次浮点运算
  - 要利用计算能力，每个双精度数需要40-80次浮点运算。
  - 这是技术和金钱的产物
  - **不太可能改善**



- STREAM基准测试中的一项测试：Triad…

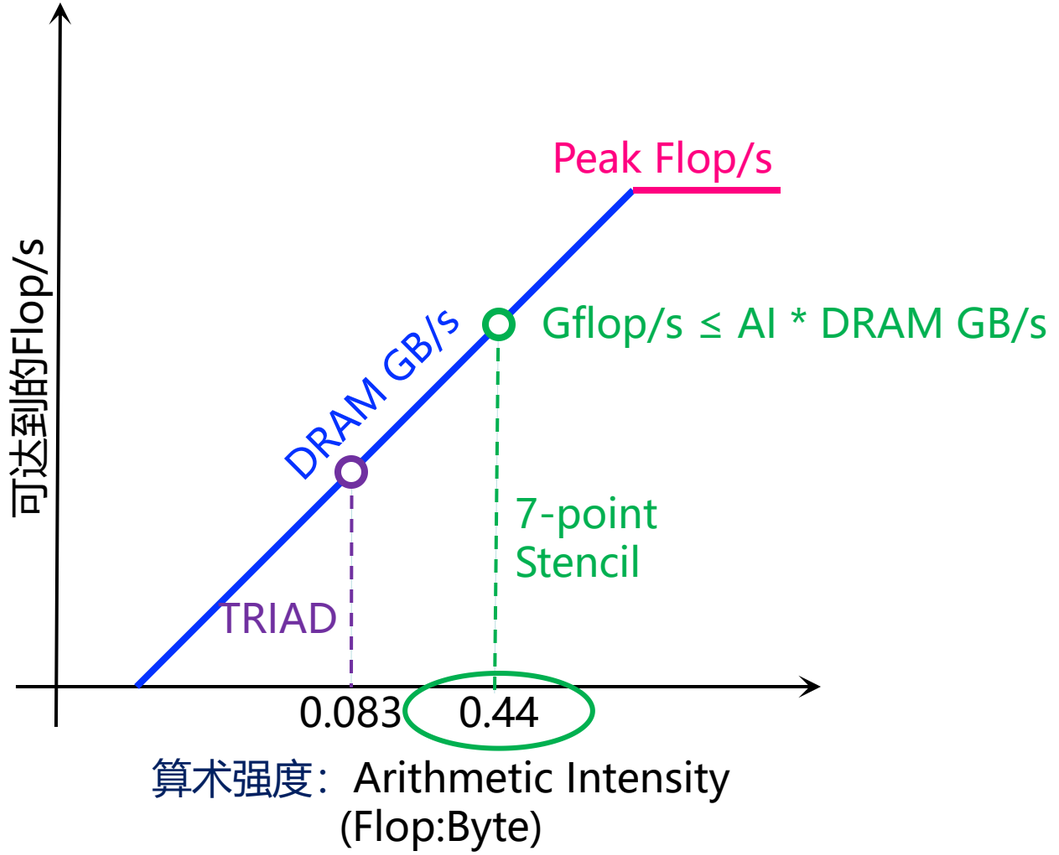
```
#pragma omp parallel for
for(i=0;i<N;i++){
  Z[i] = X[i] + alpha*Y[i];
}
```

  - 每次迭代需要进行2次浮点运算
  - 每次迭代传输24字节（读取X[i]，Y[i]，写入Z[i]）
  - **算术强度=每字节0.083次浮点运算==内存带宽受限**

Roofline 例子 #2——7-point constant coefficient stencil

- 相反的, 在7点常系数模板（7-point constant coefficient stencil）中...
  - 每次迭代需要进行7次浮点运算
  - 每个点需要进行8次内存引用（7次读取，1次存储）
  - Cache可以过滤掉每点除了1次读取和1次写入之外的所有操作
  - **算术强度=每字节0.44次浮点运算==内存受限,**
  - **但浮点运算速率是5倍**

```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  int ijk = i + j*jStride + k*kStride;
  new[ijk] = -6.0*old[ijk        ]
             + old[ijk-1       ]
             + old[ijk+1       ]
             + old[ijk-jStride]
             + old[ijk+jStride]
             + old[ijk-kStride]
             + old[ijk+kStride];
}}}
```

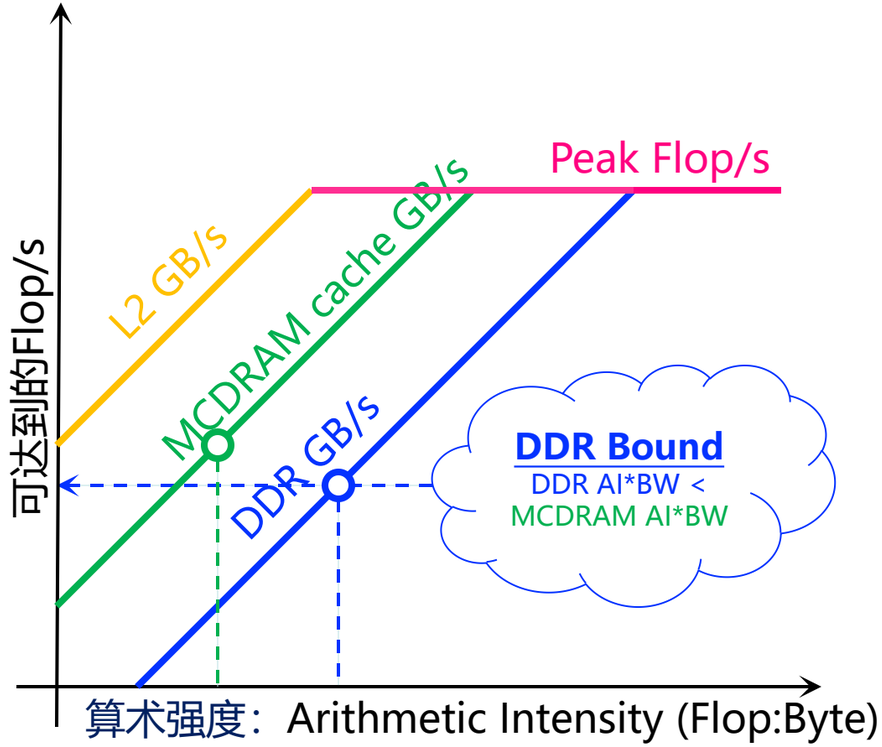


基于多级缓存层次的Roofline

- 真正的处理器有多个内存层次
  - 寄存器（Registers）
  - L1, L2, L3 cache
  - MCDRAM/HBM (KNL/GPU 设备内存、片上显存)
  - DDR (主存储器)
  - NVRAM (非易失性存储器)
- 应用程序可以在每个缓存层次中发挥局部性优化（locality）
  - 唯一的数据移动意味着唯一的算术强度（Arithmetic Intensity）
  - 此外，每个缓存级别都将具有独特的访存带宽

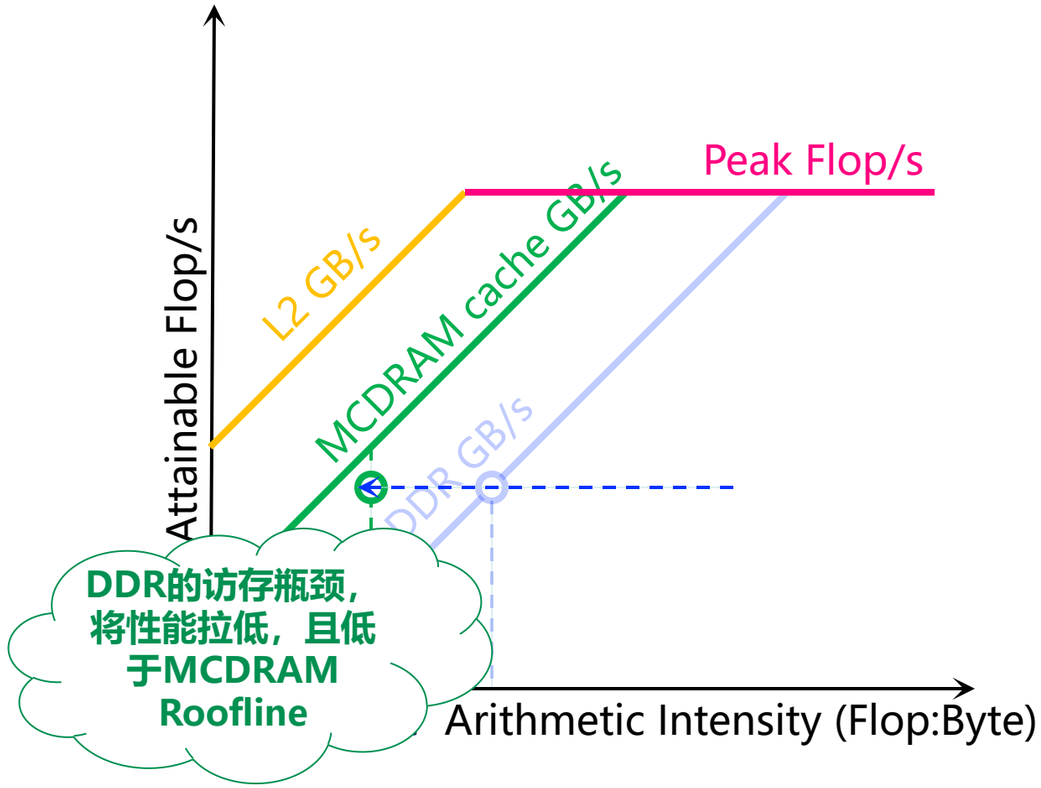
基于多级缓存层次的Roofline

- 构建出多条 Rooflines的叠加(黄、绿、蓝)
  - 测量各个缓存层次的访存带宽
  - 测量各个缓存层次的算术强度
  - 尽管循环嵌套可能具有多个算术强度和多个边界（如：flops，L1，L2，... DRAM）...
  - **... 但其实实际浮点性能，仅受最小值限制**
    - **最小的：算术强度(AI) * 访存带宽(BW)**
    - **DDR AI*BW < MCDRAM AI*BW**
  - **所以，如右图中是DDR Bound**



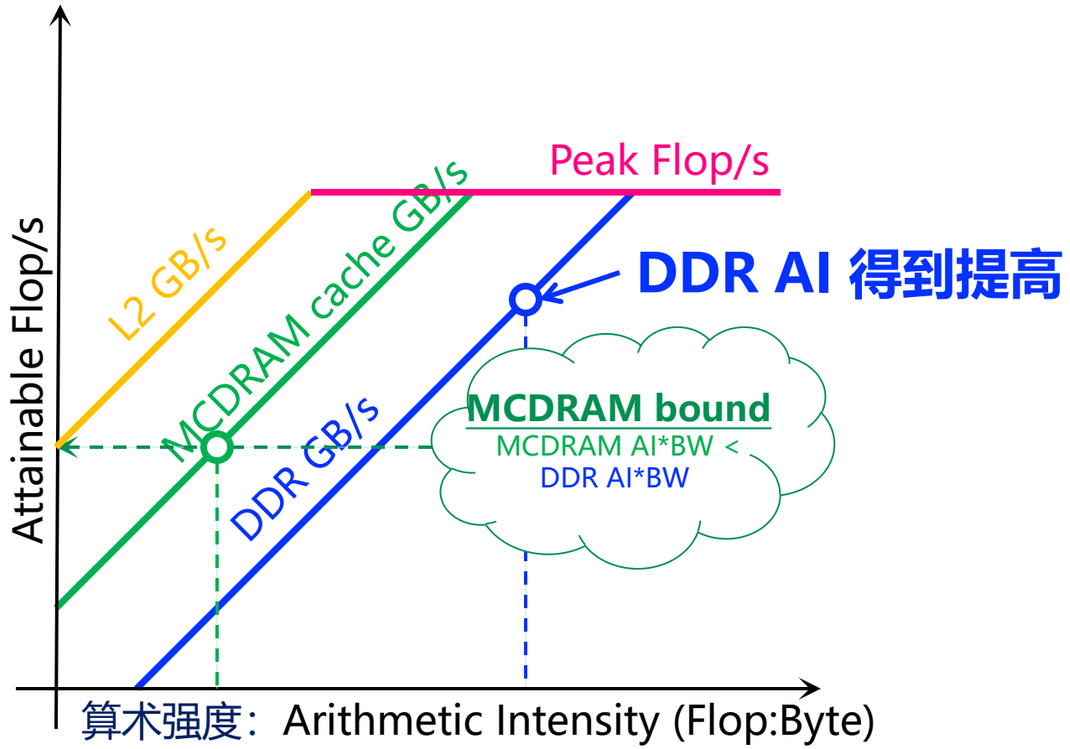
基于多级缓存层次的Roofline

- 构建出多条 Rooflines的叠加(黄、绿、蓝)
  - 测量各个缓存层次的访存带宽
  - 测量各个缓存层次的算术强度
  - 尽管循环嵌套可能具有多个算术强度和多个边界（如：flops，L1，L2，... DRAM）...
  - **... 但其实实际浮点性能，仅受最小值限制**
    - **最小的：算术强度(AI) * 访存带宽(BW)**
    - **DDR AI*BW < MCDRAM AI*BW**
  - **所以，如右图中是DDR Bound**



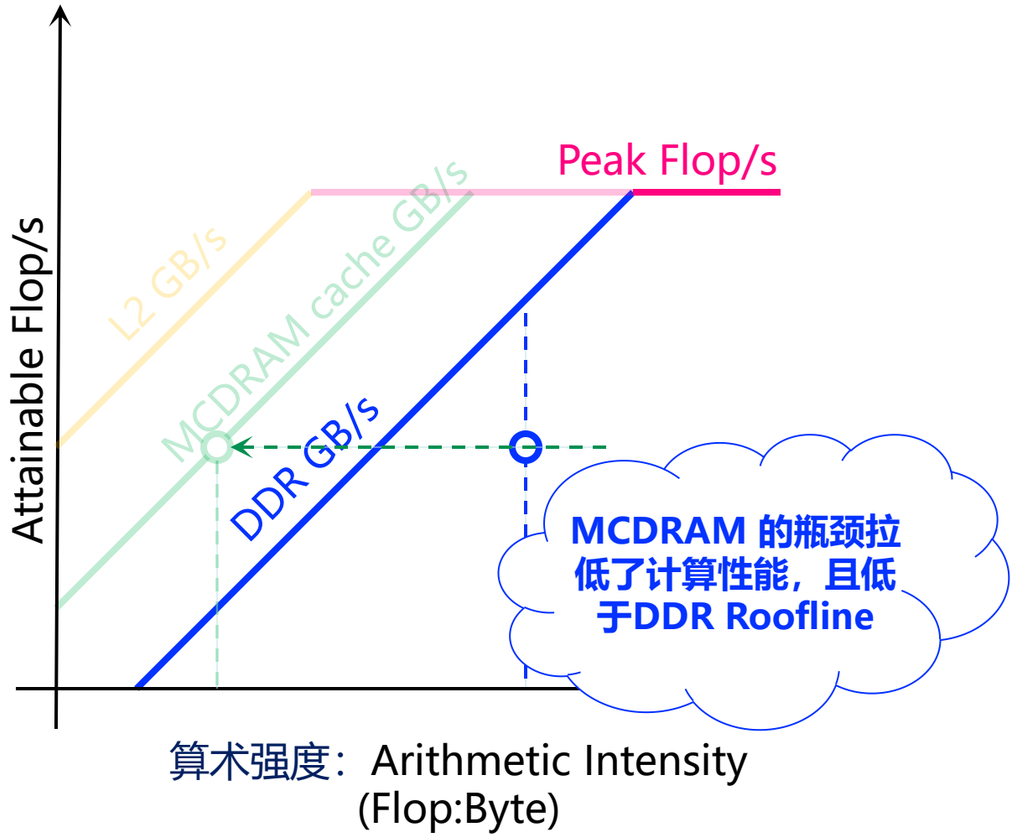
基于多级缓存层次的Roofline

- 构建出多条 Rooflines的叠加(黄、绿、蓝)
  - 测量各个缓存层次的访存带宽
  - 测量各个缓存层次的算术强度
  - 尽管循环嵌套可能具有多个算术强度和多个边界（如：flops，L1，L2，... DRAM）...
  - **... 但其实实际浮点性能，仅受最小值限制**
    - **最小的：算术强度(AI) * 访存带宽(BW)**
    - **MCDRAM AI*BW < DDR AI*BW**
    - **所以，如右图中是MCDRAM Bound**



