先进 CPU 微架构设计分析

作者：亦安

版本：V1.0 ：首次发布版本
欢迎关注亦安的公众号：亦安的数字小站

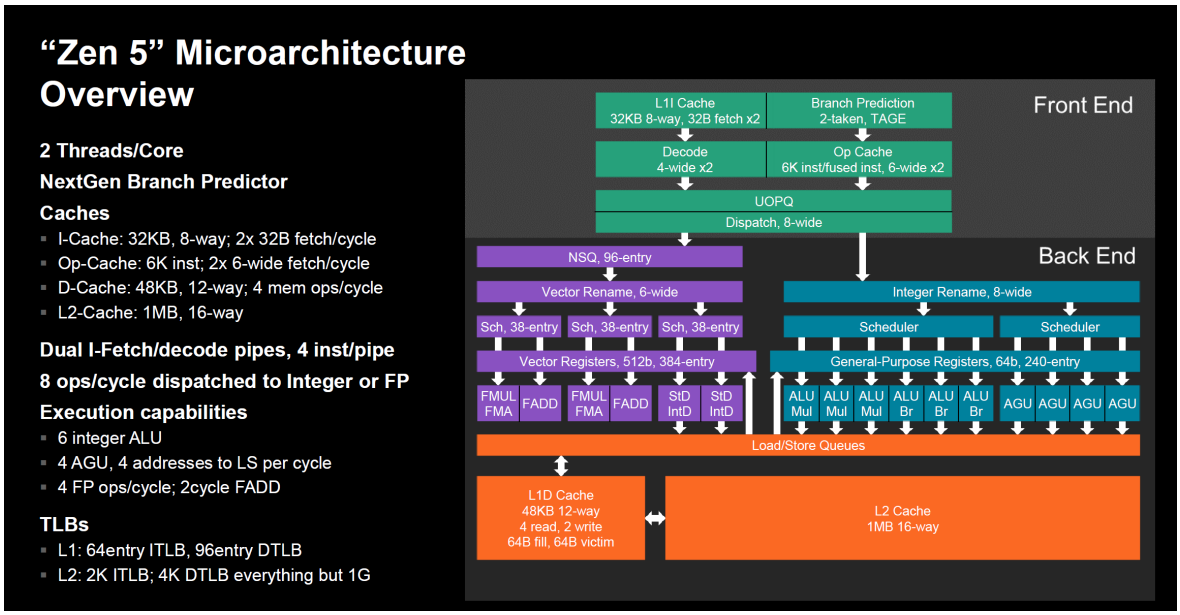
导言

此文档是先进微架构设计的分析文档，汇总当前主流厂商的一些微架构设计，目前还略显粗糙，后续会持续的更新，并且细化，当然多数的内容我已经在自己的公众号“亦安的数字小站”更新过，如果你对这种粗略的技术讨论不感兴趣，也可以通过公众号加我微信讨论一些微架构的设计细节，同时欢迎大家提出意见，并且可以和我同步维护文档。本文档将在公众号，Github：https://github.com/yian521/DawnCarol.git，以及腾讯文档同步更新。本文图片来自网络或者自制。

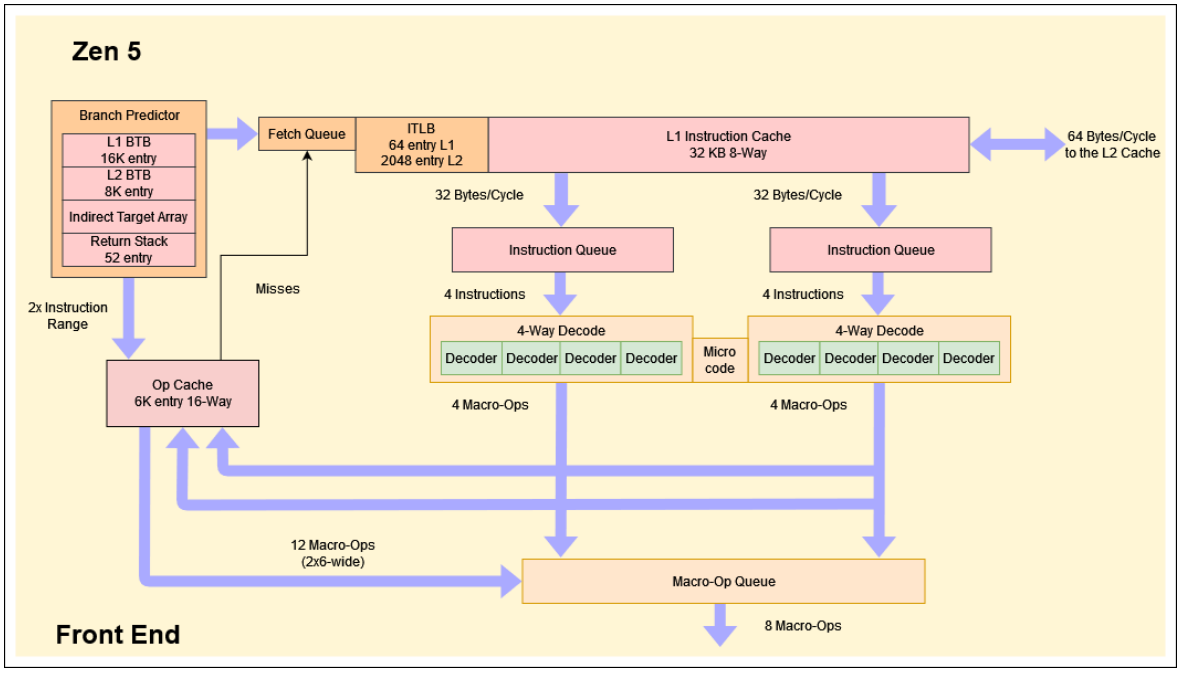
第一章　AMD Zen5 微架构设计

1.1 简介

2024 年 AMD 发布了变化巨大的一代微架构 Zen5，并宣称 Zen5 微架构是未来几代微架构的基石。相比较 Zen4，Zen5 的 IPC 增加了 16%，这个增长非常可观，支持 AVX512 变体以及 FP-512，Dispatch 宽度为 8，6 ALU，取指和解码（Fetch/Decode）均采用双流水线，本代预计采用 TSMC 4/3nm 工艺，每个核具有 2 个线程。



1.2 取指单元（IFU）



Zen5 的 IFU 相比较前代变化非常大，比较有特色的是采用双流水线，并且重新设计了预测器，对预测单元投入很大。

1.2.1 预测器

考虑到后端增加了执行带宽，前端需要给出更多的指令来支持后端的需求，而预测器则是前端核心部件之一，所以本代 AMD 就对预测器投入巨大，非常多的组件资源投入都在加倍。AMD 很多设计都是粗暴的增加组件的参数，本代了解到的是 MMU 核预测器投入了巨量资源，基本践行“大力出奇迹”的设计理念。

Zen5 的分支预测错误惩罚是 12-18cycle，这里的预测惩罚相比较 ARM 以及 RISC-V 架构的一些旗舰核是非常大的劣势，一般情况下先进的核预测惩罚压缩到 11cycle，甚至有些核压缩到 10cycle 一下，现代处理器的预测器预测精度目前提升缓慢，所以更多的架构师尤其是精简指令的架构师越来越倾向于压缩预测惩罚时间。

L1 BTB 是 16K entry 大小，这是巨大的升级，相比前代 Zen4 的 1.5K entry，升级到 10 倍，有条件/无条件直接/相同目标的间接都是 0 预测气泡，对于 Return 和多目标的间接都是 2cycle 预测气泡，考虑到其它主流厂商旗舰的 0 预测气泡预测 L1 BTB 基本都在 1K entry 左右，AMD 的设计可以说非常激进。L2 BTB 是 8K entry，如果 L1 BTB MISS 则会有 8 个预测气泡，这里非常奇怪，居然有如此巨大的气泡，因为 L1 BTB 巨大，所以考虑功耗的角度也不会同时访问 L1 BTB 和 L2 BTB，再者不同类型的指令可能也是顺序的访问，但 8 个预测气泡还是远超过我的想象。

间接目标预测器有 3072 entry，间接指令一般会有多个目标，预测会根据间接指令的历史从中选择一个目标。对于经常是一个目标的间接指令，一般用普通的 BTB 来预测结果，延迟也很低，这个特性如果普遍，微架构一般会在硬件上优化这个点。所以也说明代码使用间接类型的指令尽量不使用太多的目标，这对硬件来讲不友好，代码运行效率也低下。

方向预测使用 TAGE 算法，官方描述是更大，目前比较典型的值是 8-table，目前有论文显示到 12-table 后预测精度提升就不明显了，所以现在的优化也不是持续增加表长。

有意思的是，Zen5 的 BTB 使用双口的 SRAM 结构，这在预测器的设计中不常见，AMD 采用这个机制来实现每周期预测 2-taken 的分支。并且实现这个 2-taken/cycle 预测约束条件更少，而像其它的一些厂商会约束连续 block，不能跨 cache line 等。这种设计思想来自 30 年前的论文：“Multiple-Block Ahead Branch Predictors”

对 AMD 架构师的采访：

`George Cozma: Can the branch predictor do two taken and two ahead branches in a cycle? Or is it 2 branches of total? `

`Mike Clark: I mean it’s complicated, there’s rules around when we can do 2 ahead. But yes, in general I would say yes. `

`George Cozma: So yes, to two branches per cycle, or yes to two ahead and two taken branches?`

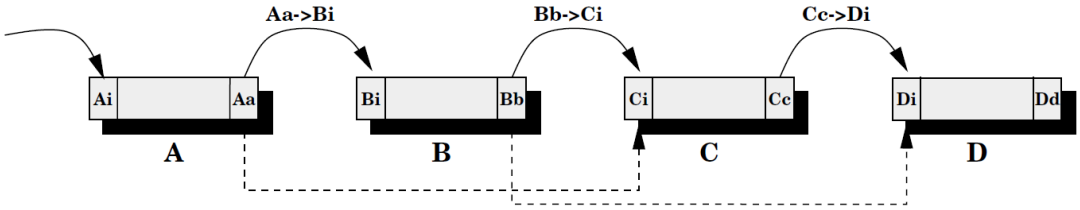
`Mike Clark: We can, in certain situations, do both.