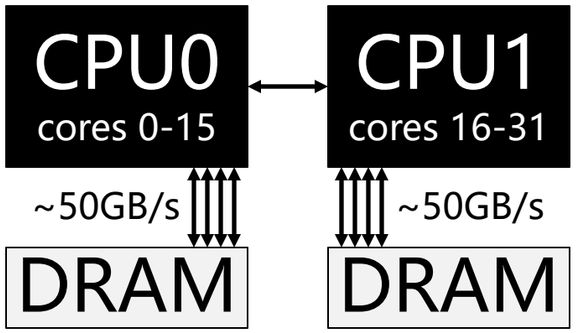
基于多级缓存层次的Roofline

- 构建出多条 Rooflines的叠加(黄、绿、蓝)
  - 测量各个缓存层次的访存带宽
  - 测量各个缓存层次的算术强度
  - 尽管循环嵌套可能具有多个算术强度和多个边界（如：flops，L1，L2，... DRAM）...
  - **... 但其实实际浮点性能，仅受最小值限制**
    - **最小的：算术强度(AI) * 访存带宽(BW)**
    - **DDR AI*BW < MCDRAM AI*BW**
  - **所以，如右图中是DDR Bound**

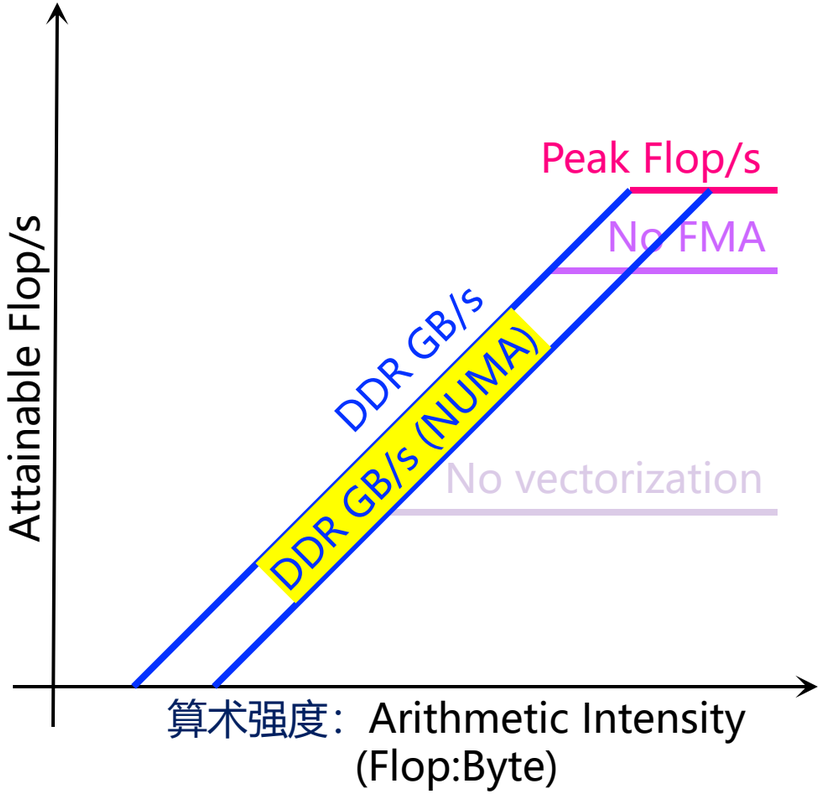


NUMA效应

- Cori的Haswell节点由2个Xeon处理器构建
  - 每个Xeon处理器都附有访存快速的本地内存
  - 允许通过互连网络，进行远程内存访问（访存慢速）
  - 不当的内存分配可能导致超过2倍的性能损失。

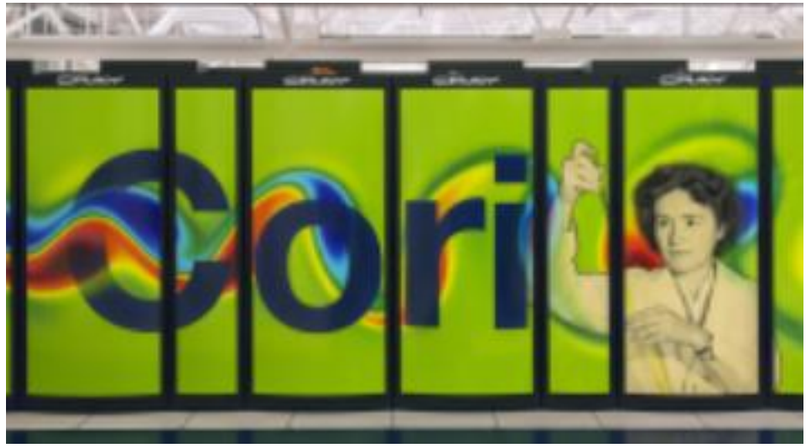


NUMA结构



Cori超算系统

- Cori是美国国家能源研究科学计算中心(National Energy Research Scientific Computing Center, NERSC)的一台超级计算机，
- 以荣誉生物化学家Gerty Cori命名，是第一个获得诺贝尔奖的美国女性
- Cori是一台Cray XC40系统，拥有622,336个Intel处理器核心，理论峰值性能为30 petaflop/s（每秒30千万亿次运算）1。



Cori超算系统

Cray XC40 supercomputing system

对核内(In-Core)性能的建模

数据、指令、线程级并行……

- 现代 CPU 使用多种技术来增加每个core的 Flop/s性能

融合乘加指令

- w = x*y + z 是线性代数中的常用表达式
- 处理器可以(FMA)使用多个指令
- FPU 将接到单个流水线中，以便完成 FMA/循环
- FMA/cycle 是指每个周期内的融合乘加（Fused Multiply-Add）操作
- FPU：浮点运算单元

Tensor Cores, QFMA, etc…

向量指令

- 许多 HPC 代码对向量元素执行相同的操作
- 半导体公司提供的向量指令，可对对2、4、8、16个元素执行相同的操作

……

例如：

x [0:7] *y [0:7] + z [0:7]

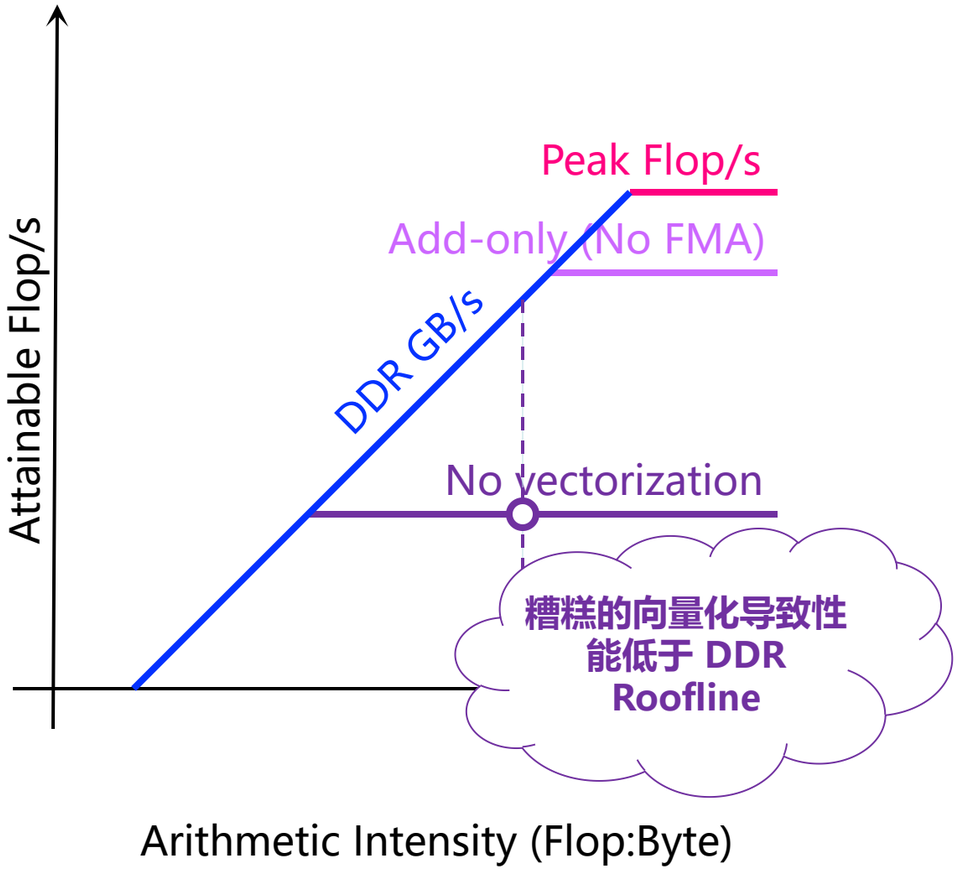
- 向量 FPU 完成 8 个向量运算/周期

深度流水线

- 执行FMA 的硬件至关重要
- 将单个 FMA 分解为几个较小的操作并对它们进行流水线化，由此让芯片厂商提高 GHz

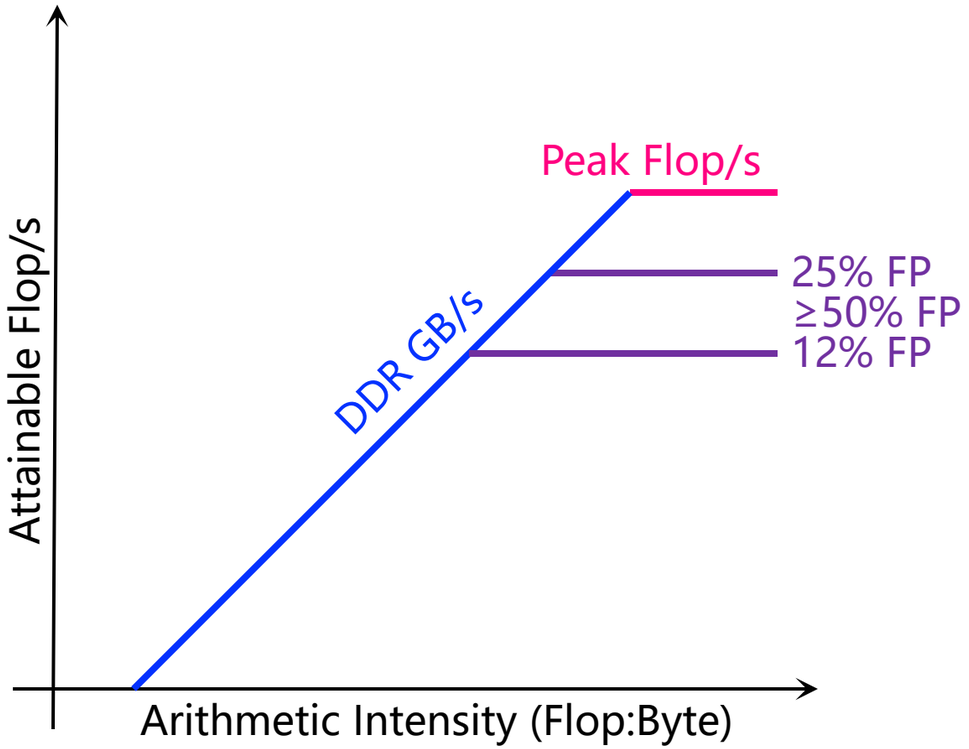
数据、指令、线程级并行……

- 如果每条指令都是 ADD（而不是 FMA），**性能将在 KNL 上下降 2 倍或在 Haswell 上下降 4 倍**
- 同样，如果没有矢量指令，**性能将在 KNL 上再下降 8 倍，在 Haswell 上下降 4 倍**
- 浮点（Floating Point）运算将会更糟
- 缺少线程将使 KNL 上的性能降低 64 倍。



超标量 vs. 混合指令

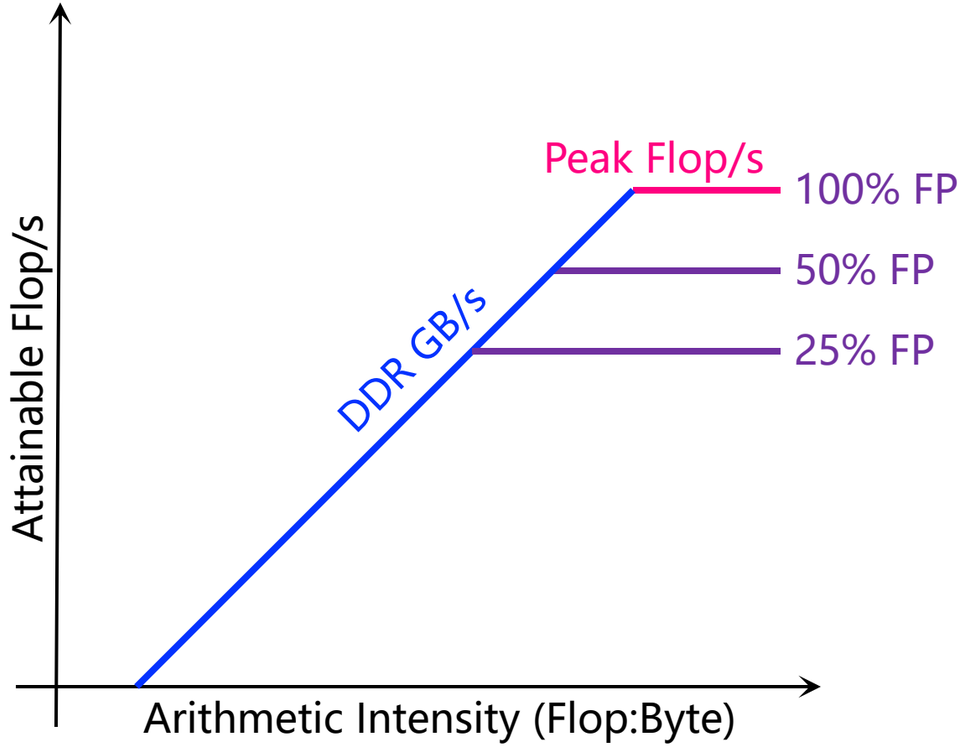
- 根据指令的组合确定计算核心数量的上限……
- e.g. Haswell(英特尔开发的一种处理器微架构，2013)
  - 4-issue superscalar
    - 处理器能够在每个时钟周期内发射（issue）最多 4 条指令到执行单元进行执行。
  - Only 2 FP data paths
    - 只有两条浮点（FP）数据路径
    - 这意味着处理器在每个时钟周期内最多只能同时执行两条浮点指令。



- 要达到 Haswell 处理器的最高性能，程序中至少需要有 50% 的指令是浮点指令。
- 因为 Haswell 处理器拥有两条浮点数据路径，能够在每个时钟周期内同时执行两条浮点指令。
- 如果程序中浮点指令的比例低于 50%，则处理器的浮点执行单元可能无法得到充分利用，从而无法达到最高性能。

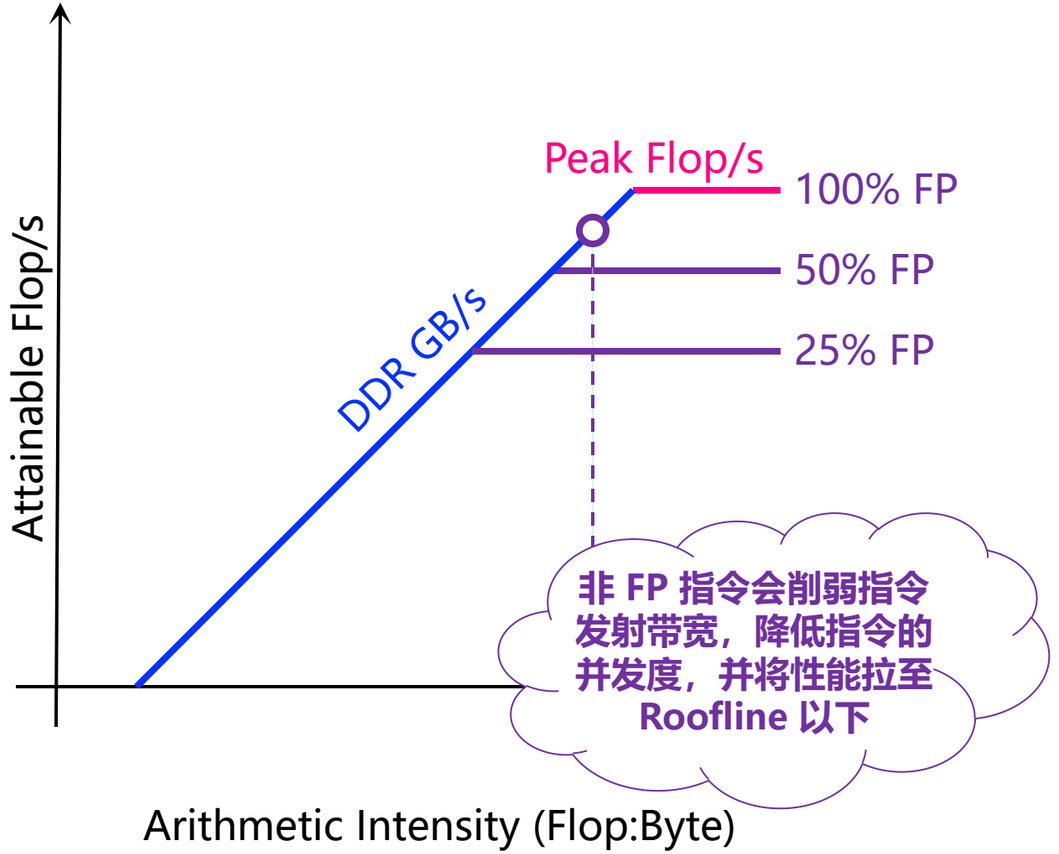
超标量 vs. 混合指令

- 根据指令的组合确定计算核心数量的上限……
- e.g. KNL
  - 2-issue superscalar
    - 处理器能够在每个时钟周期内发射（issue）最多 2条指令到执行单元进行执行。
  - 2 FP data paths
    - 有两条浮点（FP）数据路径
    - 这意味着处理器在每个时钟周期内能同时执行两条浮点指令。
  - 要达到 KNL处理器的最高性能，程序中至少需要有 100% 的指令是浮点指令。



超标量 vs. 混合指令

- 根据指令的组合确定计算核心数量的上限……
- e.g. KNL
  - 2-issue superscalar
    - 处理器能够在每个时钟周期内发射（issue）最多 2条指令到执行单元进行执行。
  - 2 FP data paths
    - 有两条浮点（FP）数据路径
    - 这意味着处理器在每个时钟周期内能同时执行两条浮点指令。
  - 要达到 KNL处理器的最高性能，程序中至少需要有 100% 的指令是浮点指令。

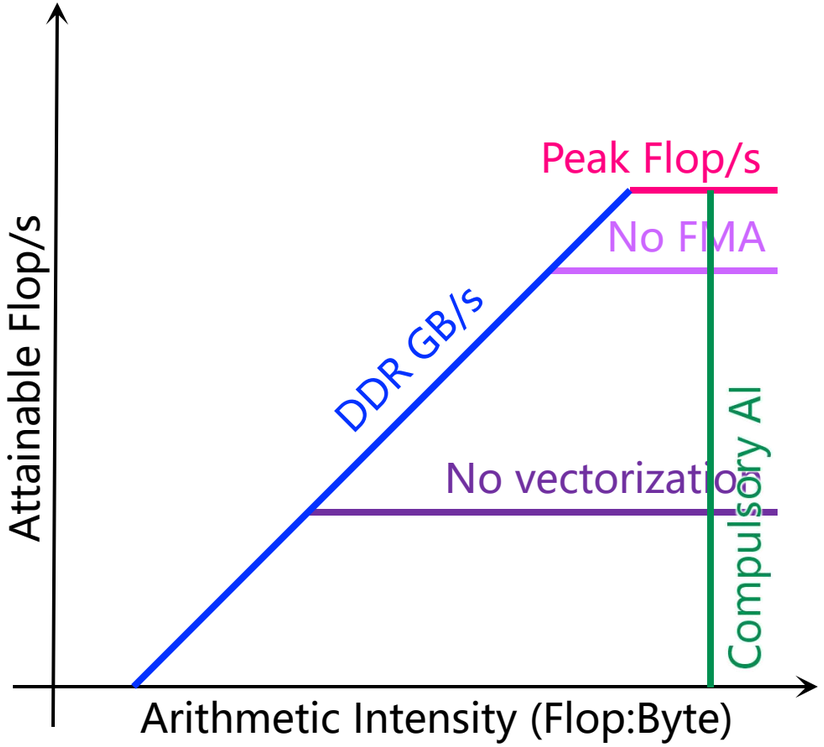


对缓存性能进行建模

局部性访存墙（Locality Walls）

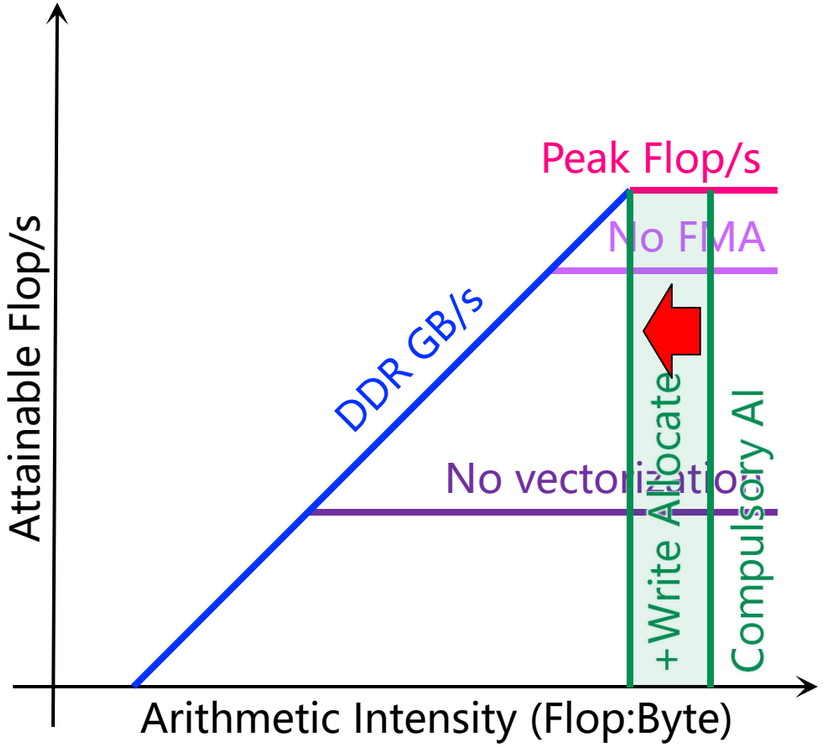
- 可以通过计算程序中必然会发生的缓存未命中（即强制缓存未命中）来估计程序的算法强度（AI）
- 算法强度：Arithmetic Intensity (Flop:Byte)
  - 指每字节访问内存所执行的浮点运算次数。这种估计方法比较简单，但可能不够准确。

$$AI = \frac{\#Flop's}{强制性cache\ miss}$$



局部性访存墙（Locality Walls）

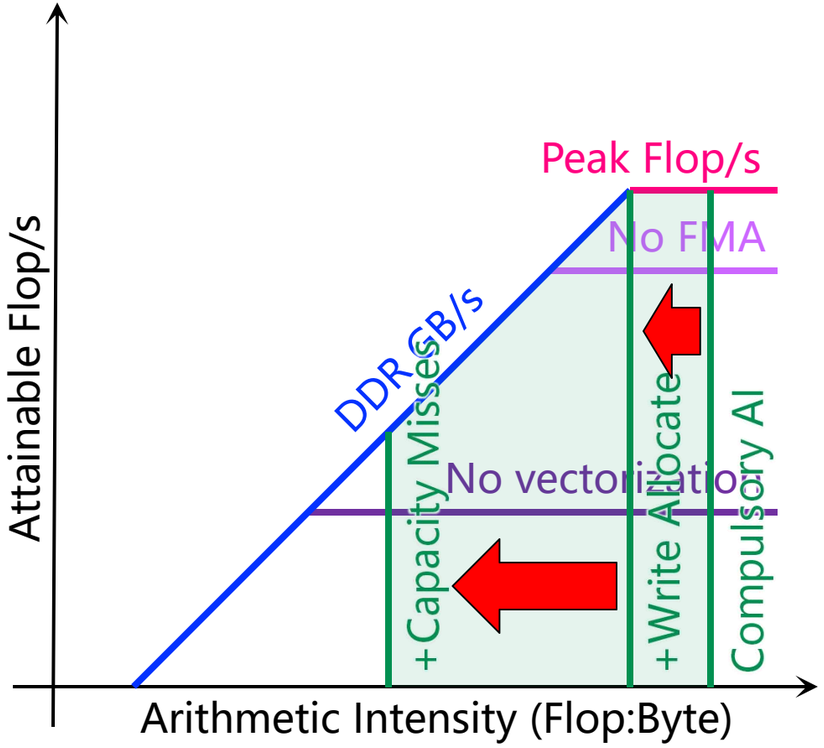
- 写分配缓存（write allocate caches）是一种缓存策略，当处理器向内存中的某个位置写入数据时，如果该位置不在缓存中，则会将其加载到缓存中。
- 这种策略可以提高程序的局部性，但也可能导致缓存中的其他有用数据被替换出去
- 因此，写分配缓存会引入**额外的写访存操作**，导致算术强度被降低



$$AI = \frac{\#Flop's}{\text{Compulsory Misses} + \textbf{Write Allocates}}$$

局部性访存墙（Locality Walls）

- 当程序访问的数据量超过缓存的容量时，就会发生缓存容量未命中（Cache capacity misses）。
- 这种情况下，缓存中的数据会被不断替换，导致缓存未命中率增加，从而降低程序性能。
- 缓存容量未命中与强制缓存未命中不同
  - 强制缓存未命中是指程序第一次访问某个数据时发生的缓存未命中，这是不可避免的
  - 缓存容量未命中是由于程序访问的数据量超过了缓存容量所导致的。通过优化程序的局部性，可以减少缓存容量未命中的发生。



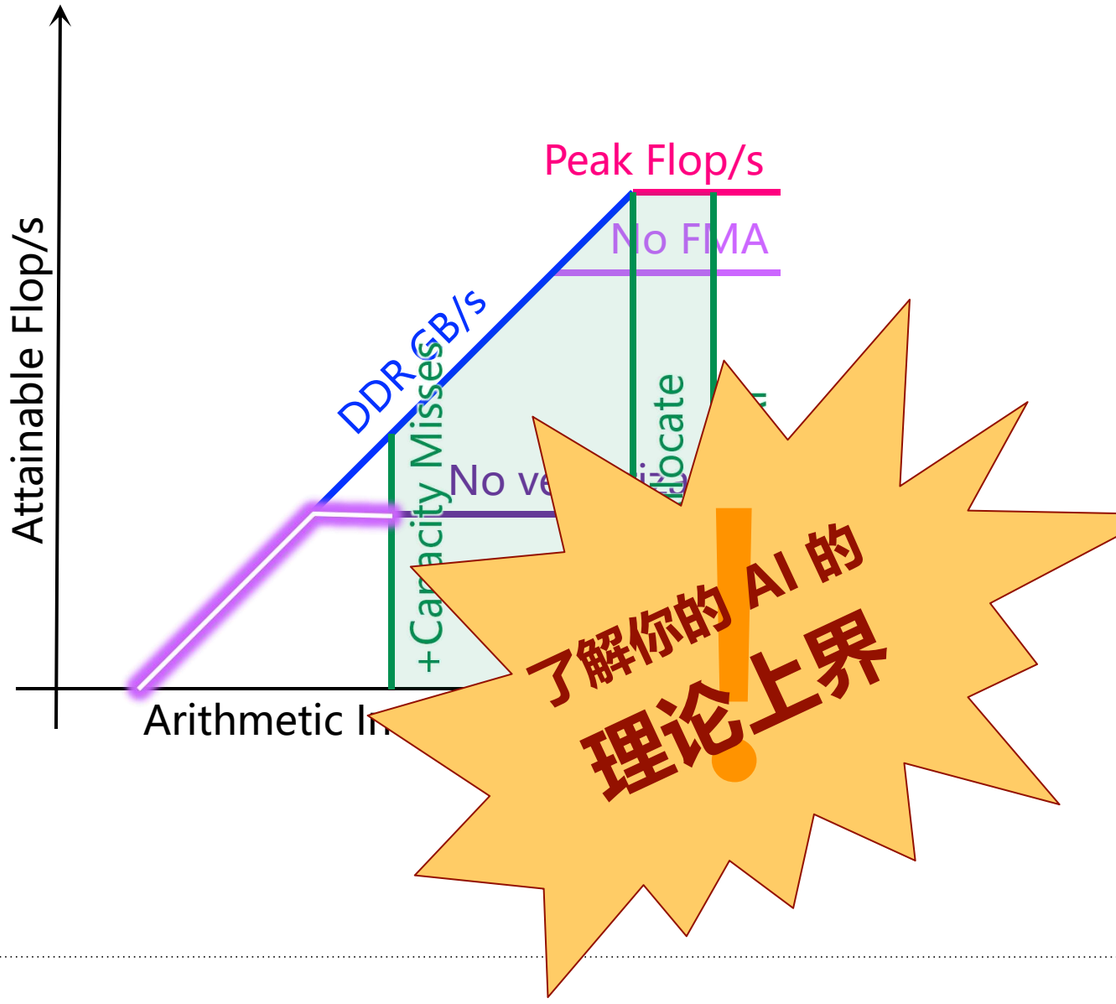
$$AI = \frac{\#Flop's}{Compulsory\ Misses + Write\ Allocates + \textbf{Capacity Misses}}$$