AMD 实现的这个预测机制至少从理论上讲是很先进的，除了 BTB 是双口，亦安之前写的文章说过 ICache 也是双口，有 3 个依据，“Multiple-Block Ahead Branch Predictors”论文中写的双口 Cache。AMD 架构师 Mike Clark: They get fed from the dual ported instruction fetch to feed the decoder。Decode 是双流水线恰好对应取指的机制，不然预测带宽可以取两个不连续的 block，取指不可以似乎有点奇怪。但 C&C 文章说，核心在运行单个线程时只能使用一个解码集群，这让我觉得很迷惑，可能是将无序指令重新拼到微操作队列比较困难，但这点还是不太能理解为什么是线程独占的解码集群，有说法是因为 Zen5 目标频率很高，所以做成共享的很困难，但 AMD 又偏偏没采用 8 宽单线程解码。后来我想了想，使用双口和双流水线的解码簇似乎是 2 个问题，后面也就没纠结这个问题了。顺便说一下，我之前曾说过双口 ICache 基本没有人这样设计，实际不是，例如高通的核 ICache 据说是双口 ，但还没查相关资料。

说回预测器，传统的预测流程大致是这样的，下图的 i 是一个 block 的有效指令起始位置，大的 BTB 是 SRAM 结构，访问需要下一个周期出结果，也就是一个 cycle 只能拿到 1 个 block。例如 Aa 分支目标 Bi 访问获得 block B，已知信息是 block 有效

指令的起始位置 Bi 和 block 的分支 Bb，然后再用 Bb 的目标地址获得下一个 block 的位置。并以此重复，每个周期只能拿到一个 block 的信息。当然对于全相联结构 DFF 构造的预测器当然可以在 1cycle 做这些事情，但时序难。

而论文的方案是：使用 Aa 分支（目标）来预测 Ci，而不是传统的 Bb 来预测 Ci，使用 Bb 来预测 Di。需要双口 SRAM。



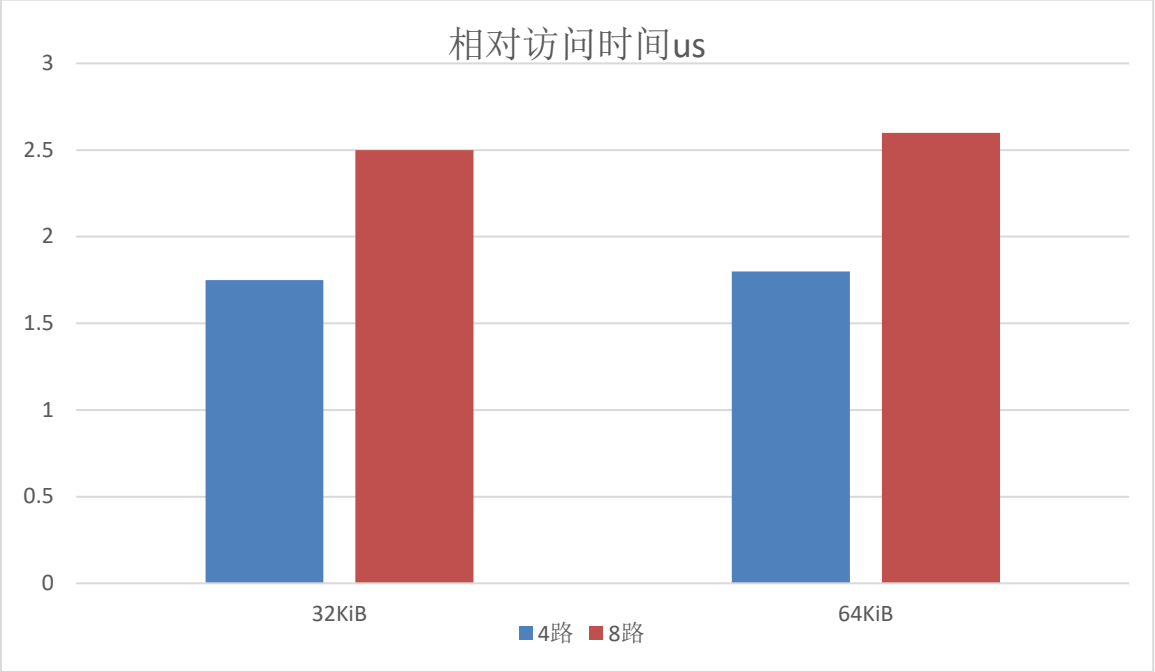
具体的设计思想感兴趣的朋友可以参考论文，此处亦安不再赘述，设计本身比较有意思。

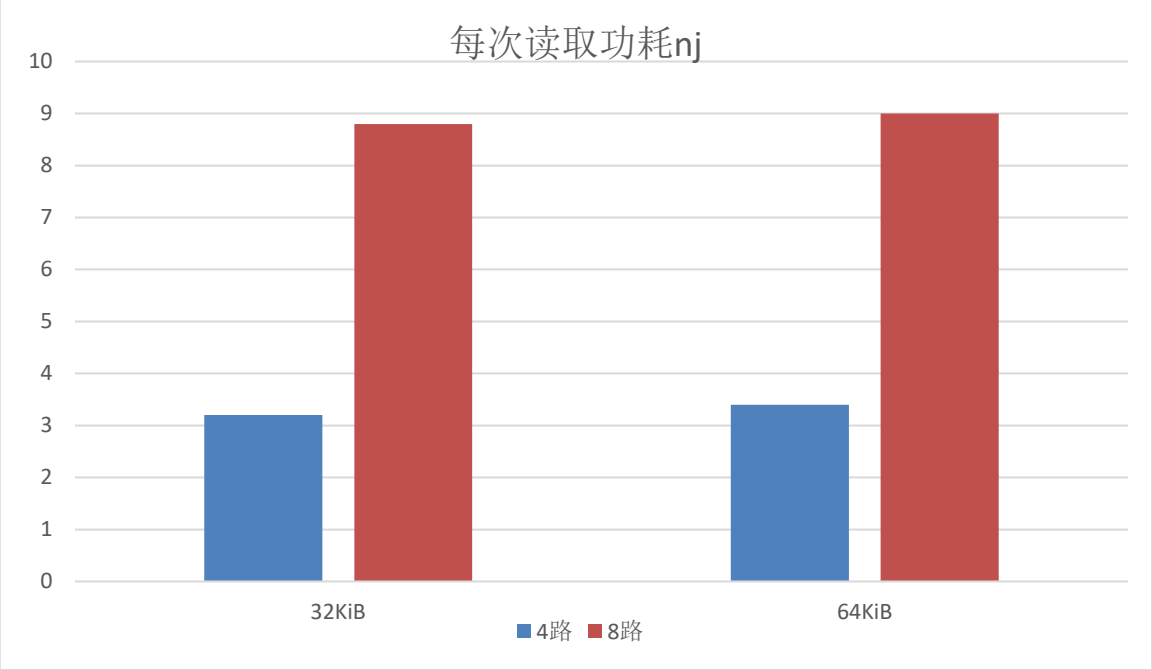
1.2.2 ICache

ICache 也属于前端的核心部件，对其研究非常多，像预取，替换算法，组织结构等等都有大量的论文。Zen5 的 L1 Cache 是 8 路组相联，具有 32KB 大小，Cache Line 是 64B，每个周期可以取 64B 的指令，需要 2 个对齐的 32B 块，L1 Cache 采用 LRU 替换算法，校验使用奇偶校验方式。这里 Zen5 取指带宽和 Zen4 相比变为 2 倍。前文也说过 ICache 可能是双口的设计，但没那么确定。这里的取指带宽只是理论上的，有测试显示在某些情况下，取指带宽弱于 Zen4，考虑到前端变化巨大，有些设计非常激进，而双 Decode 簇又不能双线程共享，这些可能都会造成 Zen5 有些情况的性能瓶颈。

有意思 AMD 和 ARM 对 ICache 设计理念的差异也体现双方对性能功耗面积取舍不同，ARM 自 A76 开始 ICache 都是 4 路 64KB（大小有些是可配的），而 AMD 自 Zen2 一直到 Zen5 ICache 均是 32KB/8 路的设计，复杂指令集的指令信息密度更高，所以采用更小的 Cache 也属于正常（尽管现在复杂指令集和精简指令集界限越来越模糊）。采用不同的结构对功耗，延迟以及命中率都有不小的影响。有一些相对通用模型显示，对于 32KB 或 64KB 的给定大小下，访问延迟均是 8 路比 4 路有明显的增加，而控制路数一定，仅增加 Cache 大小，不管是 4 路还是 8 路，将 Cache 大小从 32KB 增加到 64KB，延迟仅有小幅度的增加。而当测试读取功耗时，同样的，一定 Cache 大小，例如 32KB 或者 64KB 给定大小，8 路的读取功耗是 4 路读取功耗的 2 倍以上，当路数不变，将缓存大小从 32KB 增加到 64KB，4 路和 8 路功耗增加幅度均较小。从这个模型来看，似乎使用 4 路而增加缓存大小对延迟和功耗平衡更好，但

考虑到面积，命中率，以及有些厂商有定制工艺的角度，这种权衡不能下绝对的优劣结论，但可以看出，ARM 对能效的追求非常极致，而复杂指令集似乎对性能更加关注。亦安自己做了 2 张图如下，实际数据可能有偏差，仅参考数据趋势即可。





1.2.3 Op Cache

OC 用于存储之前解码过的指令，OC 命中可以让流水线跳过取指和解码，对于复杂指令集来讲这几乎是必须要实现的特性，可以在命中时降低流水线深度，在预测错误需要恢复时如果命中也可降低惩罚周期。本代 OC 具有 6K entry，低于前代的 Zen4，仅从数量上而言似乎是退步？这是因为增加了指令融合的机制，指令密度增大，属于 OC 的常规优化。Zen5 的 OC 每周期可以出 12 条指令，如果没有命中 OC，每周期只能出 4 指令/每线程，这大大提高了带宽，所以 OC 的命中与否直接影响了性能。

OC 是 64 set/16 路结构，每个 set-way（一个 entry）可以存储最多 6 条指令或者融合指令，融合指令可以提高 OC 的存储效率，其实各个 SRAM 结构，例如 BTB/TLB，都有类似的来提升存储密度的微架构。OC 使用物理 tag，可以让 OC 被双线程共享。

1.3 内存管理单元（MMU）

L1 TLB 也在这里说，ITLB 是 64 entry 全相联的设计，存储 4KB，2MB，1GB 的页面，DTLB 是 96 entry 全相联设计，存储 4KB，16KB，2MB，1GB 的页面。并且对 4KB 的连续页面进行聚合，聚合 4 个连续的页面。L1 TLB 的设计基本是业界常规的设计，也不会有特别花的设计方式。

L2 ITLB 是 8 路组相联，具有 2048 entry，存储 4KB 和 2MB 页面，这里 1GB 是不存到 L2 的，仅存在 L1 TLB，对于指令而言，操作系统基本不会分配 1GB 的页面，放到 L2 ITLB 里面可能会增加潜在访问次数，从这个角度看，放到 L1 ITLB 更合理。

L2 DTLB 是 16 路组相联具有 4096 entry，存储 4KB 和 2MB，额外的 4 路组相连结构的 L2 DTLB 存储 1GB，具有 1024entry。这里的选择是 PPA 的平衡，例如页面混装还是分开装的选择，分开 TLB 装不同页面有利于降低延迟，同时访问增大了功耗。混合装页面有利于资源利用率，一个周期的功耗相对低（但完成一个任务的功耗未必低），可能需要多次访问，这里只是简单聊下，不展开讲，不同的场景这些利弊也可能不同，不是绝对的，但不会差的太远，这里的权衡不会像我讲的那么简单。

值得关注的是 Zen5 的 L2 TLB 非常巨大，给出个横向对比，今年 ARM 的旗舰服务器核 V3 的 L2 TLB 是共享的 2048 entry，Intel 今年的 Lion Cove 是 1024 entry+1024entry 的大小，各家的设计基本都在 2K 的样子，而 AMD 则是指令 2048 entry，数据 4096entry+1024entry 大小，4096 entry 的 TLB 可以理解为采用了非常激进的预取，但尤其奇怪的是给 1GB 的页面单独分配了一个 1024 的 TLB，目前的软件生态真的会需要如此多的 1GB 页面吗？