局部性访存墙（Locality Walls）

- 通过计算程序中必然会发生的强制缓存未命中，来估计程序的算法强度
- 写分配缓存会引入额外的写访存操作，导致算术强度被降低
- 缓存中的数据会被不断替换，导致缓存未命中率增加，从而降低程序性能

➢ **Compute bound 变成了memory bound**

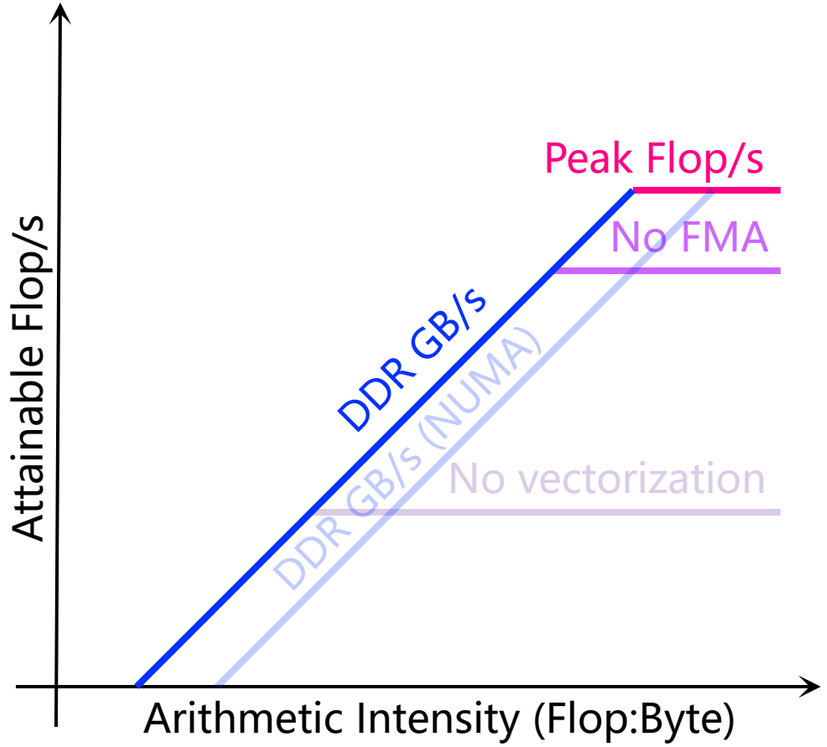
$$AI = \frac{\#Flop's}{Compulsory\ Misses + Write\ Allocates + Capacity\ Misses}$$



通用优化指南

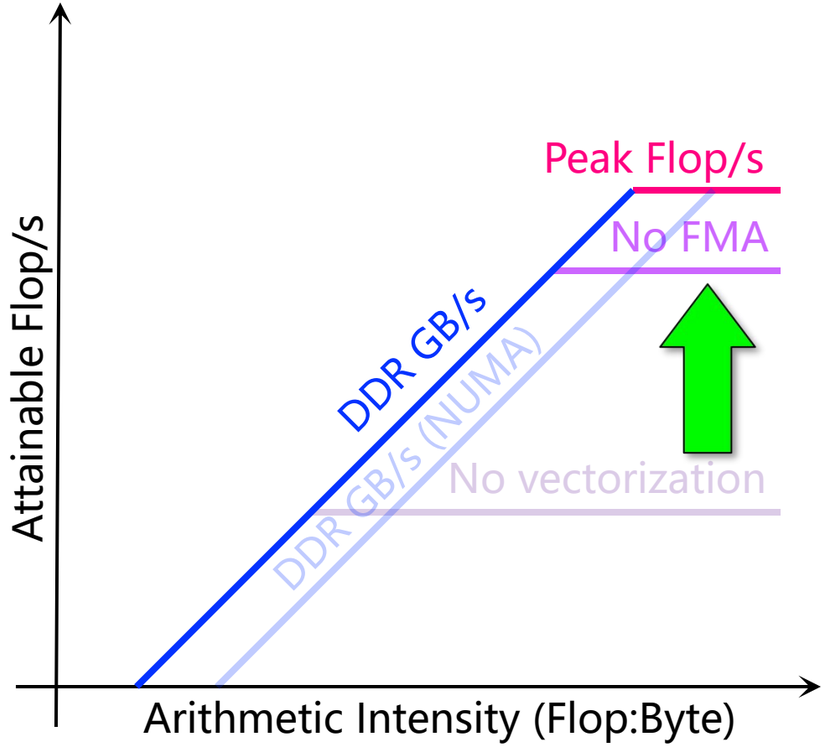
通用优化指南

- 从广义上讲，可以通过三种方法来提高性能：



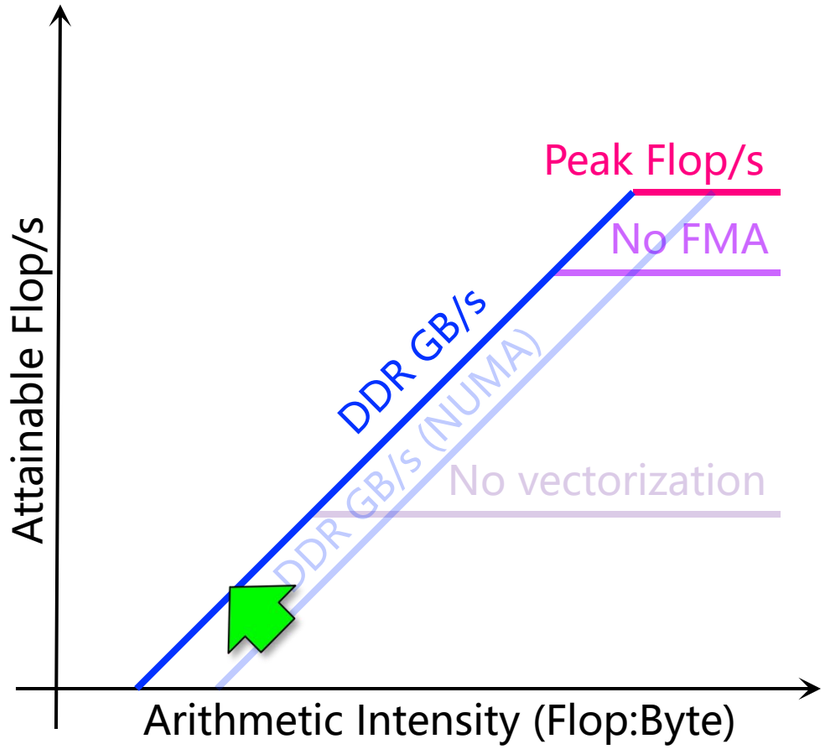
通用优化指南

- 从广义上讲，可以通过三种方法来提高性能：
- **最大限度地提高内核性能（例如让编译器进行矢量化）**



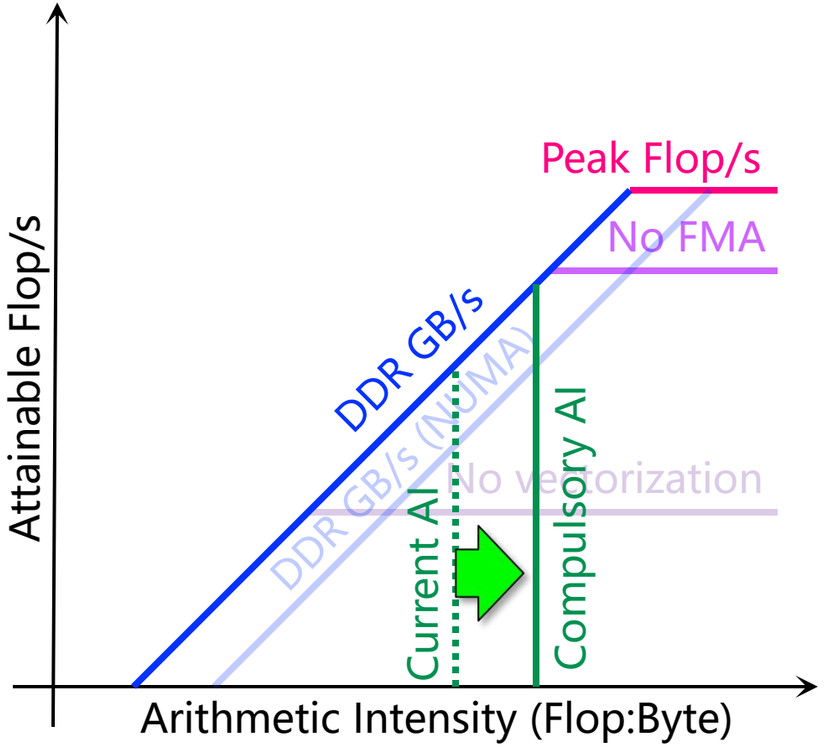
通用优化指南

- 从广义上讲，可以通过三种方法来提高性能：
- 最大限度地提高内核性能（例如让编译器进行矢量化）
- **最大化内存带宽（例如 NUMA-aware 分配）**



通用优化指南

- 从广义上讲，可以通过三种方法来提高性能：
- 最大限度地提高内核性能（例如让编译器进行矢量化）
- 最大化内存带宽（例如 NUMA-aware 分配）
- **最小化数据移动（增加 AI）**



构建屋Roofline Model还需要回答一些问题……

这些问题往往会让调优师不知所措……

目标计算机的属性

(基准测试)

我机器的峰值计算能力是多少?

向量化或 FMA 在我的机器上有多重要?

我机器的 DDR 访存带宽是多少 GB/s?

L2访存带宽?

应用程序的执行属性

(插装检测)

我的Kernel函数实际搬移了多少数据?

我的Kernel函数

我的Kernel实际上做了多少次flop运算?

访存和计算的鸿沟有多严重?

受硬件限制的kernel属性

(理论)

的强制性算术强度是多少?

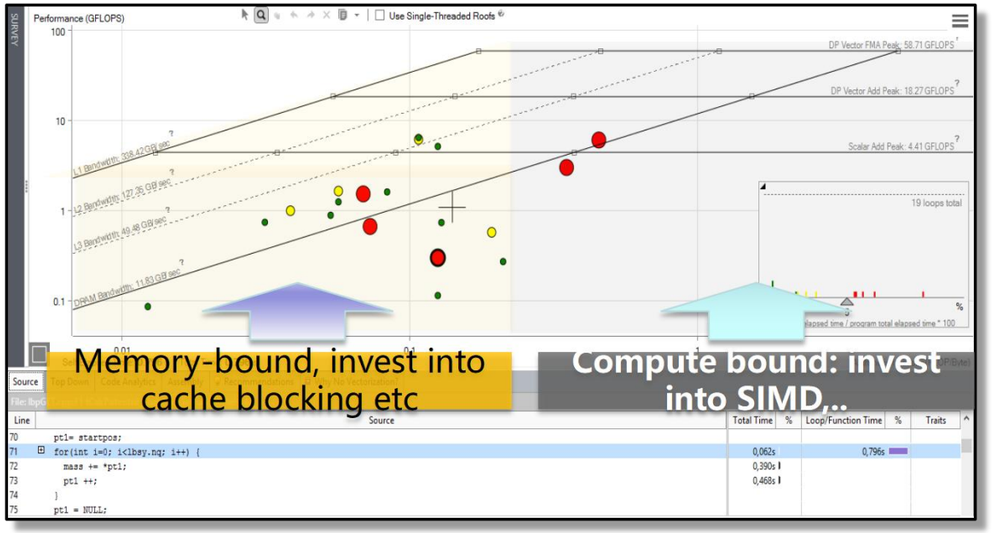
(通信下界)

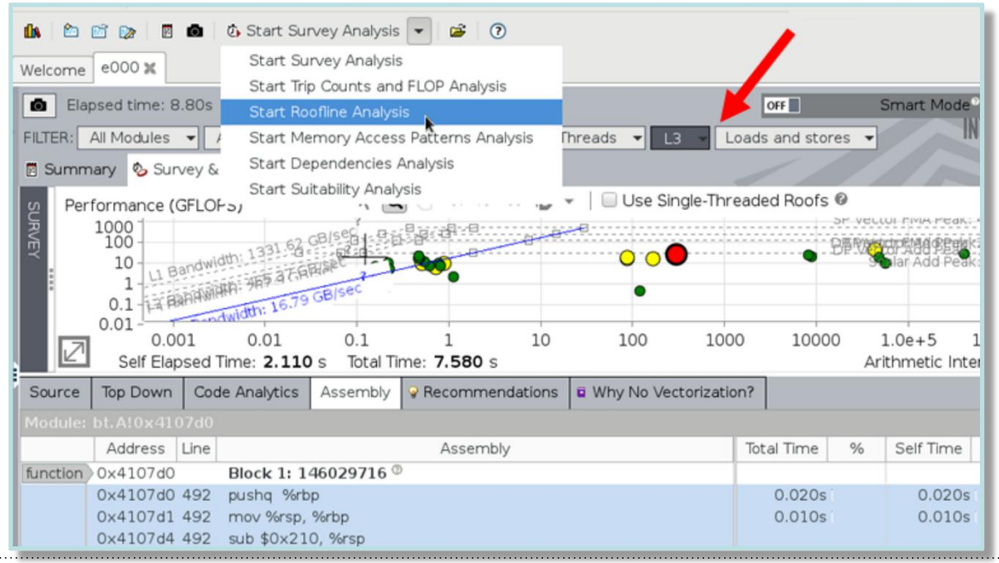
我的核函数可以向量化吗?

要回答这些问题，我们需要工具……

Intel Advisor

- 包括 自动化Roofline 建模分析
  - ✓ 自动化应用性能检测
    （每个循环嵌套/函数一个点）
  - ✓ 计算每个函数的 FLOPS 和 计算强度 (**CARM**)
  - ✓ AVX-512 支持，包含掩码
  - ✓ **I集成缓存模拟器[1] (层次化 roofline建模)**
  - ✓ 自动对目标系统进行基准测试（计算上限 ）
  - ✓ 与现有的 Advisor 功能完全集成





[1]Technology Preview, not in official product roadmap so far.

层次化Roofline vs. Cache感知式Roofline

…理解不同的 Roofline 公式的含义

有两种主要的Roofline公式：

- 层次化Roofline (Hierarchical Roofline: original Roofline w/ DRAM, L3, L2)
  - **Williams, et al, “Roofline: An Insightful Visual Performance Model for Multicore Architectures”, CACM, 2009**
  - **Chapter 4 of “Auto-tuning Performance on Multicore Computers”, 2008**
  - 为每个核函数给出多个访存带宽上限和多个算术强度数值
  - Performance bound受到每秒浮点运算次数（flops）和程序在访问内存时遇到的局限性限制(原始单一指标 Roofline 的叠加)
- Cache感知式Roofline（Cache-aware Roofline）
  - **Ilic et al, "Cache-aware Roofline model: Upgrading the loft", IEEE Computer Architecture Letters, 2014**
  - 在评估计算机程序的性能时，我们可以定义多个带宽上限，但只使用一个算法强度（AI）指标 (flop:L1 bytes)
  - 当缓存局部性（容量、冲突等）降低时，在不变的 AI 下，性能会从一个带宽上限降到更低的带宽上限。
    - 当程序访问内存时，如果缓存局部性降低（例如由于缓存容量不足或缓存冲突），那么程序的性能会下降。具体来说，程序的带宽会从一个较高的上限降到一个较低的上限，而算法强度（AI）保持不变。
- 现有工具的集成
  - 一些工具使用 Hierarchical Roofline，一些使用 cache-aware == **用户需要了解两者的差异**
  - Cache感知式Roofline已集成到 Intel Advisor工具中
  - 层次化Roofline（缓存模拟器）的评估版也已集成到 Intel Advisor 中

[1]Technology Preview, not in official product roadmap so far.

层次化Roofline

- 捕捉缓存带来的影响
- 层次化 Roofline 模型中，***不同层次的缓存将使用不同的“字节”进行计算***
- 算术强度（AI），取决于问题的大小。
  - 这意味着，当问题的大小增加时，缓存容量不足会导致缓存未命中（capacity misses）增加，从而降低算术强度。
- 内存/缓存/局部效应***被观察为 AI 减少***
- 需要***性能计数器或缓存模拟器***才能正确测量 AI

Cache感知式Roofline

- 捕捉缓存带来的影响
- Cache-aware Roofline 模型中的 AI 是指***呈现给 L1 缓存的 Flop:Bytes（加上非临时存储）***
- 算术强度（AI），与问题的大小无关。
  - 这意味着，无论问题的大小如何，算术强度都保持不变。
- 内存/高速缓存/局部性效应***被观察为性能下降***
- 需要***静态分析或二进制代码分析***来测量 AI

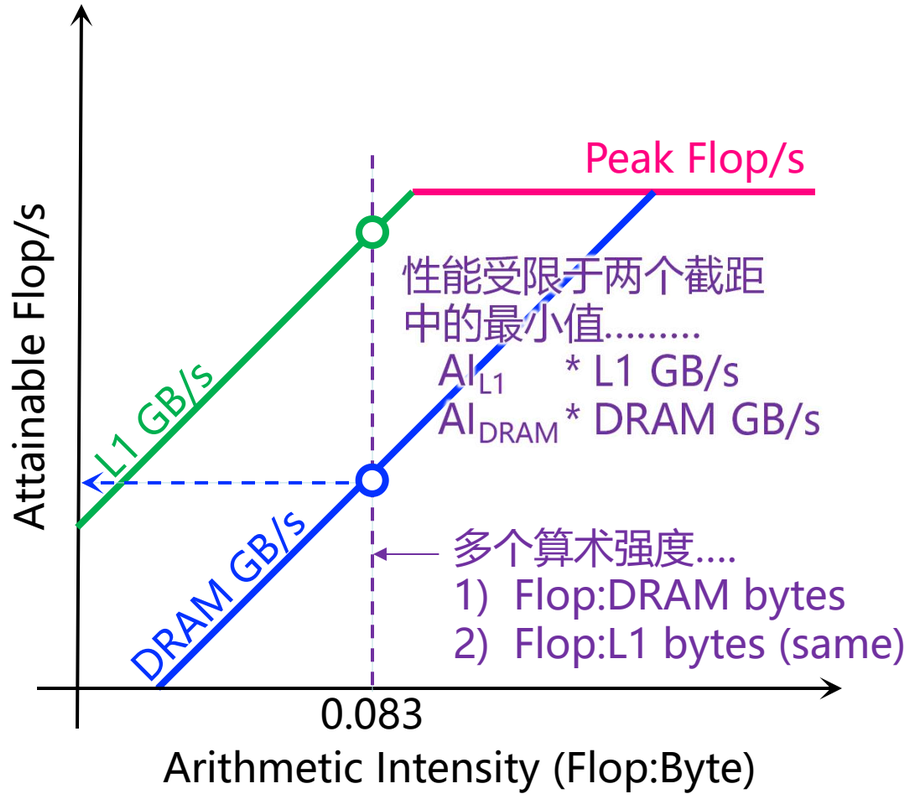
Example: STREAM

- L1 的算术强度...
  - 2 flops
  - 2 x 8B load (old)
  - 1 x 8B store (new)
  - = 0.08 flops per byte
- 没有缓存复用情况...
  - 对于任何增量，迭代 i 不会触及与迭代 i+delta 关联的任何数据
- ...导致 DRAM AI 等于 L1 AI

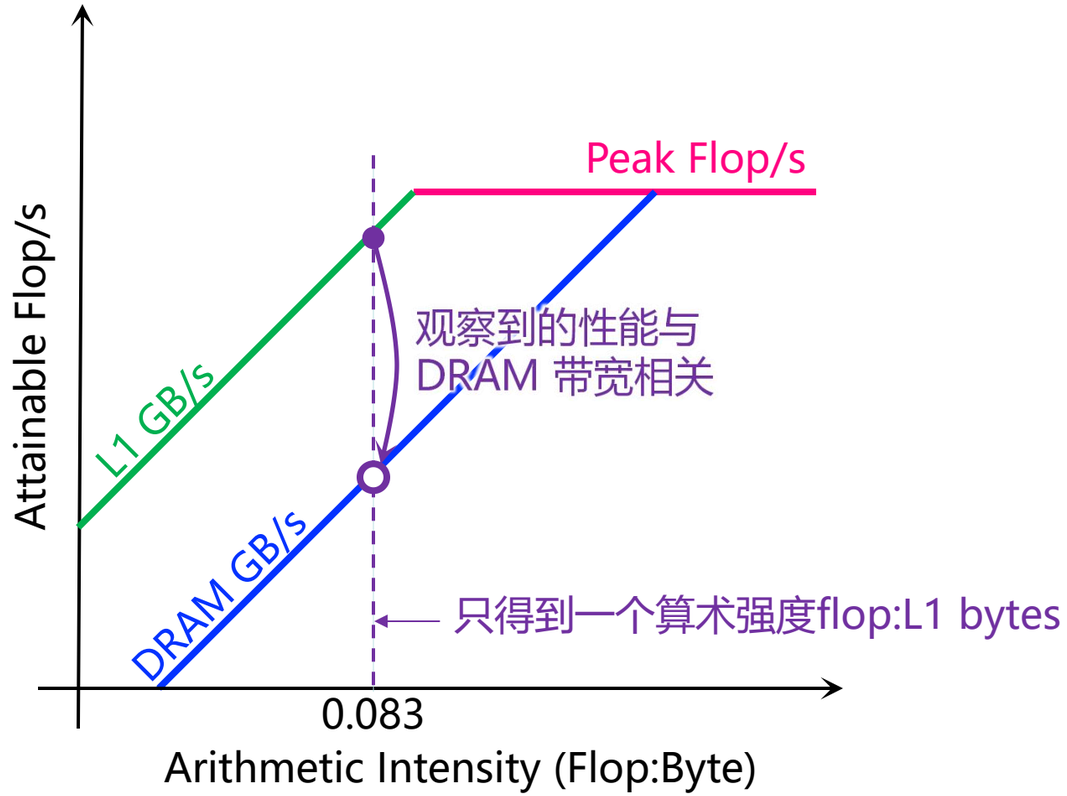
```
#pragma omp parallel for
for(i=0;i<N;i++){
  Z[i] = X[i] + alpha*Y[i];
}
```

Example: STREAM

Hierarchical Roofline



Cache-Aware Roofline



- 缓存重用对算术强度的影响取决于所使用的Roofline模型。
- 在Hierarchical Roofline模型中，算术强度取决于问题的大小，因此缓存重用可以减少缓存未命中（capacity misses），从而提高算术强度。

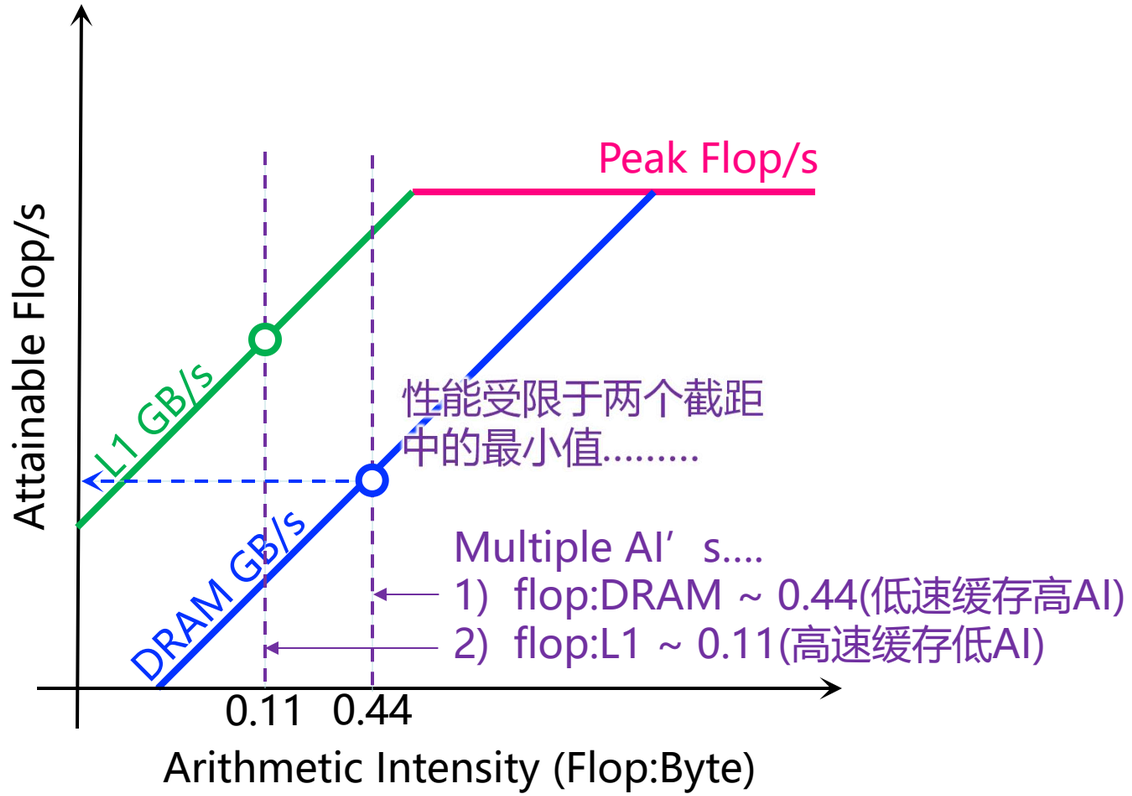
Example: 7-point Stencil (小规模问题)

- L1 的算术强度…
  - 7 flops
  - 7 x 8B load (old)
  - 1 x 8B store (new)
  - = 0.11 flops per byte
  - 一些编译器可能会进行寄存器洗牌以减少加载次数.
- 适度的缓存重用（cache reuse）
  - old[ijk] 在 i,j,k 的后续迭代中重用
  - old[ijk-1] 在 i 的后续迭代中被重用。
  - old[ijk-jStride] 在 j 的后续迭代中被重用。
  - old[ijk-kStride] 在 k 的后续迭代中被重用。
- …导致 DRAM AI 大于 L1 AI

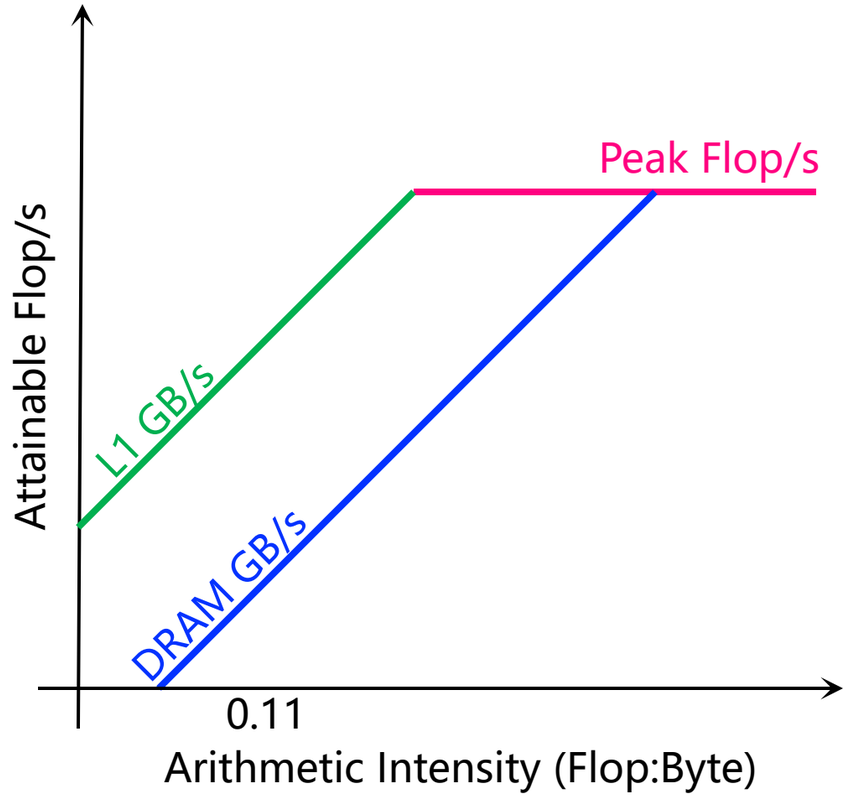
```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  int ijk = i + j*jStride + k*kStride;
  new[ijk] = -6.0*old[ijk        ]
             + old[ijk-1       ]
             + old[ijk+1       ]
             + old[ijk-jStride]
             + old[ijk+jStride]
             + old[ijk-kStride]
             + old[ijk+kStride];
}}}
```

Example: 7-point Stencil (Small Problem)