PWC（page walk cache）加速页面转换的过程，减少对 Cache 的访问，常用的技

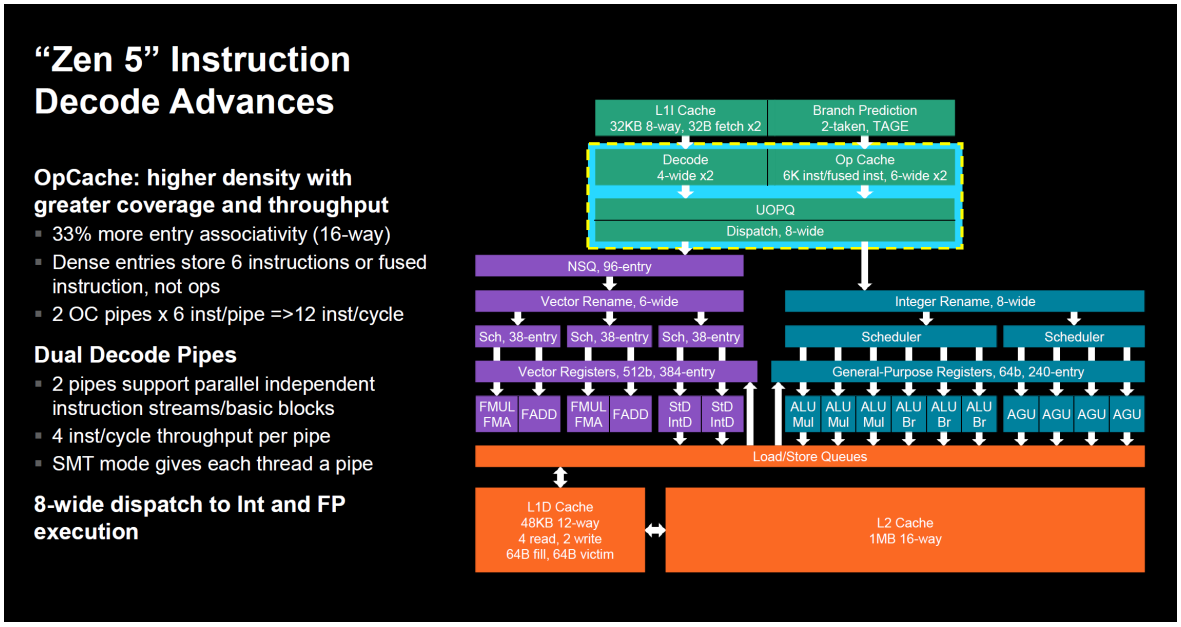
术。

可能会有人奇怪为什么 X86 常规的页面是 4K/2M/1G，但 TLB 会存储 16K/4M 呢？如果大家关注过 ARM 的微架构，也会有类似的情况，不同于架构常规定义的页面大小，这些大小的页面是因为有硬件聚合技术，smash 技术，连续技术等产生的。简单说硬件聚合就是将连续的 VA 和连续的 PA 放入同一个 TLB entry 来增加存储密度的，一般设计是 4 聚合，即 4 个 4K 页面聚合存成 16K 的大小。Smash 技术是大页面粉碎成小页面存到 TLB，这个我没有注意是不是架构支持的，如果仅是微架构支持的话可能维护需要稍微注意点。连续页面技术是架构支持的，软件配置标志位，表示连续的页面可以使用，例如 RISC-V 的拓展可以实现 16 个连续 4K 实现 64K 的页面，ARM 也有类似的机制，基本都是 16 连续，只是更完善，连续的页面也更多。

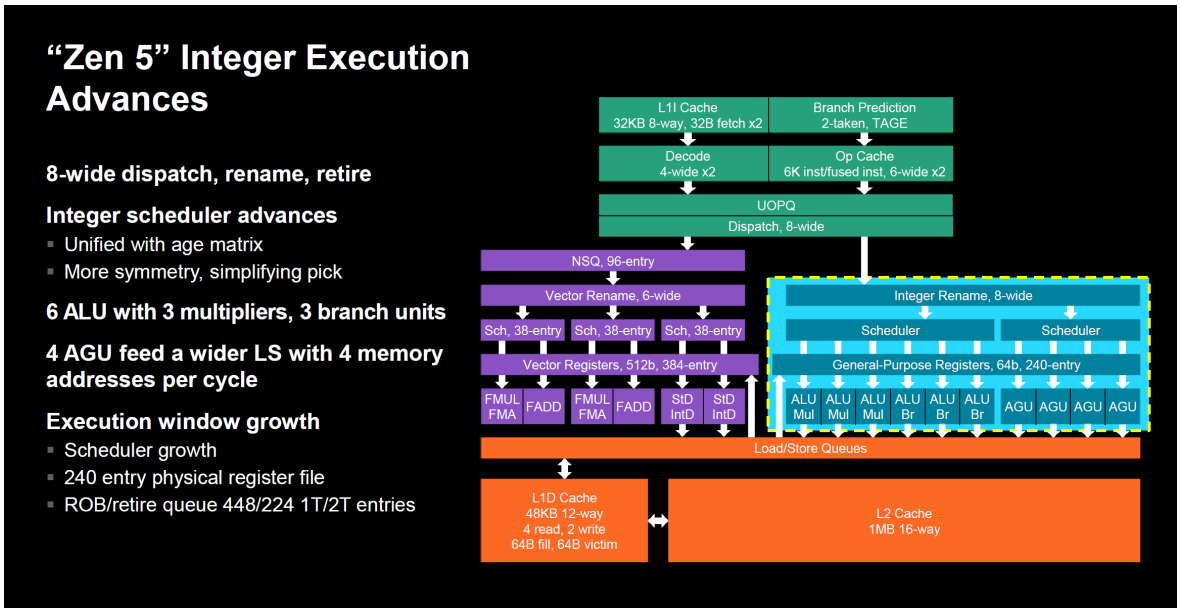
在 CPU 上的 MMU 相关的微架构都非常成熟了，不同于 GPU，CPU 各家做的差异不是很大，都是围绕那几个点权衡，架构上和软件上的支持倒是在不停跟进。

1.4 解码（Decode）

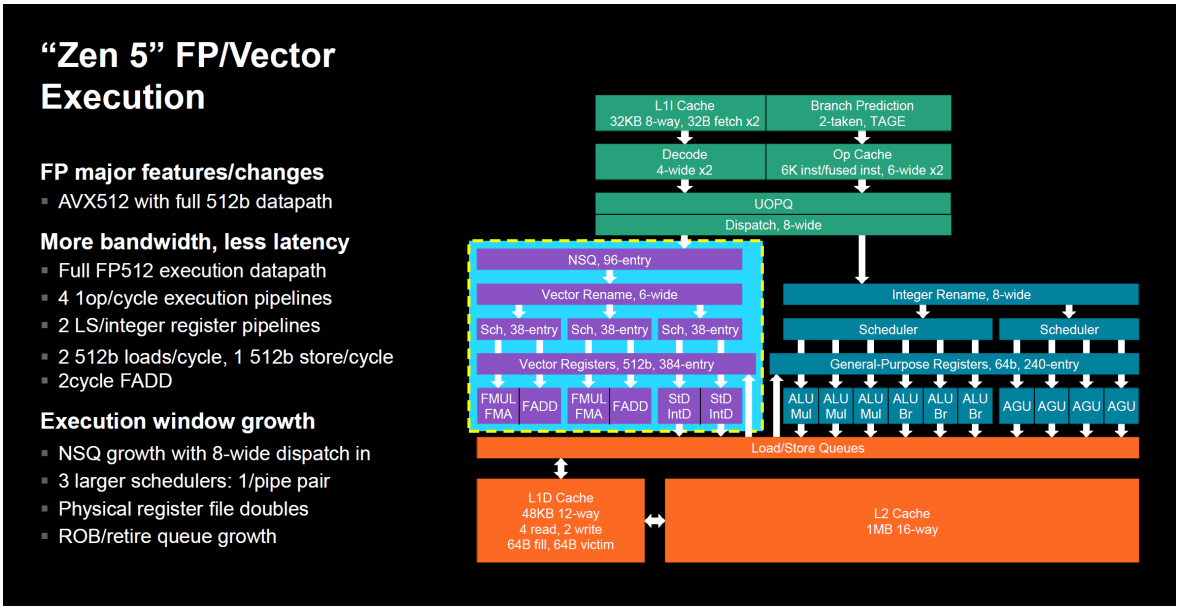
解码模块比较有特色的是采用双流水线，双流水线解码 32byte x2 以支持 ICache 最多出 64byte 指令，每个 decode 流水线有 20 深度的队列`IBQ`用于对预测/取指等单元的解耦。每个解码流水线可以扫描 2 个`IBQ entry`，最多出 4 条指令。可惜双流水线只能给单线程使用，不可以共享，甚至当只有一个线程的时候，也只能使用一个解码簇，我不确定这个解码簇和 Intel E 核的解码簇是否类似，从各方消息来看是不一样的，不能线程共享的解码簇是一个瓶颈，不过和 intel E 核相比，毕竟 Zen5 的目标频率更高，希望后续 AMD 能处理掉这个问题。