Hierarchical Roofline



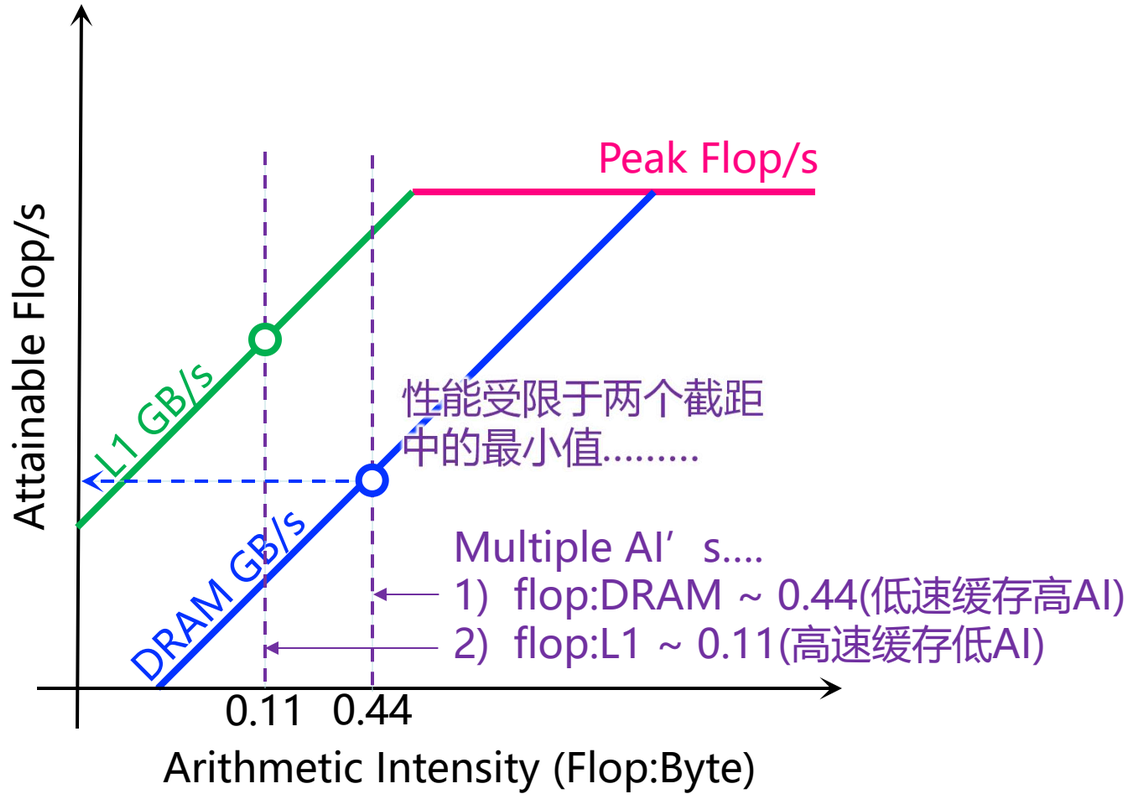
Cache-Aware Roofline



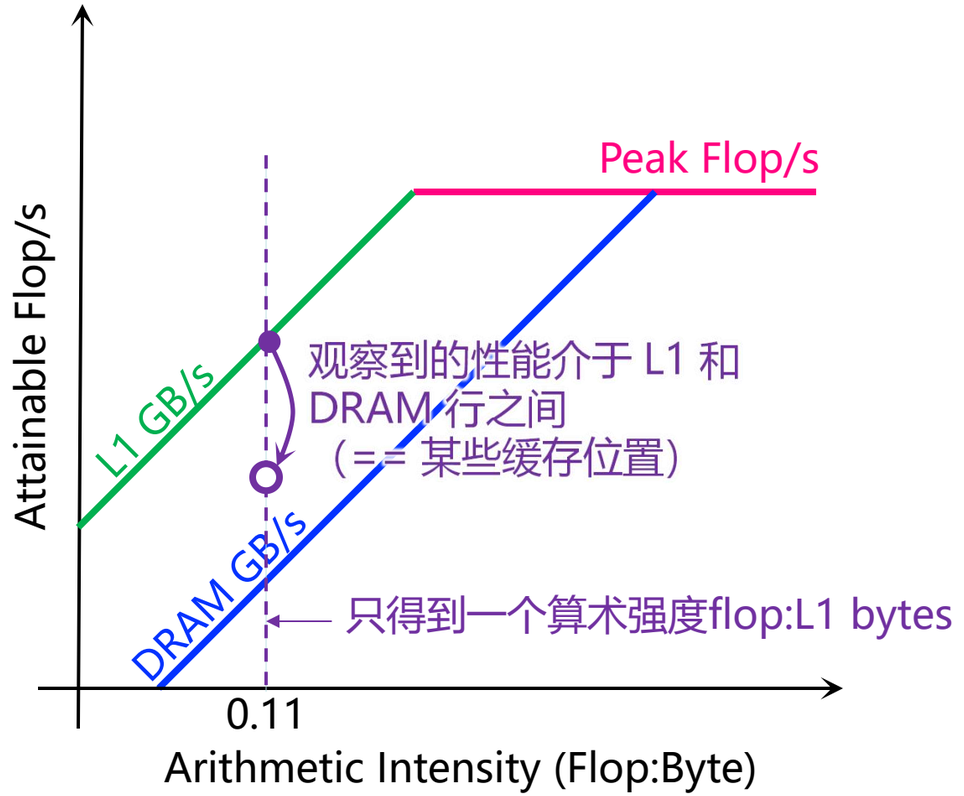
- 高速缓存（如L1缓存）的容量较小，而低速缓存（如L2或L3缓存）的容量较大。如果问题的大小超过了高速缓存的容量，那么会出现缓存未命中（capacity misses），从而降低算术强度。
- 而对于低速缓存（DRAM），由于其容量较大，因此可能不会出现缓存未命中，从而具有更高的算术强度。

Example: 7-point Stencil (Small Problem)

Hierarchical Roofline



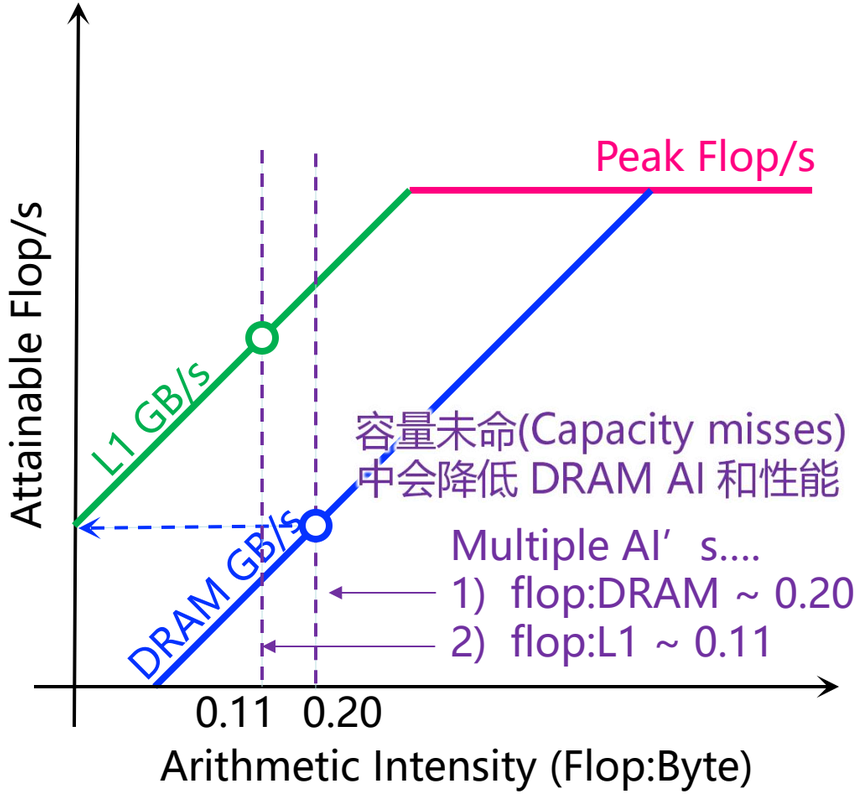
Cache-Aware Roofline



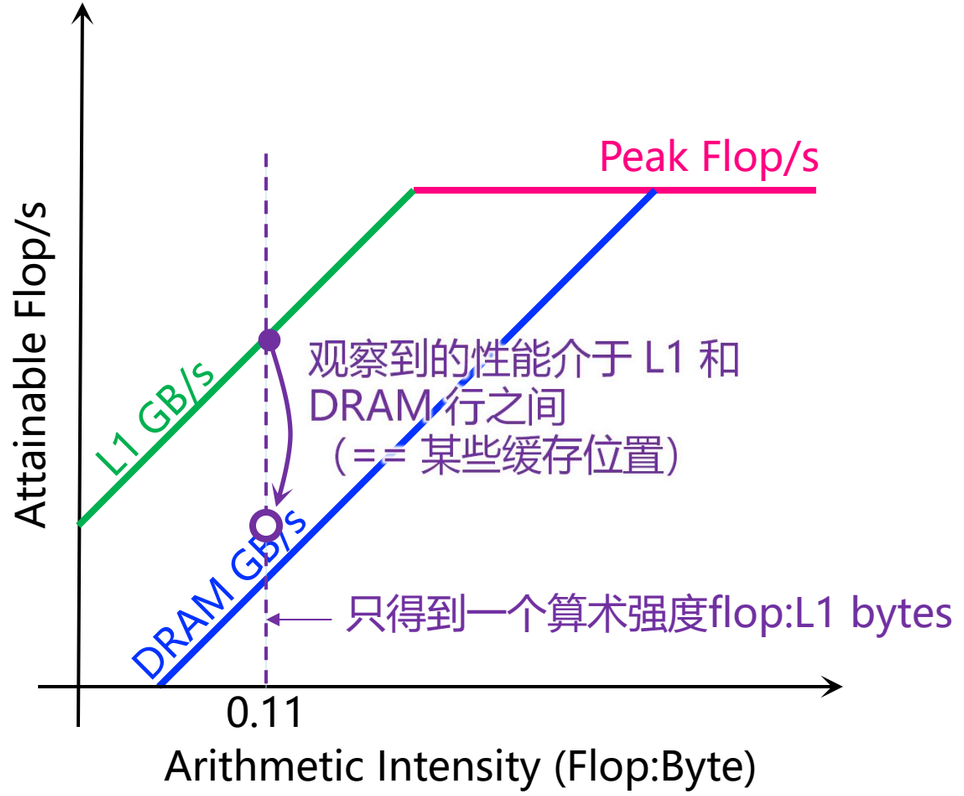
- 高速缓存（如L1缓存）的容量较小，而低速缓存（如L2或L3缓存）的容量较大。如果问题的大小超过了高速缓存的容量，那么会出现缓存未命中（capacity misses），从而降低算术强度。
- 而对于低速缓存（DRAM），由于其容量较大，因此可能不会出现缓存未命中，从而具有更高的算术强度。

Example: 7-point Stencil (Large Problem)

Hierarchical Roofline



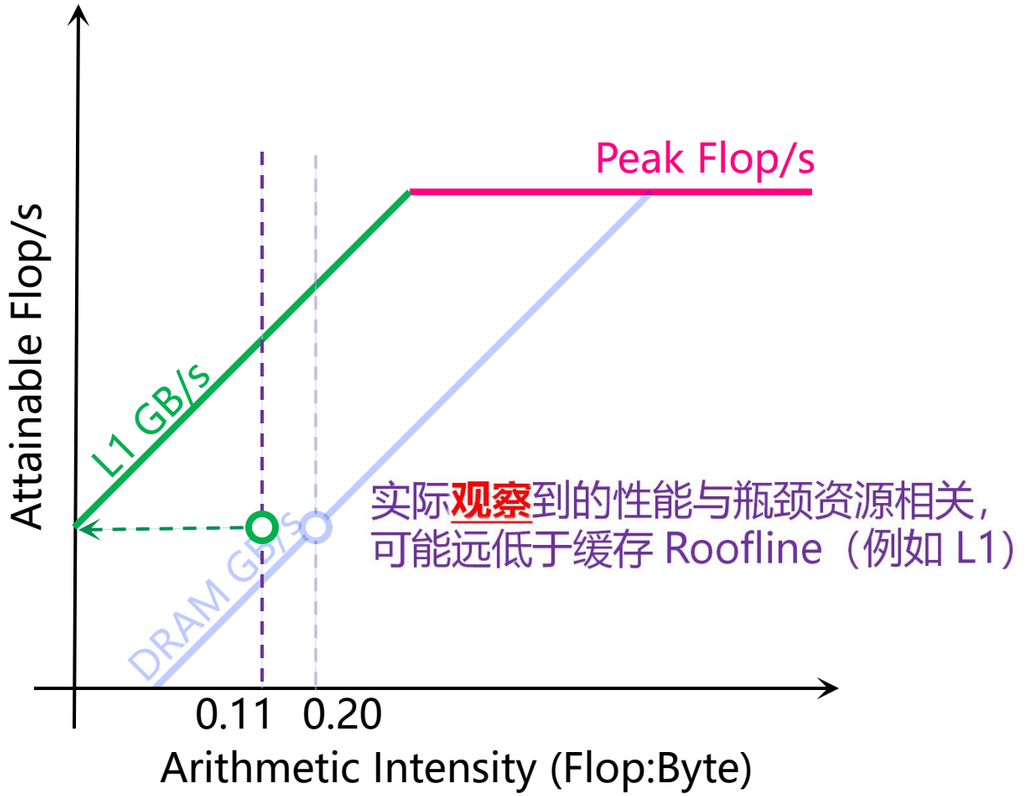
Cache-Aware Roofline



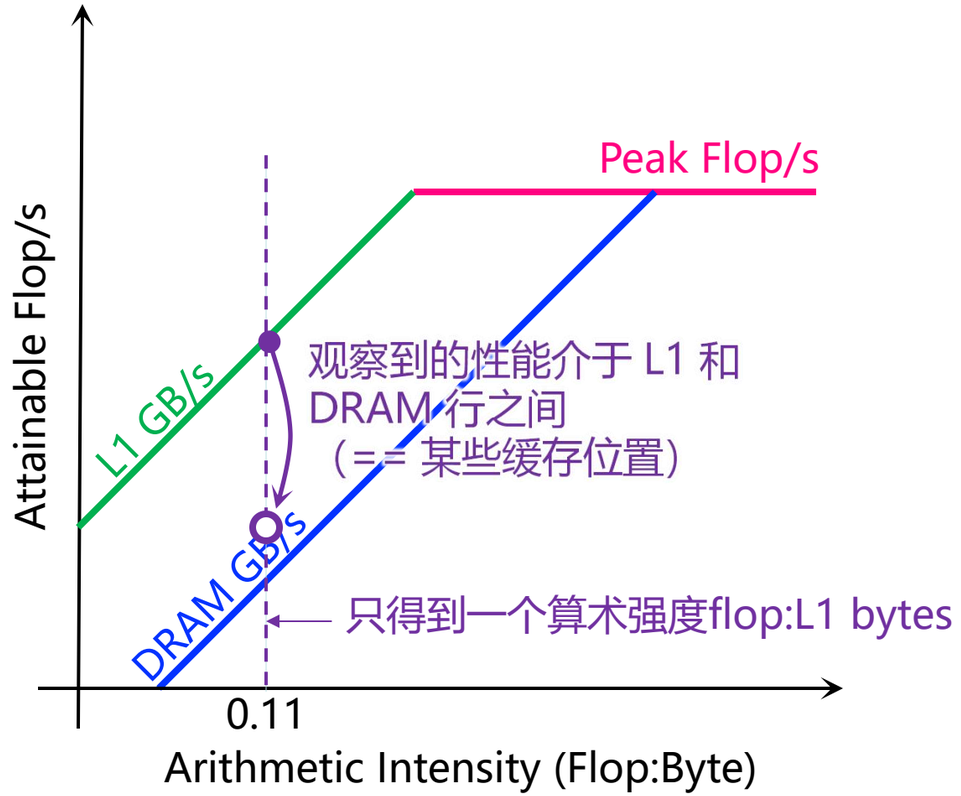
- **问题的大小**会影响**缓存未命中（capacity misses）**。
- 缓存未命中（capacity misses）指的是当缓存容量不足以容纳所有需要的数据时，导致的缓存未命中。**当问题的大小增加时，需要处理的数据量也会增加。**如果缓存容量不足以容纳所有需要的数据，那么就会出现缓存未命中。
- 因此，问题的大小会影响缓存未命中（capacity misses）。当问题的大小增加时，如果缓存容量不足以容纳所有需要的数据，那么缓存未命中就会增加。

Example: 7-point Stencil (Observed Perf.)

Hierarchical Roofline

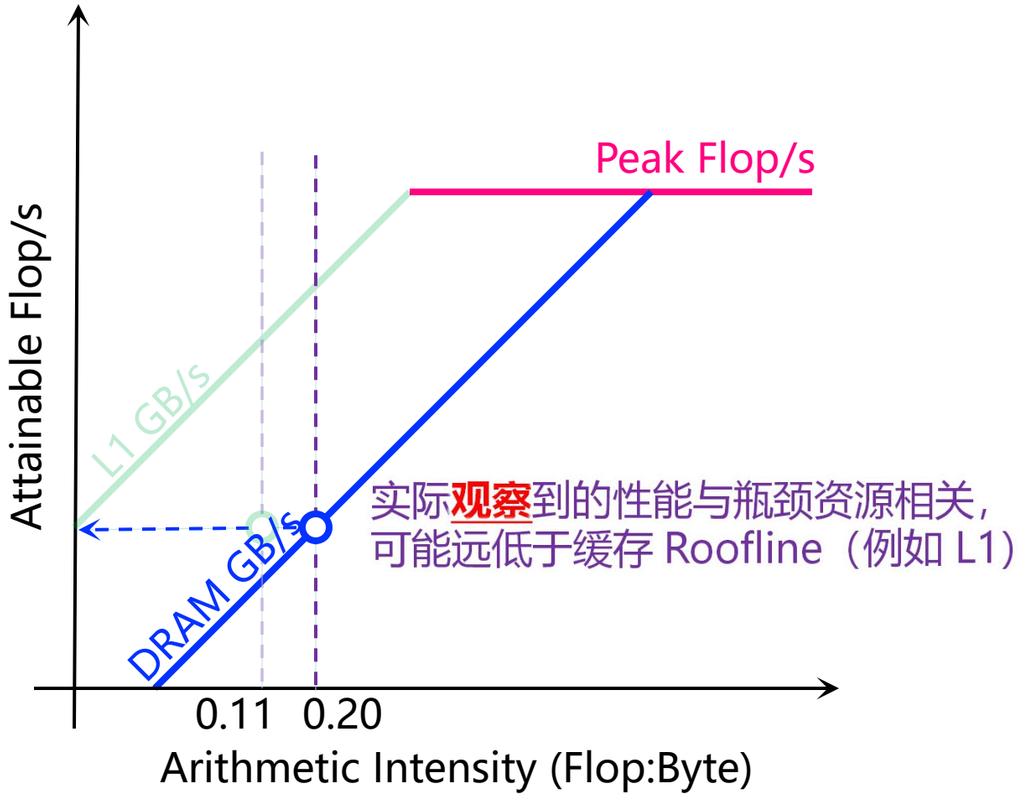


Cache-Aware Roofline

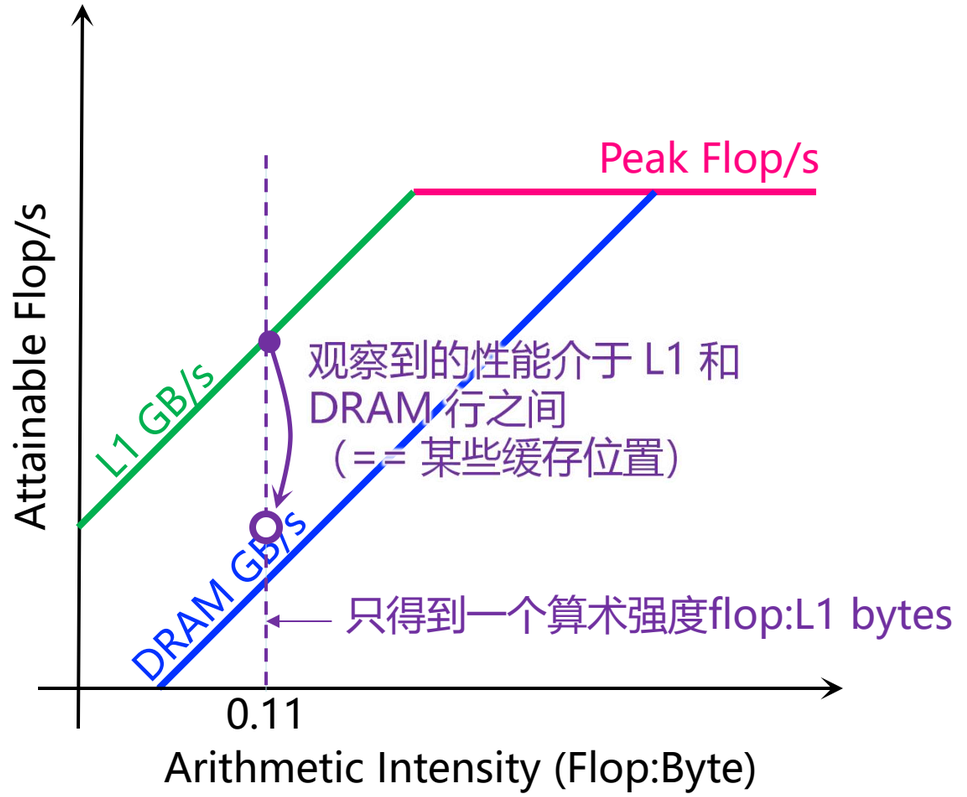


Example: 7-point Stencil (Observed Perf.)

Hierarchical Roofline



Cache-Aware Roofline

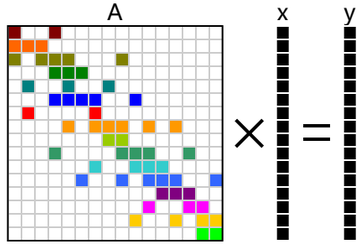


Roofline的实践示例

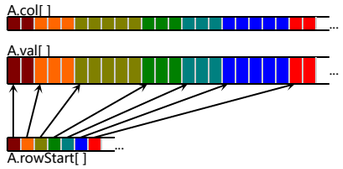
稀疏矩阵向量乘法（SpMV，Sparse Matrix Vector Multiplication）

- 什么是稀疏矩阵?
  - 大多数条目都是 0.0
  - 性能优势：仅存储/操作非零值
  - 需要元数据来重建矩阵
- 什么是 SpMV?
  - 计算 y=Ax，其中 A 是稀疏矩阵，x 和 y 是密集向量
- 挑战
  - **非常低的算术强度（通常 <0.166 flops/byte）**
  - 难以利用指令级并行 ILP （对流水线或超标量处理器不利）
  - 难以利用数据级并行 DLP（对 SIMD 不利）

- 由于稀疏矩阵中大部分元素都为零，因此spmv的计算过程中会出现许多不规则的内存访问和控制流。
- 这使得spmv难以充分利用指令级并行（ILP）和数据级并行（DLP），从而影响其在流水线、超标量和SIMD处理器上的性能。



(a)
algebra conceptualization



(b)
CSR data structure

```
for (r=0; r<A.rows; r++) {
  double y0 = 0.0;
  for (i=A.rowStart[r]; i<A.rowStart[r+1]; i++){
    y0 += A.val[i] * x[A.col[i]];
  }
  y[r] = y0;
}
```

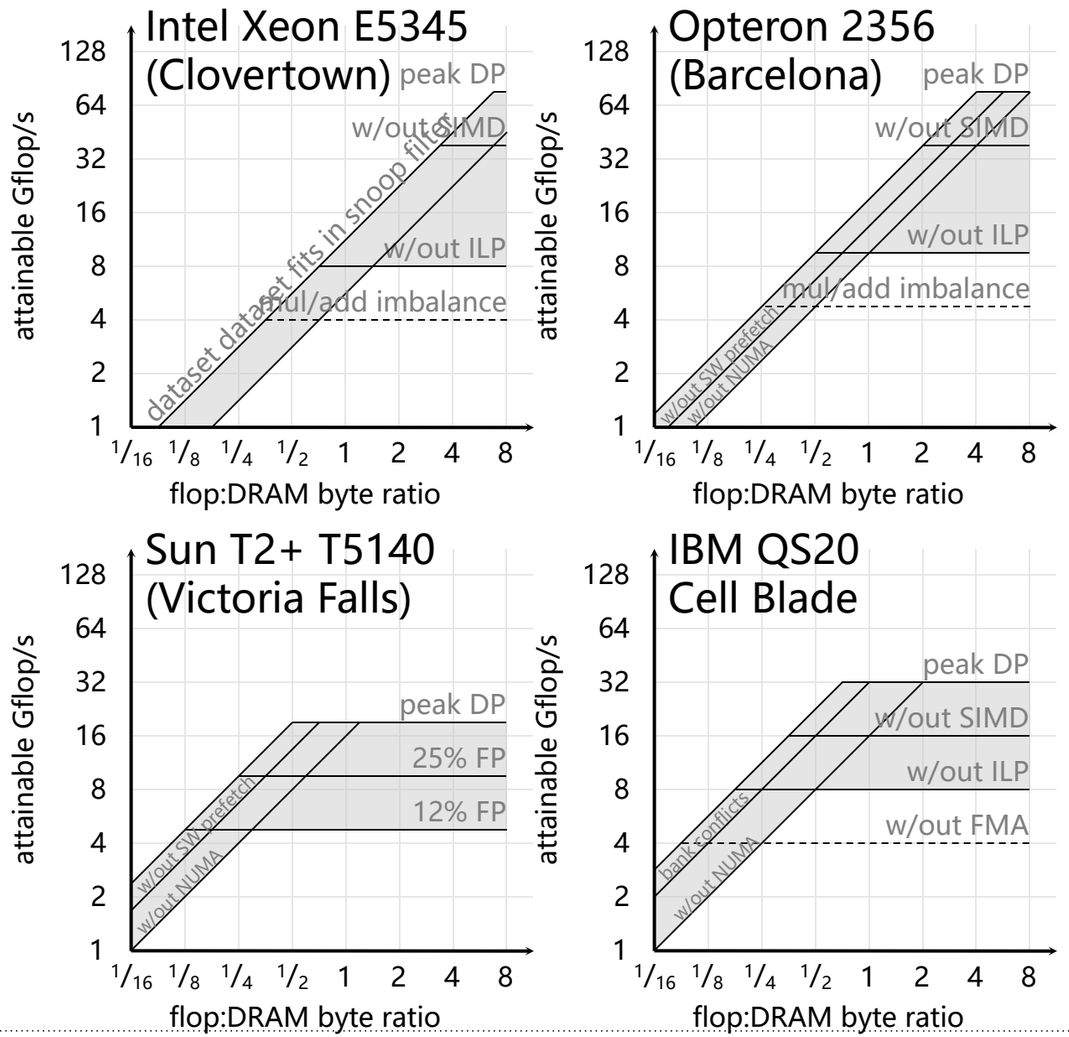
(c)
CSR reference code

Roofline model for SpMV

- 双精度 roofline models
- In-core 优化1..i
- DRAM 优化1..j

$$GFlops_{i,j}(AI) = \min \begin{cases} InCoreGFlops_i \\ StreamBW_j * AI \end{cases}$$

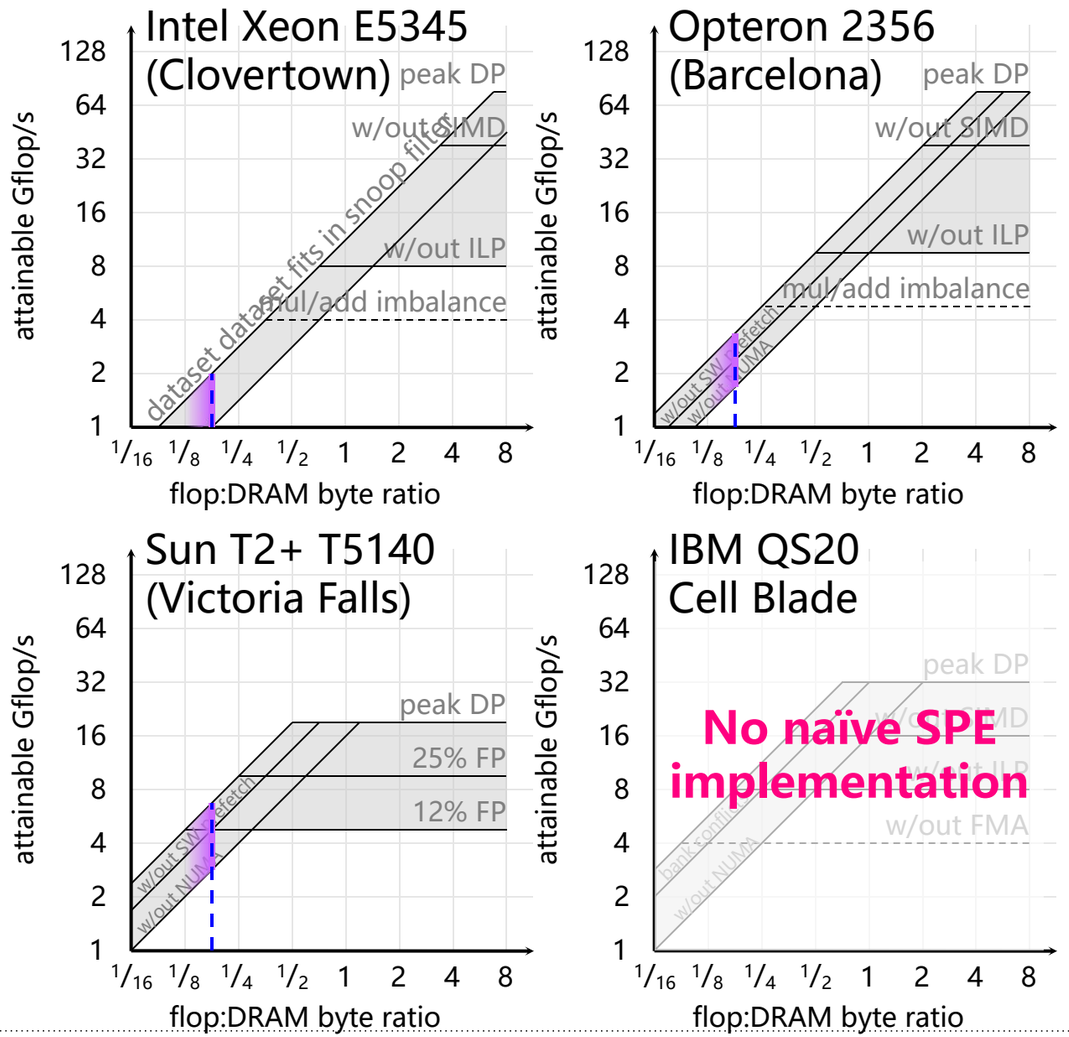
- FMA（浮点乘加）是一种计算操作，它将两个数相乘并将结果与第三个数相加。
- 在SpMV（稀疏矩阵向量乘法）中，FMA操作被用来计算矩阵中的非零元素与向量元素的乘积，并将结果累加到结果向量中。



Roofline model for SpMV

(叠加算术强度)