1.5 整数执行单元（IEU）



1.6 浮点执行单元（FPU）



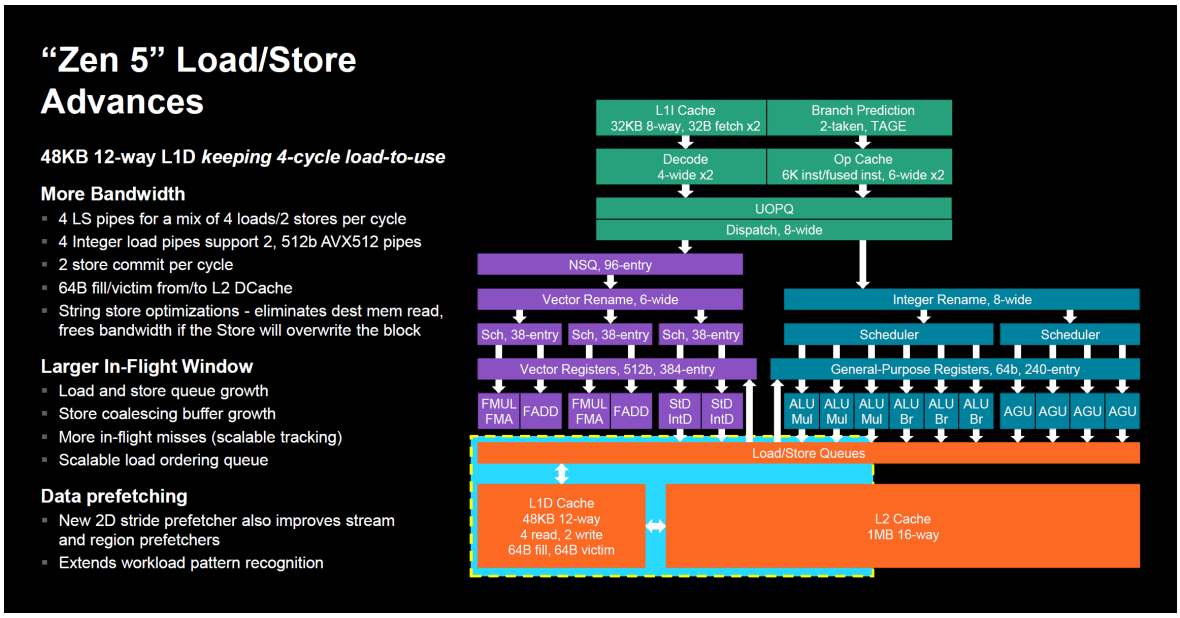
1.7 访存单元（LSU）

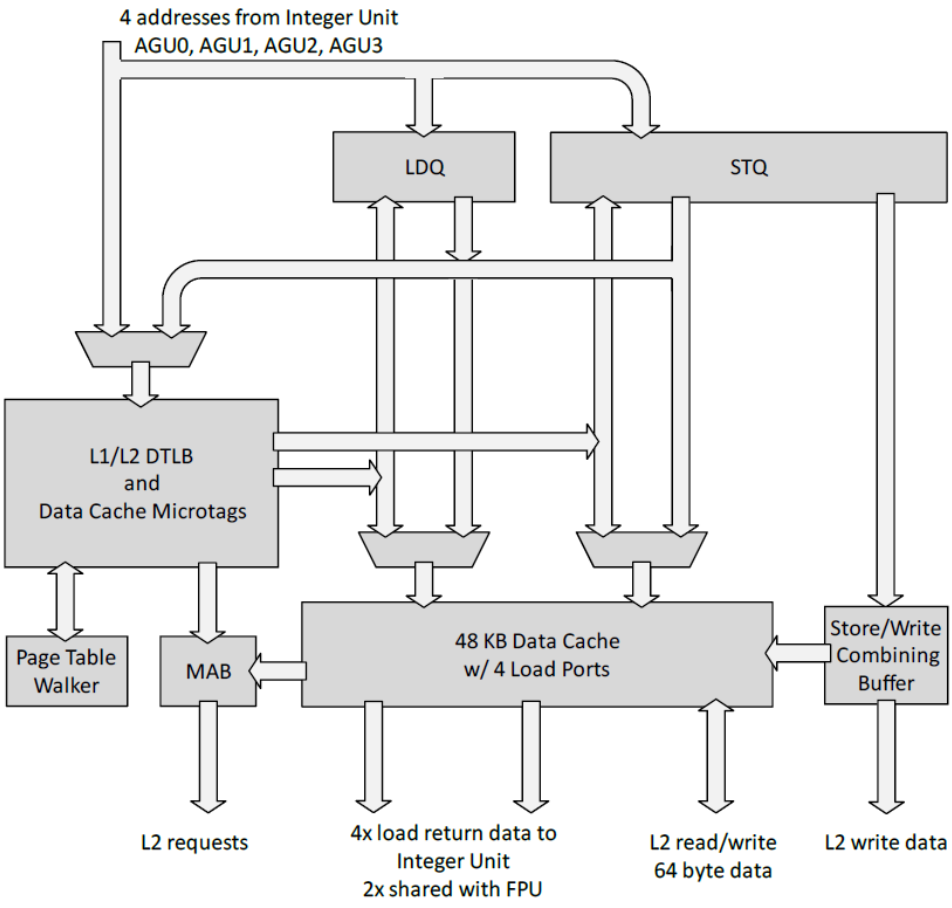
访存单元用于处理数据访问，L1 Dcache 是 12 路组相联，具有 48KB 大小，和前代 Zen4 的 32KB/8 路相比，不管是容量还是路数都有提升，理论上讲命中率会有不少的提升。

LSU 每个周期支持最多 4 个对 Cache 的操作，所有的操作均可以是 load，支持最大 2 个 128/256bit 的 load 或者 1 个 512bit 的 load，最大支持 2 个 store 操作，如果是 512bit 的 store，则支持最多一个 store 操作，带宽有了很大提升，是前代的 2 倍。

LSU 可以跟踪 64 个未完成的 load，并且不限制完成 load 的数量，同时支持动态追踪操作，可以 load bypass 更老的 load，也可以 load bypass 不冲突的更老的 store，做这些操作确保不会违反架构规定的 load 和 store 顺序。同时 LSU 可以跟踪 124 个未完成的 L1 MISS。

关于访问延迟，简单的地址生成，L1 Cache 命中，有 4cycle 的 load-use 延迟，浮点有 7cycle load-use 延迟。简单的地址生成，L2 Cache 命中则 14cycle 的延迟，浮点有 17cycle 的延迟。L3 命中，简单地址生成有 46cycle 的延迟，浮点有 53cycle 的延迟。如果是复杂的地址生成模式，则在简单地址生成模式上增加一个周期的延迟。



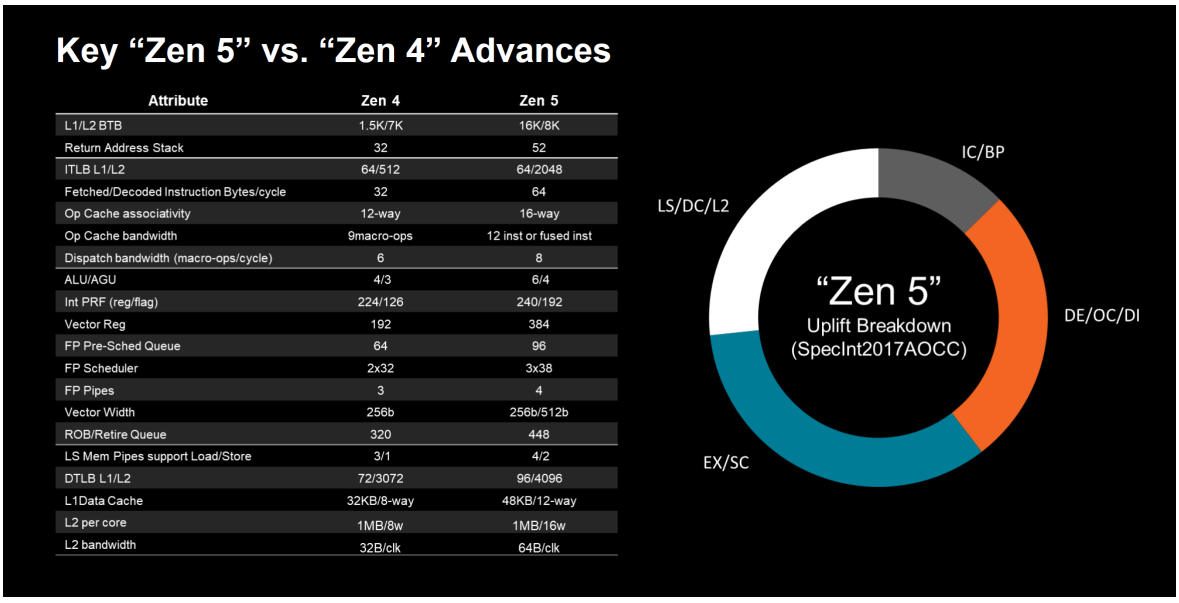


1.8 二级缓存（L2C）

Zen5 的 L2 是 1MB/16 路，和 Zen4 相比大小没有变化，相联度为 Zen4 L2 的 2 倍，考虑到这点，L2 的命中率应该会增加，AMD 在 L2 的设计上和 ARM/Intel 这些厂商相比向来是比较小的，AMD 更加重视 L2 的访问延迟。但是坦白说，我自己的观点，L2 的容量是很重要的，延迟问题可以通过更好的预取，预测机制来覆盖，但现在尤其是服务器的工作集越来越大，L2 能覆盖更大的数据和指令范围真的非常重要，L2 如果 MISS，很多时候代价昂贵。

L1 和 L2 之间带宽增加到 512bit，并且优化了预取机制，官方介绍时新的 2D stride 预取。

1.9 Zen5 和 Zen4 比较



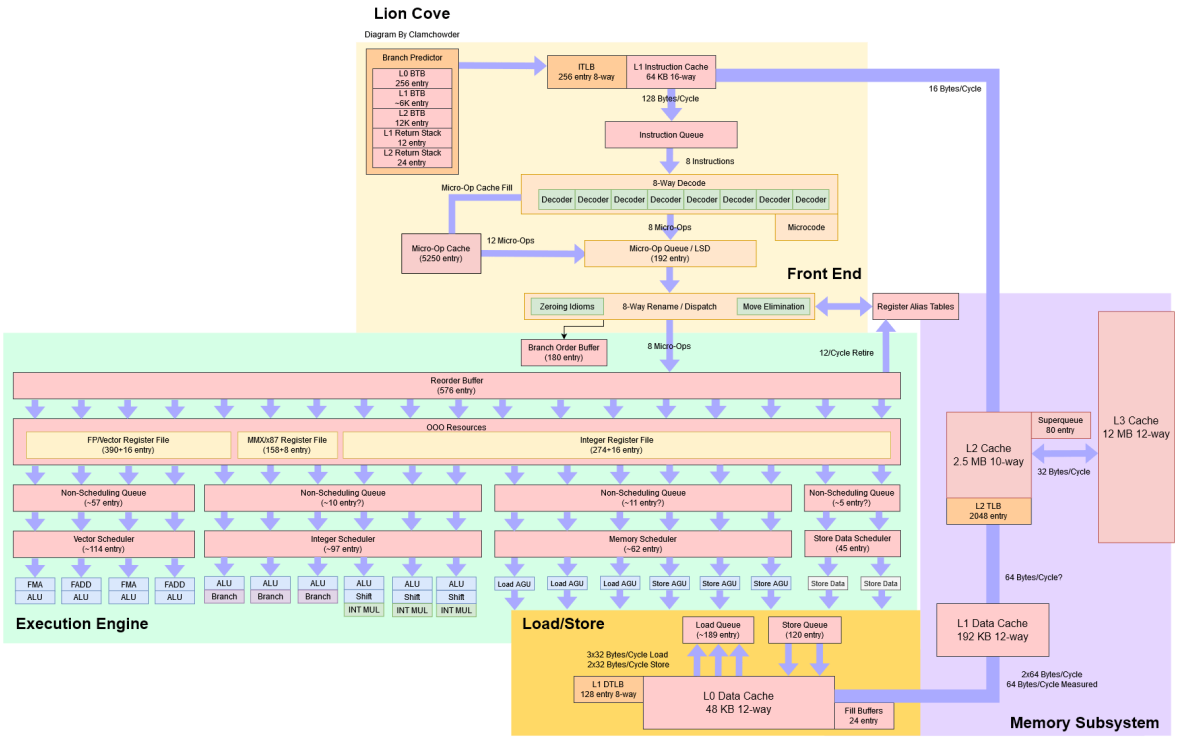
AMD 和自己架构的纵向对比，从 Zen4 到 Zen5 从参数上升级非常激进，像类似 TLB 量的升级基本走了 ARM 5 代的升级量，而某些部分的带宽升级 Zen5 是 Zen4 的两倍，L1 BTB 甚至是前代的 10 倍，有些地方的设计让人觉得没那么关注细节，这种升级也秉承了 AMD 设计的一贯“大力出奇迹”设计思路，尽管据测试显示，Zen5 有些方面在某些特定情况下性能甚至低于 Zen4，但如果能践行一些有趣的设计思路并且完善 Zen5 的设计缺陷，后面的几代 Zen 系列架构依然值得期待。