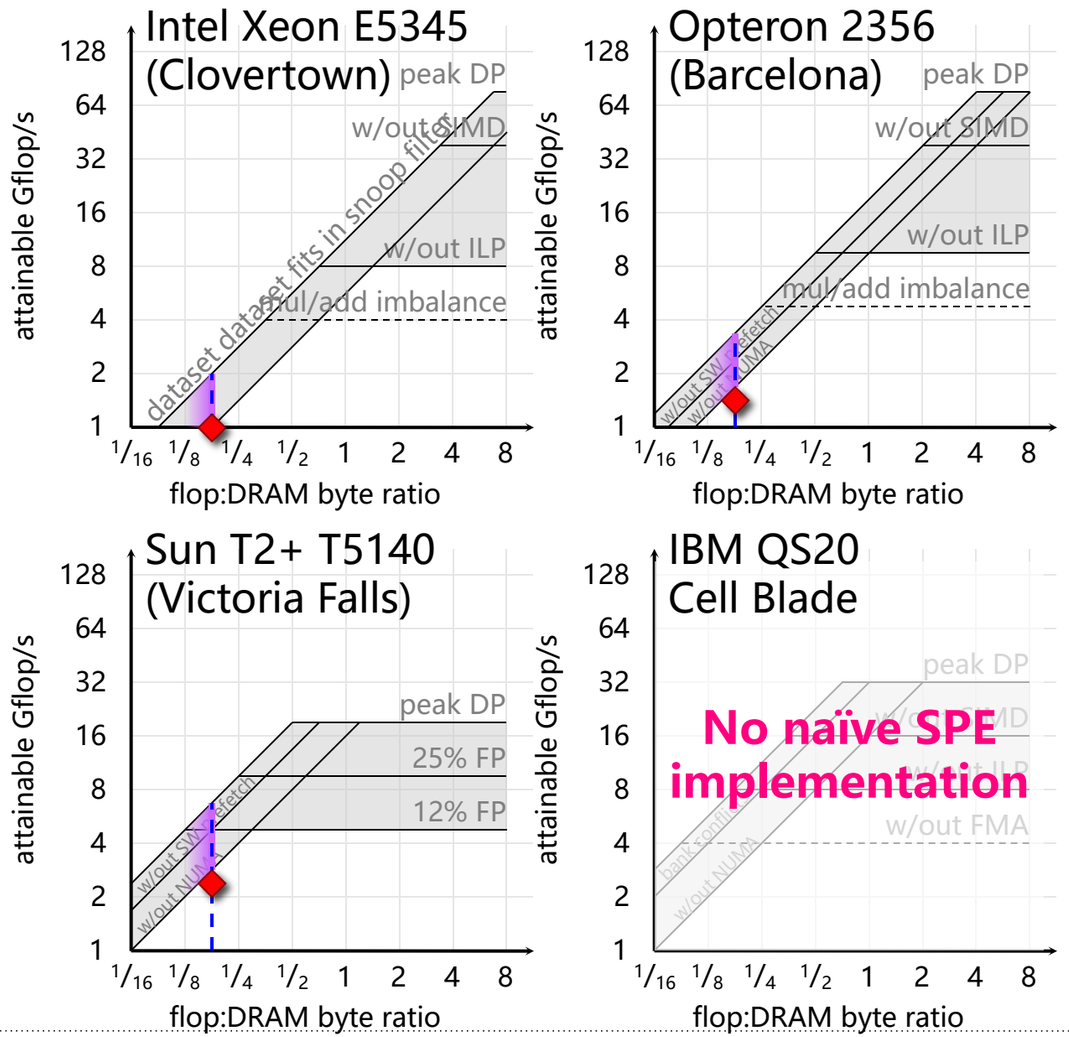
- 两个单位步长流（两个连续访问相邻元素的数据流）
  - 步长（stride）指的是在内存中连续访问两个元素时，它们之间的距离
  - 单位步长（unit stride）指的是步长为1，即连续访问的两个元素在内存中相邻
- 内置FMA（浮点乘加）
- 无指令级并行 ILP
- 无数据级并行 DLP
- FP在整个程序中的比重是12-25%
- 算术强度 flop:byte < 0.166



Roofline model for SpMV

(开箱即用的并行，非最优)

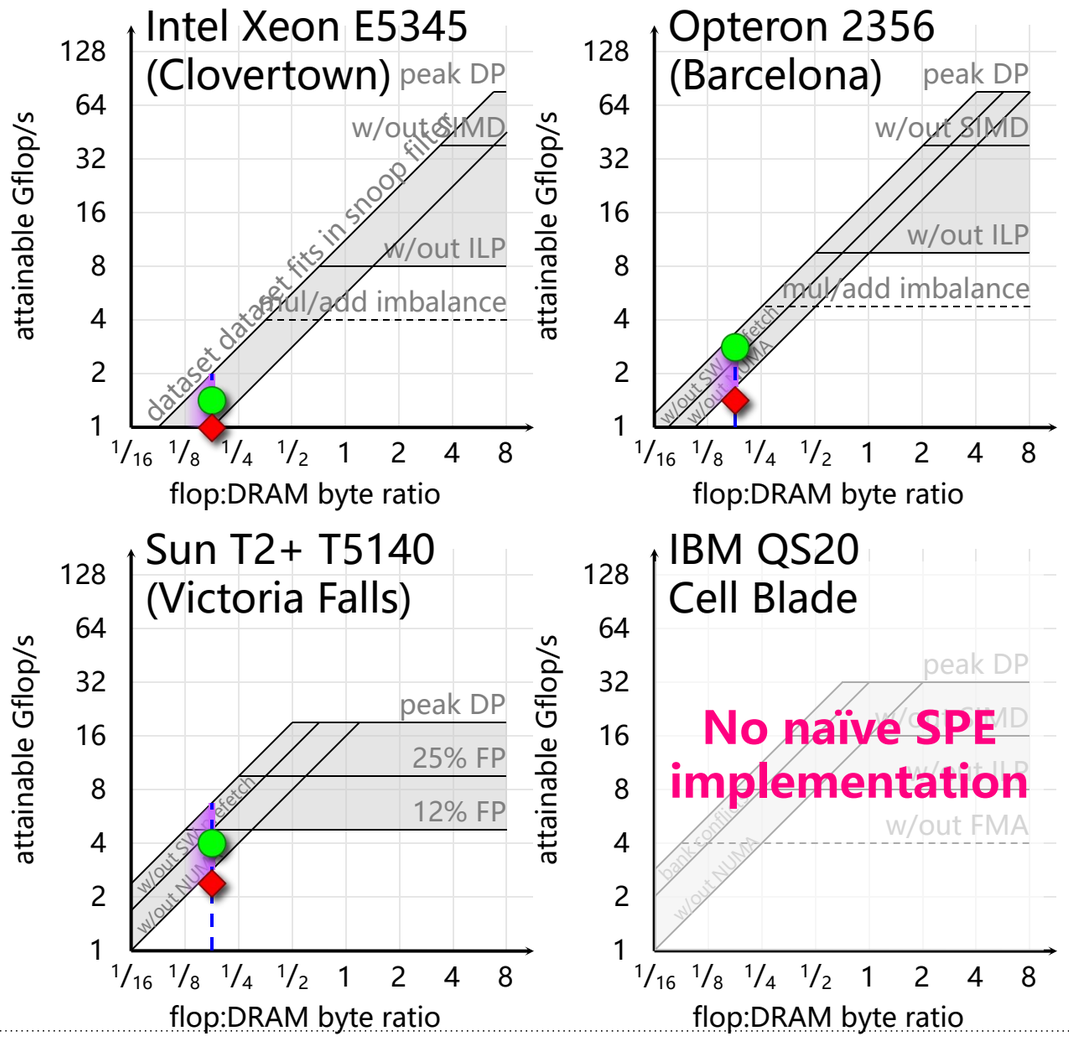
- 两个单位步长流（两个连续访问相邻元素的数据流）
- 内置FMA（浮点乘加）
- 无指令级并行 ILP
- 无数据级并行 DLP
- FP在整个程序中的比重是12-25%
- 算术强度 flop:byte < 0.166
- 将稠密矩阵以稀疏格式存储



Roofline model for SpMV

(NUMA 和 SW 预取)

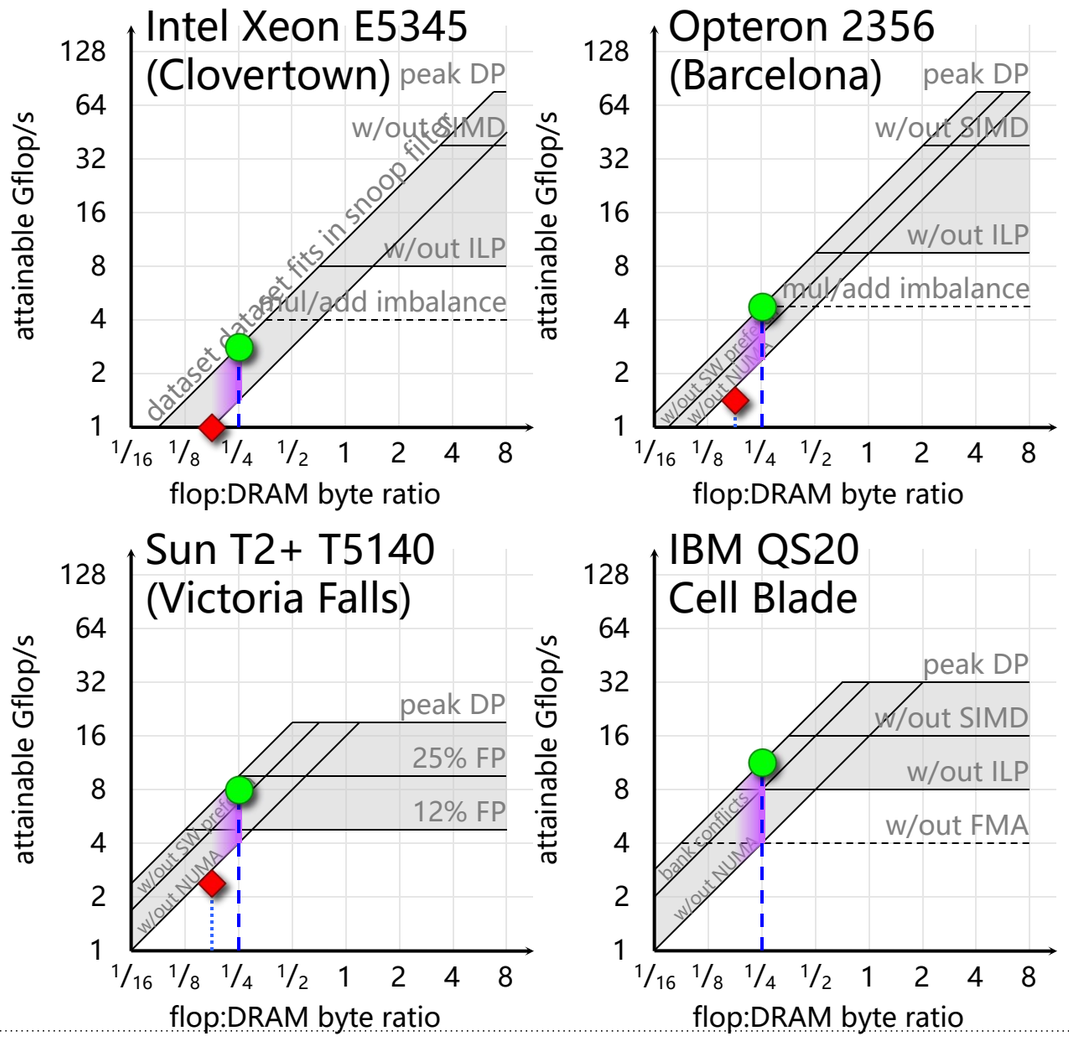
- 算术强度
  - flop:byte ~ 0.166
- 利用所有内存通道
  - **内存带宽最大利用率**



Roofline model for SpMV

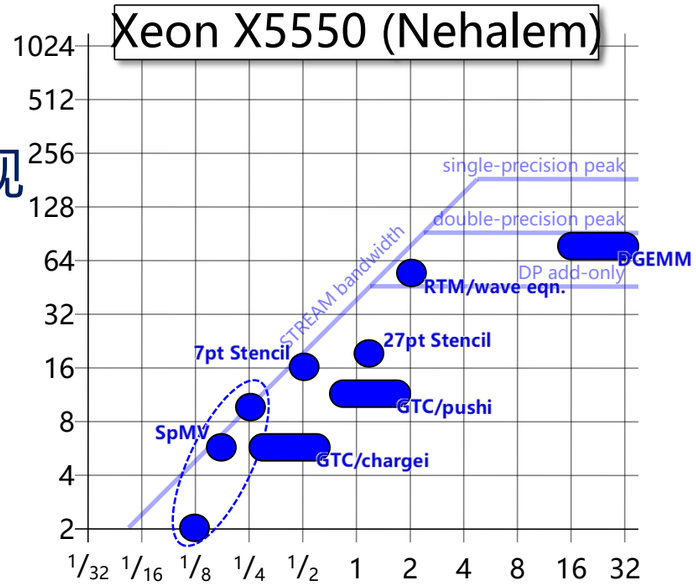
(矩阵压缩)

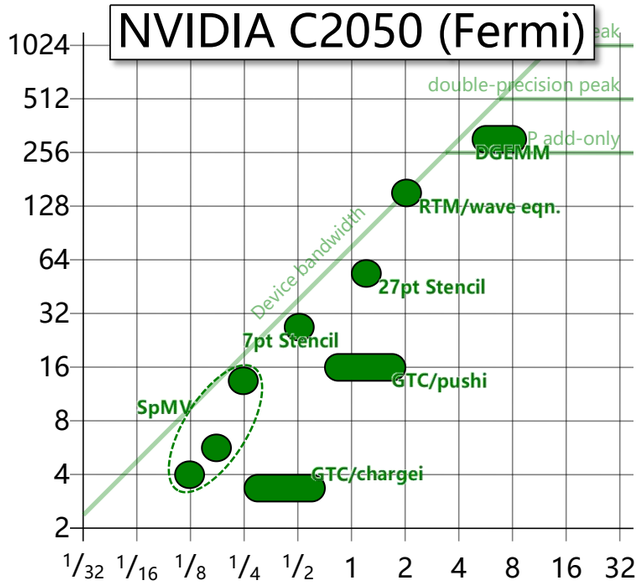
- 内置FMA
  - 融合乘加运算
  - 计算机指令
  - 在一个时钟周期内执行乘法和加法运算
- 寄存器阻塞优化技术（Register blocking）
  - 添加非零元素,来重新组织矩阵
  - 改善 ILP（指令级并行）
  - 改善DLP（数据级并行）
  - 提高flop:byte 比率
  - 提高FP% 的指令占比
  - 但可能会增加额外的内存流量



针对CPUs 和GPUs的Roofline

- 使用 Roofline 比较各种双精度内核的 CPU 和 GPU 性能
  - Flop/s 是理论数值；
  - 内存带宽来自 STREAM 或 SHOC。
  - 优化后的kernel函数性能与 Roofline 模型在两个平台上都有很好的相关性。
  - 一些不规则的应用程序（如 PIC）表现不佳，值得进一步研究。





Questions?