第二章　Intel Lion Cove 微架构设计

2.1 简介

今年 Intel 发布的微架构叫`Lunar Lake`，其 P 核（性能核）代号是`Lion Cove`，本文主要关注`Lion Cove`。相比较前代的微架构，本代改变很大，至少从技术上讲是的，可能出于 AI 的原因，以及来自 AMD，苹果，乃至 ARM 架构的冲击，Intel 这几年在努力改善自己的设计。

今年 P 核的重点是 AI，高达 18 个执行端口，12MB 的共享 L3，更宽度调度器，分离整数和向量调度器，还有让我惊讶的一点，预测器的宽度是前代的 8 倍，当然还做了一个违背祖宗的决定，取消了超线程。



和前代相比，`Lion`的发射宽度和`Decode`升级到 8，这也是当前各家旗舰 CPU 的主流参数。预测器的`L0 BTB`由前代的`128entry`升级到`256entry`，覆盖的指令范围变得更大，但是 L1 和 L2 的 BTB 并没有作容量上的升级，对于方向预测器亦安没有找到相关的资料。

`Micro-Op Cache`由`4096entry`升级到`5250entry`，相比较 AMD 的 Zen5 的 6K 略低。有意思的是，对于`DCache`在传统的 L1 和 L2 之间插入了一级`192KB`的 Cache，这是一个很有趣的设计，本文将`48KB`的`Cache`称为`L0`，`192KB`的`Cache`称为`L1`。`L2`的`Cache`由前代的`2MB`的大小升级为`最高 3MB`，目前主流的旗舰 CPU 是 2MB 大小。

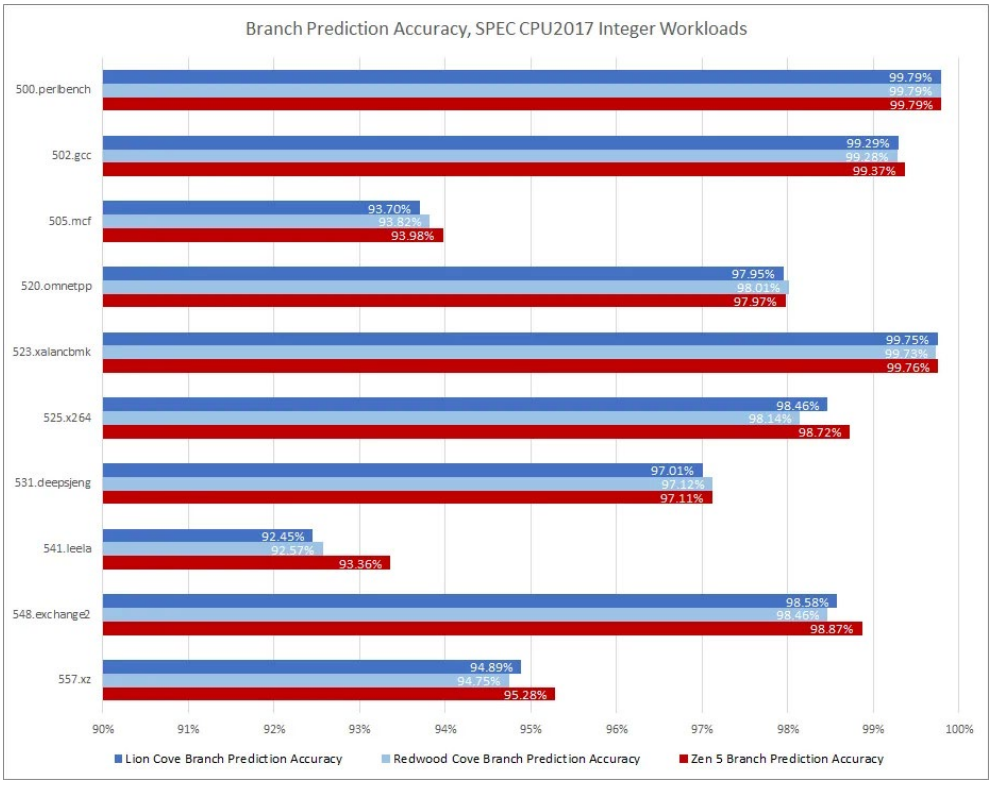
2.2 取指单元（IFU）

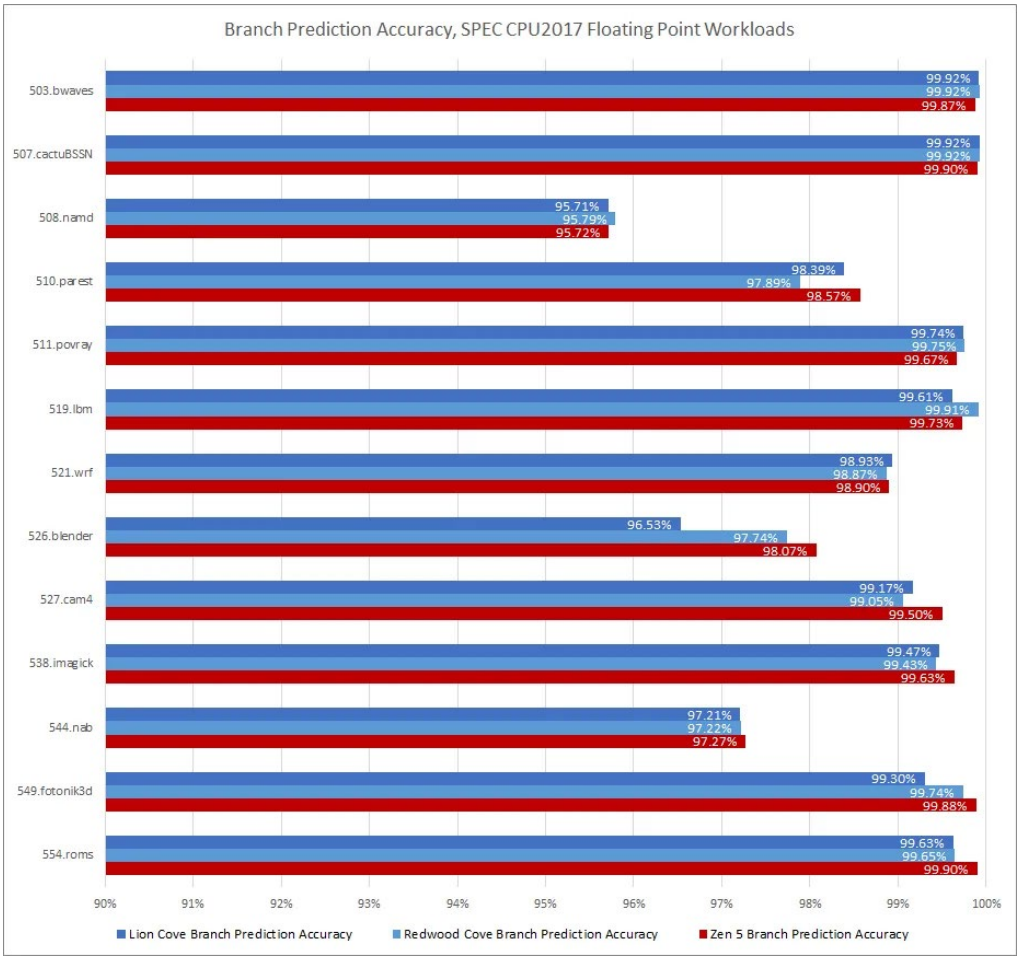
2.2.1 预测器

分支预测器向来是前端的核心，乃至整个 CPU 都属于影响性能瓶颈的模块，今年 Intel 对预测器宽度的描述，是前代的 8 倍预测宽度，从取指带宽也能看出 Intel 遥遥领先，前提是能吃满带宽，8 倍的预测带宽，猜测一下潜在实现方式：加大 block 的宽度，这样做会浪费资源，并且即使有软件优化，也很难将平均跳转的指令距离变大（单周期一个 block 的有效指令和跳转指令位置，block 长度，block 在 cache line 位置决定），所以这种方式属于比较激进的设计，能取出的有效指令很多时候都很少，但毕竟还可能会出现连续不跳转到指令流，所以这种设计不排除有收益，目前没见过公开宣称说自己是这种做法的，并且出于软件优化兼容的考量，block 大小应该不太会轻易修改。还有当然是一个周期能预测多跳转的分支，这很好理解，我在描述的 Zen5 预测器的文章中写过，如果只实现连续 block 的预测，应该很难实现如此大的预测带宽，所以 intel 的预测器如果不谈效率只谈参数，应该是最先进的（这有点耍流氓）。查了一下 Intel 的软件优化手册，对预测器的描述，向来只有“提升了预测器性能”。

预测带宽如果巨大，提前取指很远，可以很好的掩盖 L1 Cache MISS 的延迟，甚至可以掩盖 L2 Cache MISS 的延迟，不过除了预测带宽，我觉的，预测精度可能更重要，这涉及到刷新流水线的惩罚，以及可能带来潜在的 ICache 和 ITLB 的污染问题，当然污染主要是 ICache，不过这不太好评估影响。

从`Spec2K17`分支预测的测试结果看，`Lion`和前一代相比没有太多进步，甚至有的方面有退步，而相比较`Zen5`综合似乎稍微弱一点，因为没有看到比较多的预测器参考资料，不好评价，至于具体的算法没有找到资料，可以确定预测带宽很大。使用 3 级的 BTB，资源大小似乎和前代没有变化，预测方向的算法不清楚，按照行业的主流设计应该还是 Tage，不清楚具体的微架构设计与前代相比变化多少。


Branch Prediction Accuracy, SPEC CPU2017 Integer Workloads
500.perlbench
99.79%
99.79%
99.79%
502.gcc
99.29%
99.28%
99.37%
505.mcf
93.70%
93.82%
93.98%
520.omnetpp
97.95%
98.01%
97.97%
523.xalancbmk
99.75%
99.73%
99.76%
525.x264
98.46%
98.14%
98.72%
531.deepsjeng
97.01%
97.12%
97.11%
541.leela
92.45%
92.57%
93.36%
548.exchange2
98.58%
98.46%
98.87%
557.xz
94.89%
94.75%
95.28%
90%
91%
92%
93%
94%
95%
96%
97%
98%
99%
100%
Lion Cove Branch Prediction Accuracy
Redwood Cove Branch Prediction Accuracy
Zen 5 Branch Prediction Accuracy

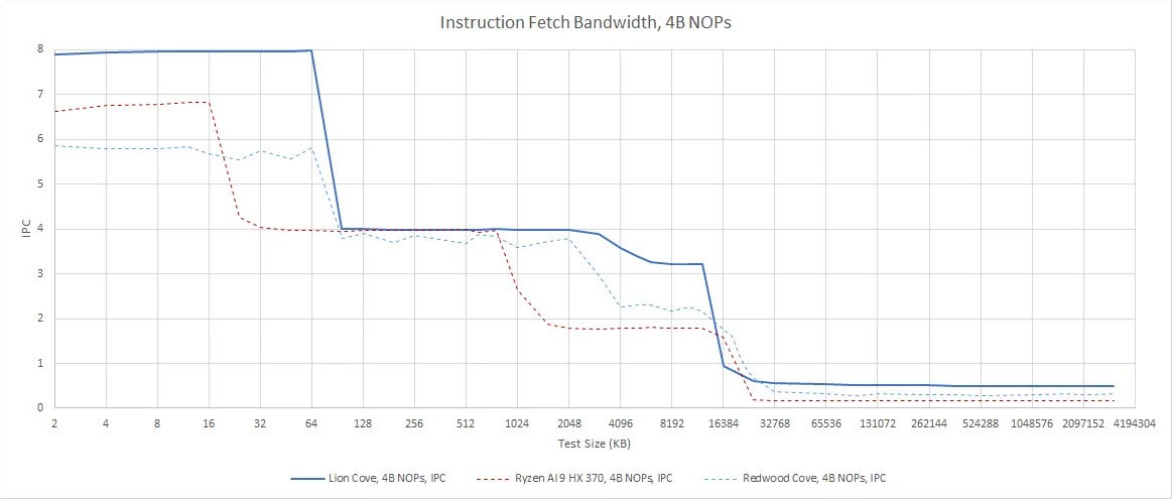

Branch Prediction Accuracy, SPEC CPU2017 Floating Point Workloads
503.bwaves
99.92%
99.92%
99.87%
507.cactuBSSN
99.92%
99.92%
99.90%
508.namd
95.71%
95.79%
95.72%
510.parest
98.39%
97.89%
98.57%
511.povray
99.74%
99.75%
99.67%
519.lbm
99.61%
99.91%
99.73%
521.wrf
98.93%
98.87%
98.90%
526.blender
96.53%
97.74%
98.07%
527.cam4
99.17%
99.05%
99.50%
538.imagick
99.47%
99.43%
99.63%
544.nab
97.21%
97.22%
97.27%
549.fotonik3d
99.30%
99.74%
99.88%
554.roms
99.63%
99.65%
99.90%
90%
91%
92%
93%
94%
95%
96%
97%
98%
99%
100%
Lion Cove Branch Prediction Accuracy
Redwood Cove Branch Prediction Accuracy
Zen 5 Branch Prediction Accuracy

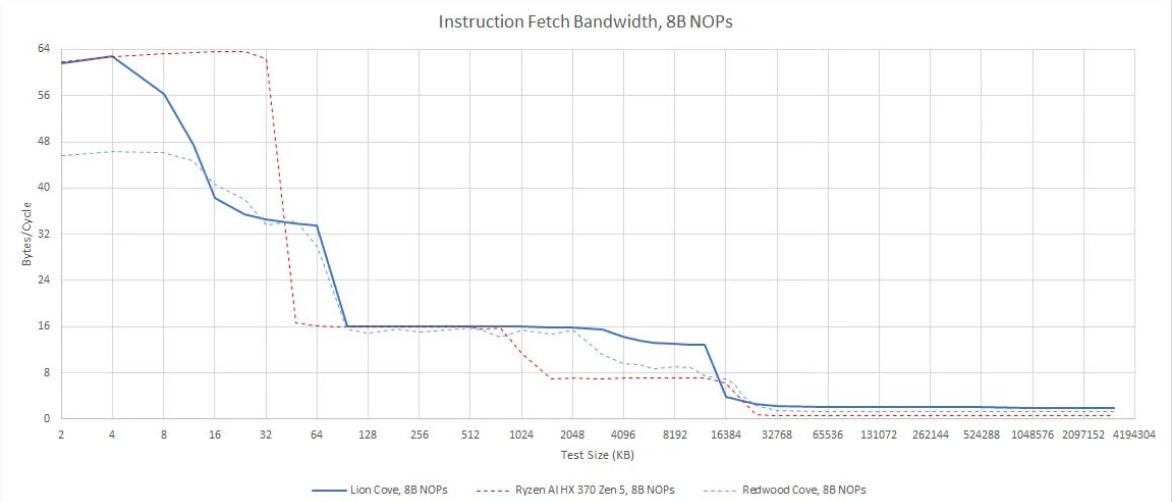
2.2.2 ICache/Op Cache

`ICache`是 16 路的`64KB`大小，取指带宽是`128Byte/Cycle`，其实这一点让我感觉困惑，当然我也没设计过这个模块，像 AMD 这些公司，64B 的带宽配的解码基本是 8 宽度解码，但 Intel 为什么是 128B 的带宽配置 8 解码宽度？`Mop-Cache`的带宽是 12 条指令，容量从`4096`增加到`5250`。`decode`是当前典型大核具有的 8 宽度，并且每个解码槽都可以为单线程提供服务。ICache 很有特色，取指带宽是前代的 4 倍，考虑到预测带宽是前代的 8 倍，预测器大概率设计比较激进，但精度没有特别大的影响。

根据 C&C 的测试，当使用 4B NOP 测试时，Lion Cove 在 64KB 的指令范围都能基本实现 8IPC 的，而到 L2 的范围，基本能实现 4 IPC，和 Zen5 不同，Zen5 的解码簇不能 2 个线程共享，所以这种情况下，Lion 实现的带宽更好。而超过 64KB 的 ICache 范围，带宽基本变到一半，一直到超过 L2 的容量，从曲线很容易看出

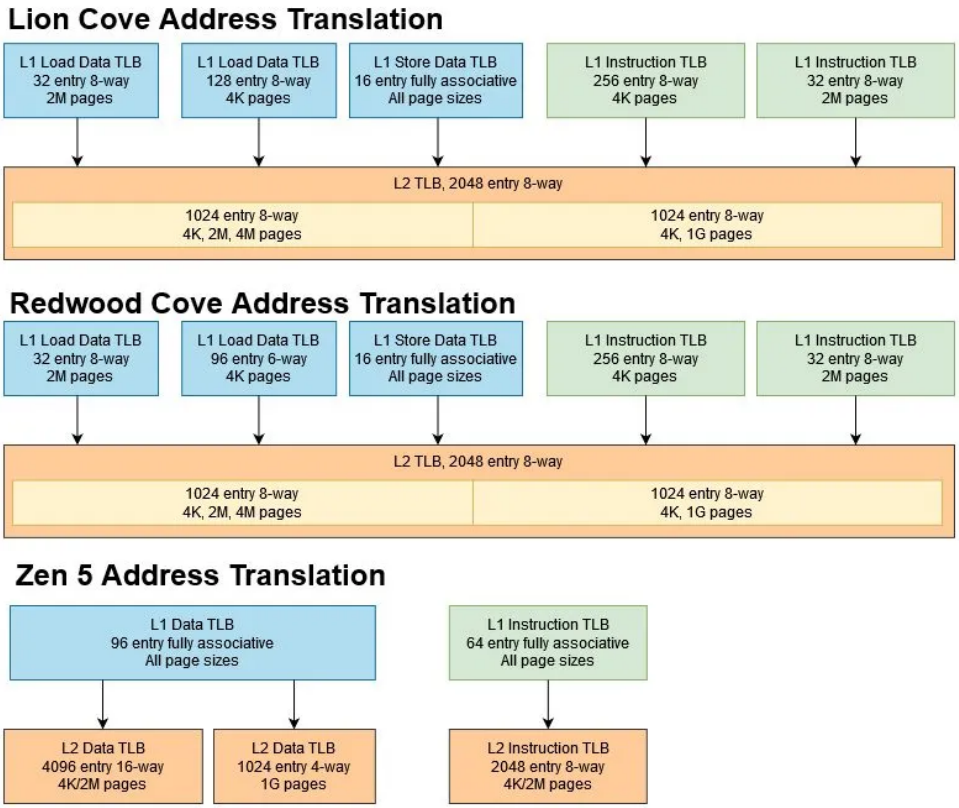
这种测试只是为了测试某些情况下的带宽，实际的指令运行场景比这个构造的测试要复杂的多。





2.3 内存管理单元（MMU）

对于 MMU 的考量，Zen5 和`Lion`的设计可以说天差地别（参数上），`Lion`的`L1 TLB`是`Load`和`Store`分开的，而 Zen5 是全相联的，这点设计我觉得没什么奇怪的，只要能接受延迟面积的因素，`L1 TLB`怎么设计都有道理。但同样是 X86 架构，AMD 的`L2 TLB`是`2048entry+1024entry+4096entry`，而`intel`则是`1024entry+1024entry`，常规的聚合类的技术大家都是有的，但参数上居然有如此巨大的差距？AMD 的巨大容量可以理解为它采用了非常激进的预取来掩盖页面翻译的延迟，但从 intel 的 L1 设计看，LS 常用的页面应该是 4KB 和 2MB，AMD 给 1G 的页面单独给出了`1024entry`巨大的 TLB 还是挺奇怪的，只能理解为 Zen5 想压缩 L2 的延迟以及它可能测试了大量的未来可能需要的 AI 应用？不管怎么说，intel 的 MMU 设计还是属于当前各家的主流设计范围内的，细节虽然不清楚，但 MMU 也不会太出花活。



2.4 解码（Decode）

8 宽度的 Decode，网上的测试博主结果显示，目前 Intel 的 Decode 能跑满带宽。

2.5 乱序（OoO）

乱序的有个比较特色的变化，之前一直使用的统一调度器现在拆分了，整数调度器是 6 个，向量调度器 4 个，和之前统一的 5 个调度器相比，是原来的 2 倍，不了解调度器的知识，不好评价，以后完善这部分。

2.6 访存单元（LSU）

从公开的资料看，此处的改变属于比较大的（执行单元和 BPU 没看到更多资料），在 L0 和 L2 之间插入了一个大小为`192KB`的`L1 Cache`（原本前代的`L1 Cache`对应本代的`L0 Cache`，只是个名字，也可以称为`L1.5`）。这种设计似乎脱离了各个厂商的主流设计`L1/L2/L3`，这就涉及到那个经典的问题“为什么 Cache 需要分级？应该分多少级呢？分级是怎么确定的？”。目前主流的厂商的 CPU L2 容量是 2MB，而`Lion`的 L2 容量“最高到 3MB”，我们知道 Cache 容量越大延迟越高，就设计水平以及使用的工艺都差不多的情况下，Intel 应该很难使用 3MB 的 L2 的情况下将延迟压到和其它厂商一致，所以这个所谓`L1（192KB）`的 Cache 应该是了掩盖 L2 大容量带来的延迟而作的设计平衡。当然增加带宽也是可能的原因。

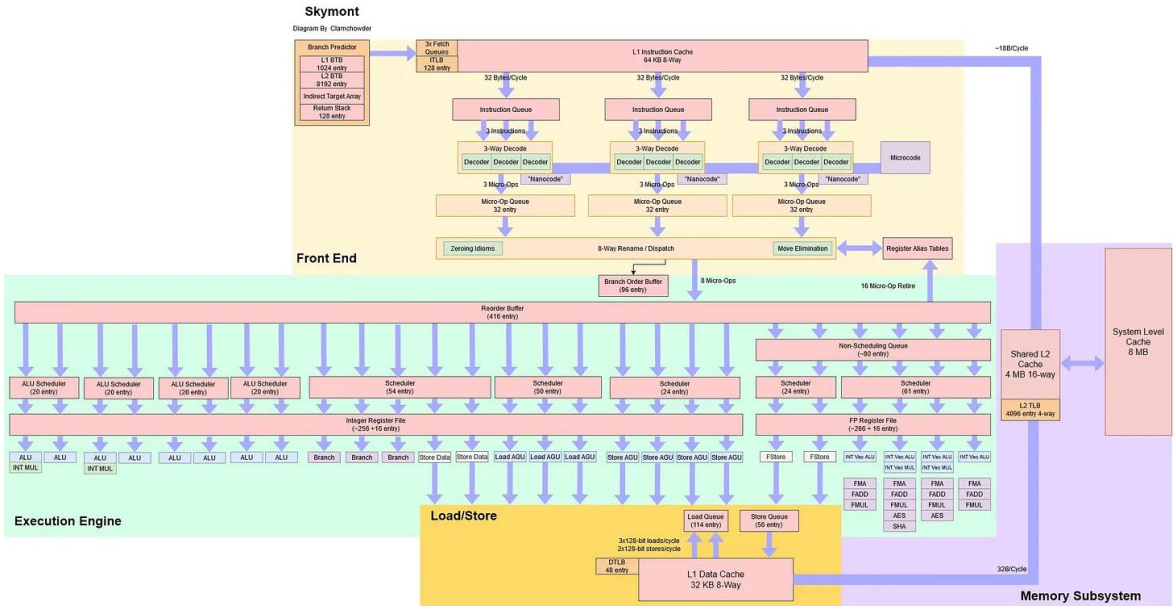
2.7 二级缓存（L2 Cache）

从 Intel 的各代微架构显示，相比较 AMD，Intel 更愿意加大 L2 的大小，同时平衡好延迟，命中率。

第三章　Intel Skymont 微架构设计

3.1 简介

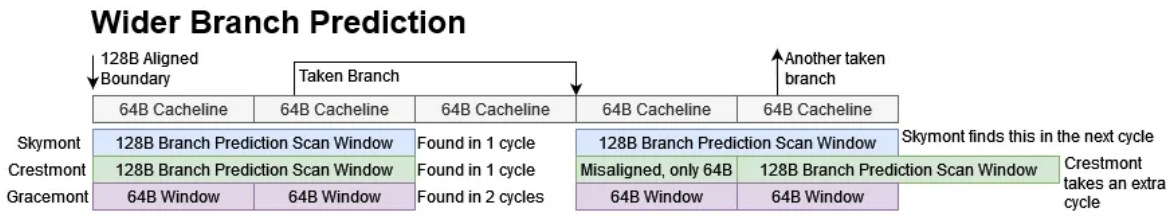
`C&C`对`Skymont`的测评标题是“`Skymont: Intel’s E-Cores reach for the Sky`”，看了下各个评测网站对`Intel`的本代`E`核（能效核）评价，基本都比较正面，本文主要关注一些微架构的设计。



3.2 取指单元（IFU）

L1 BTB`大小是`1024 entry`可以实现 0 气泡预测，而`L2 BTB`的大小是`6K entry`，一般是延迟是`3~4cycle`，相比较 ARM BTB 预测的延迟多一个 cycle。

`Intel`的分支预测应该是相当领先的，目前的各家旗舰`CPU`从`ICache`发送指令宽度基本都是`64B`，而`Intel`的`P`核是`128B`，`E`核是`3 cluster，96B`，非常大的取指带宽，当然不同指令架构的`block`宽度也会影响取指带宽，`block`宽度除了和架构的一些特性有关也和编译器优化有关，所以换句话说，取指带宽很宽也不一定是预测器非常先进，所以我说“应该”领先。当然我说取指带宽很大就先进也基于跑满带宽有效指令效率大家差异不大的前提。（有些基于参数的判断是有很多前提的，但由于文章篇幅的原因，我一般也都不会展开讲）



根据 C&C 的文章，本代的 E 核优化了预测，前一代预测窗口需要 128B 对齐才能一个 cycle 出结果，而本代则不需要，如上图所示。

从后端的分支执行数量以及取指带宽看，原本我猜测`Skymont`每个周期可以预

测 2 个跳转的指令，但有`C&C`说每周期不可以预测`2`跳转指令，3 个分支执行是为了执行更多的不跳转分支，他们信息应该更准。

至于`ICache`和前代一致`64KB/8WAY`，`Intel`和`AMD`对`ICache`“路”的设计普遍比 ARM 架构的核高，当然大家感兴趣可以查找资料这些设计权衡的原因，AMD Zen5 的文章中也有简述，在此不再赘述。E 核前端没有`uop-Cache`。

3.3 解码（Decode）

E 核比较有特色的是解码簇的引入，尽管解码簇 Intel 早在 2019 年就使用该微架构，但今 2024 年给出的改变是，使用 3 个解码簇而非原来的双解码簇，这更加消耗资源。

这是 C&C 对 Intel 架构师的采访：

“Why go with the 3 by 3 decode cluster?” To which Stephen said, “It was a statistical bet. And while three 3-wide decoders is a little bit more expensive in terms of the number of transistors then a two by 4-wide decode setup but, it better fits the x86 ISA. In x86 you usually see a branch instruction every 6 instructions with a taken branch every 12 instructions, which better fits the three by 3-wide decode setup Skymont has. A 3-wide decoder is also easier to fill then a 4-wide decoder and having 3 clusters is more flexibly then having 2 clusters hence why Skymont has three 3-wide clusters instead of two 4-wide clusters.”

只讲了为什么用 3 个解码簇，但没讲为什么用解码簇，最初因为每个解码簇都是 32B，很像一个 block（实际复杂指令集可能不是 8 指令一个 block，尽管没有查阅资料，但应该大于 10，也许设置为 12？），可能是分支预测器的原因，类似 AMD 的做法，但毕竟是 3 个解码簇，并且 intel 是 Simple Decode，而 AMD 则是 Complex Decode，自然推翻了这种猜想。查了一些网站的资料，然后查到 intel 有个专利：VARIABLE-LENGTH INSTRUCTION STEERING TO INSTRUCTION DECODE CLUSTERS 描述，*Embodiments of apparatuses and methods for variable-length instruction steering to instruction decode clusters are disclosed. In an embodiment, an apparatus includes a decode cluster and chunk steering circuitry. The decode cluster includes multiple instruction decoders. The chunk steering circuitry is to break a sequence of instruction bytes into a plurality of chunks, create a slice from a one or more of the plurality of chunks based on one or more indications of a number of instructions in each of the one or more of the plurality of chunks, wherein the slice has a variable size and includes a plurality of instructions, and steer the slice to the decode cluster.*

因为 X86 是变长指令，所以做了个设计将变长指令切片并且转换为类似 RISC 类

微指令（固定指令长度），来规避 X86 指令 Decode 的一些固有问题，当然还有很多调度和检测的功能，尽管听上去简单，实际这个设计是个复杂问题，并且这种设计的解码是乱序解码，打破我们常规的认知“前端是顺序的”，从而也会导致分支预测在整个设计中变得更为重要，因为预测错误的代价更加昂贵，感兴趣可以看下这个专利以及 Intel 对这个设计的软件优化建议。

3.4 后端（Backend）

Skymont 有 9 个 Simple Decode，Dispatch 是 8，Issue 高达 26，Retire 是 16，执行端口是夸张的 26，公开的信息除了增加解码簇以外，还吸引人注意的就是巨大的执行端口。前代的 Crestmont 微架构 Decode 端口是 6，Dispatch 端口是 6，Issue 是 17，Retire 是 8，执行端口是 17，除了 Dispatch，每项都提升很大。从直观上看，后端的相当多的参数设置均比较“失衡”，Dispatch 仅有 8 的情况下，Retire 设置了 16，同样的，为什么前端送入的指令宽度不大，却设置了 26 个执行端口，这种平衡也许和资源利用有关，或者 Intel 经过仿真将这些模块增大有利于某种性能，但需要增加的实际代价却不高？不管怎么说，亦安都是欢迎一些打破常规认知的设计。

3.5 二级缓存（L2 Cache）

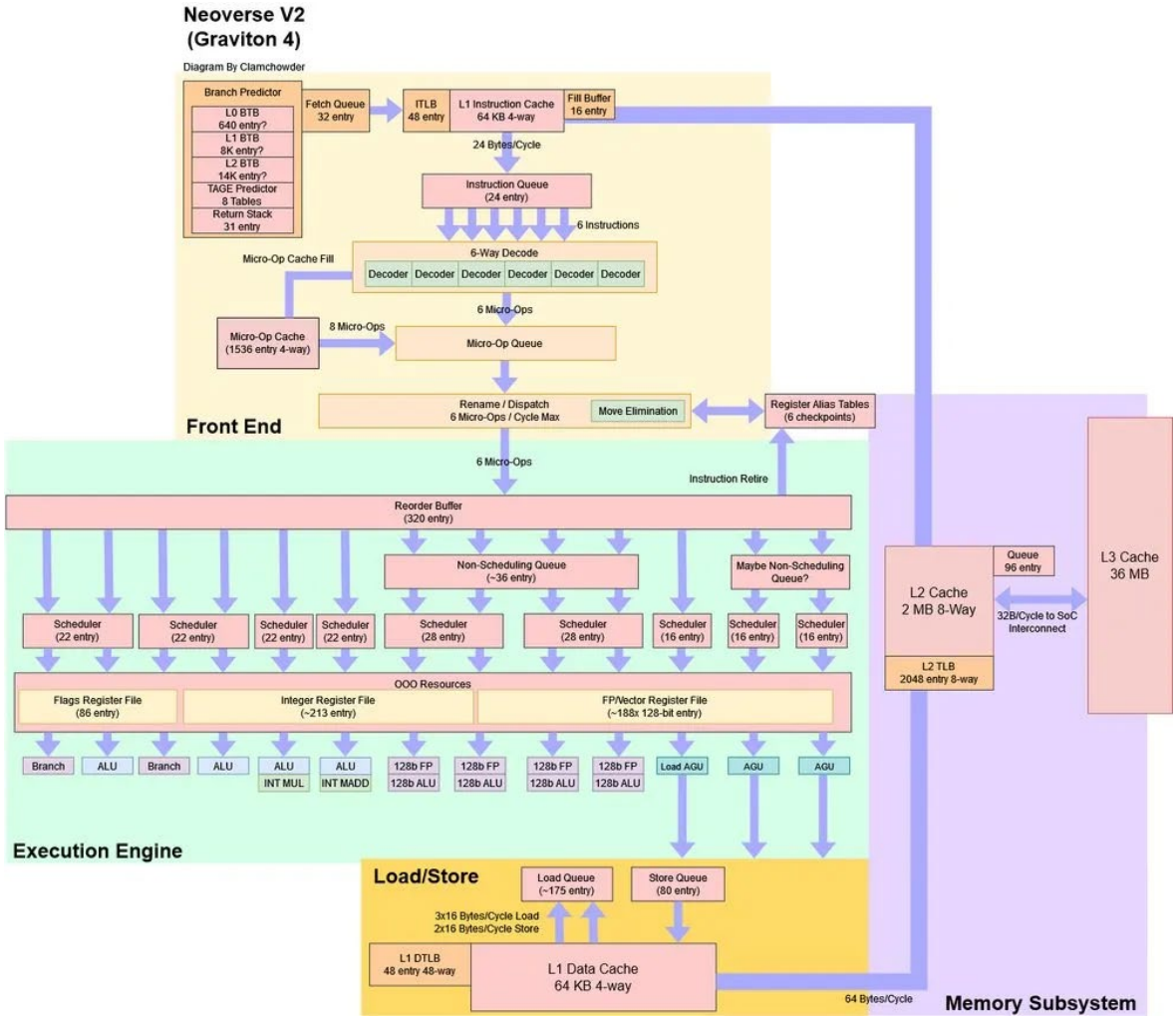
Skymont 具有 4MB 由 4 核共享的 L2，目前主流 CPU 设计厂商中，Apple 和高通采用共享 L2 的方案。Skymont 的共享 L2 可以提供 128B/cycle 的带宽，和前代的 E 核 Crestmont 相比带宽变为 2 倍，这是非常巨大的提升。

第四章　ARM V2 微架构设计

4.1 简介

Neoverse V 和 N 系列是 ARM 的服务器产品线，最初起源于 A76，V 系列对应移动端的超大核 X 系列，N 系列对应移动端大核 A 系列。2024 年初 ARM 已经发布 V3/N3 系列，到目前亦安写文章的时间为止，V3/N3 还没有看到实际已经商业化的产品，所以网上也没有相关的测试数据，本文将根据网络上的 V2 的一些测试数据以及 ARM 已经公开的数据分析 V2 的微架构设计。

下图是 ARM 官方公布的 V2 的微架构相关数据，目前 V2 采用 ARM V9 架构指令集，使用 48 虚拟地址和 48 物理地址。

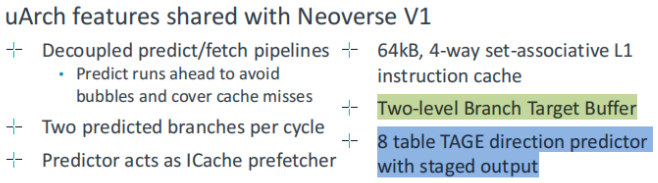


4.2 取指单元（IFU）

4.2.1 预测器

BTB 结构采用经典的 2 级结构（这是官方在 HotChips 的说法，C&C 说是 3 级结构），之前 ARM 的做法是，L1 BTB 可以无气泡定向（一般很小，我一般将 nanoBTB 称为 L1 BTB，现在 0 气泡的 BTB 也做到 K 级别了），L2 BTB 当检测到一些特定场景不需要复杂算法（路径延迟不大）则直接 bypass，这个延迟是 2cycle，而需要完整的算法定位使用哪个预测结果时，则需要花费 3 个 cycle，具体不详细说了，基于此一开始认为 C&C 的软件测试误将 L2 BTB 旁路的机制当成一级 BTB，但他的文章又说最后一级 BTB 延迟是 2-3cycle，说明他知道最后一级是有旁路机制的，很奇怪是怎么测的，并且他发文章的时候，ARM 早就公开了 V2 微架构。按照下图官方的描述只能理解为官方说的 2 级是 L1 和 L2，但最后一级由于功能问题又拆分了 2 级（我怀

疑这里所谓的 L2 拆分 2 级就是指旁路，ARM 有时候宣传就会神经兮兮的把一个业界常用的说法换成一个看起来高大上的词汇，我没有看当时 ARM 的演讲视频，不好确定），而 C&C 将这些理解为 3 级，说 2 级和 3 级都算合理，只需要知道常规的主预测器延迟是 2~3 周期即可，我们也不必太纠结，后续如果有详细的信息我会更新文档。（这里亦安也不确定，如果由比较确定的答案我会更新文档）



uArch features **new with Neoverse V2**

Branch Target Buffer	10x larger nanoBTB Split main BTB into two levels with 50% more entries
TAGE	2x larger tables with 2-way associativity Longer history
Indirect branches	Dedicated predictor
Fetch bandwidth	Doubled instruction TLB and cache BW
Fetch Queue	Doubled from 16 to 32 entries
Fill Buffer	Increased size from 12 to 16 entries
uOp cache	Reduced size for efficiency

而除了 BTB 的问题，预测方向的算法是业界常用的 TAGE-SC-L，据我所知，ARM 用了很多年的这个算法，并且预测的效果很不错，V2 使用的 8-table，这个表项长度很快就到一些论文所说的瓶颈长度了，也就是继续增加不会有比较高性价比的预测精度提升，印象中是 14-table，不过我感觉短期内 12-table 都不容易，代价很高。

间接预测器官方描述是 Dedicated predictor 专用预测器，直观理解是给出一个单独的间接预测器，当然这里的所谓专用不知道 BTB 是不是也存在一个专用的 IBTB，演讲没说估计按照一贯做法是没有单独的 IBTB。

关于预测 2-taken，我之前有文章写了，当时说是 V2/N2 刚引入这个技术，实际不是，V1 之前有这个特性的。ARM 目前为止没有透露太多的 2-taken 的技术，猜测是预测两个连续的 block，限制在一个 Cache Line 内能读出吧，考虑到分支跳转的通用的特性，提升应该有限，需要软件优化，以及微架构上将预测 2-taken 的约束逐渐去掉。之前移动端 X4 第一次上 10 Decode 那会就被测试出前端吐不出那么多指令，有点参数营销的意思，但不管怎么说，这个提升指令并行性还是目前 CPU 追求的重要目标之一。预测 2-taken 如果能做的很好，还是重要的一个特性。

其它涉及的是一些参数的提升，例如 FetchQueue 从 16 提升到 32。

科普 TAGE 技术点：

目前 TAGE 算法还是资源平衡的前提下能实现的精度最好的预测器，如果你关注分支预测器算法可以关注以下一些问题：

1. 选择的表长

2. 如何平衡资源消耗
3. 平衡延迟
4. 如何在前 3 前提下提高精度
5. ctl 的位宽选择以及管理
6. tag 的位宽选择
7. u 的位宽选择以及管理
8. 查找以及预测结果的选择算法优化
9. 如何更好的更细粒度的利用替代预测器
10. 更新和分配的机制，这里的更新分配也包含对 u 和 ctl 的管理

非常多需要关注的点，确定参数需要建模去测试，实际实现的时候需要平衡 PPA，如果对实际理论和实现感兴趣可以看看以下论文：

A. Seznec, P. Michaud,“A case for (partially) tagged Geometric History Length Branch Prediction”, Journal of Instruction Level Parallelism , Feb. 2006

A. Seznec“The L-TAGE predictor”, Journal of Instruction Level Parallelism, May 2007

A.Seznec, J. San Miguel, J. Albericcio“The Inner Most Loop Iteration counter: a new dimension in branch history”, Micro 2015, December 2015

A.Seznec,“TAGE-SC-L branch predictors”, 4th Championship Branch Prediction, June 2014,

A. Seznec, TAGE-SC-L Branch Predictors Again. 5th JILP Championship Branch Prediction (CBP-5), June 2016

4.2.2 ICache

ICache 一直都是前端非常关注点核心部件，V2 的 ICache 是 64KB，4 路组相联结构，Cache Line 是 64B，ARM 的很多代的核都是 64KB 4 路的设计，在分析 AMD Zen5 的文章里，我也说过原因，这体现 ARM 对功耗的追求，并且现在的很多厂商都在倾向于增加路数来增加性能了，不知道 ARM 会不会首先在服务器系列的 Core 上做出设计的改变。

ICache 校验采用奇偶校验，每周期可以出 6 指令，VIPT 的方式，VIPT 会有别名的问题，但如果设计恰当也可以避免这个问题，而使用 PIPT 在 index 前就需要拿到 PA，这无疑增加了流水线深度，这是极度关注流水线深度的 ARM 不愿意看到。如果大家关注 Zen5 和 Lion Cove 的 ICache 就会发现他们设计的路数很高，Lion Cove16 路，每路大小是 4K，Zen5 8 路每路大小也是 4K，这不是巧合，除了提高命中率，也可以很简单的处理 VIPT 别名的问题，这种权衡是很复杂的，和指令集的一些特性以及编译器怎么去优化都是关联的，设计 CPU 不是一个存粹的硬件问题。

4.2.3 Op Cache

Op Cache 这个微架构设计在高性能 CPU 设计中不是一个新鲜的技术，但服务器系列的 V2/N2 是首次引入，在移动端则是追述到 A77，首次引入该设计。而和 RISC 相对的复杂指令集，设计厂商采用该设计非常常见，例如 AMD 和 Intel 很早就引入该技术，主要是为了存储解码后的指令，当命中的时候可以绕过对 ICache 的访问以及绕过 Decode，对于复杂指令集，微架构的流水线较深，所以这个设计非常有意义。

ARM Macro-OP 共有 1536 entry，采用 4 路 skewed 组织结构，根据相关资料命中率可以到 85%，命中可以出 8 指令，不清楚有没有指令融合，命中率还是相当可以的，采用 VIVT，替换算法使用 NRU。

在 V3/N3 就删除了 Op Cache，在 V2 的微架构中，Op Cache 的 entry 数比前代就减少了，给出的理由是平衡能效，这里就能看出 ARM 并没有那么喜欢这个结构，并且按照 ARM 的一贯做法都是将一些解码工作放在回填 Cache 前，也就是将一部分 Op Cache 功能做入 L1 Cache，这样的好处是可以减少取指之后流水线工作压力，这是硬件设计中的一个很关键的思路，可以提前处理一些信息减少关键路径上的压力，以面积换时序和性能。

4.3 内存管理单元（MMU）

尽管 ITLB 和 DTLB 分别和 IFU/LSU 紧耦合，但还是将两者放入本节描述。TLB 使用经典的 2 级结构，指令侧和数据侧分开，L1 ITLB 使用全相联结构，DFF 构造，具有 48entry，缓存 4KB，16KB，64KB 以及 2MB 页面的地址映射。L1 DTLB 使用全相连结构，DFF 构造，具有 48entry，缓存 4KB，16KB，64KB，2MB，512MB 的页面。L1 TLB 主要能覆盖住常规情况下的访问即可。

L2 TLB 是 8 路 SRAM 的结构，共计 2048entry，这个参数是个很常规的设计，为了提升命中率，目前 L2 TLB 基本都是 8 路了。L2 TLB 是指令和数据共享的，现在非常多的微架构采用分开的设计结构，但 ARM 对资源利用真的很偏执，所以依旧采用共享的方案。L2 TLB 缓存 4KB，16KB，64KB，2MB，32MB，512MB，1GB 大小的页面映射，假如，这些页面在一段时间出现的概率很平均，会导致每次访问到大页面的延迟很大，但实际的程序不会这样，以典型 4KB 命中算一般访问 L2 TLB 的延迟是 3cycle（不计算 miss 本身发射和返回处理的时间）如果是更大页面的延迟会更大。像 AMD 的做法是多个页面分开存在 TLB 中，这样可以并行查找不同的页面，压低访问延迟，代价是功耗的增大。

手册描述有翻译缓存 MMUTC，这个应该是加速翻译的 Cache，也就是 Page Walk Cache，或者叫 Walk Cache Buffer，按照 ARM 以往宣传特性推测，ARM 没有说是单独的结构，应该就是混合结构。

L2 TLB 还有聚合和预取的一些设计，聚合一般是 4 虚拟物理地址聚合放入一个 TLB entry，预取一般就是简单预取机制，这里不再详述。

4.4 解码/重命名/提交（Decode/Rename/Dispatch）

解码模块我没那么了解，也没太多要写的，ARM 一贯会有预解码的习惯，所以 Decode 的流水线猜测不会很深，可能就 1 或 2 深度，Decode 宽度是 6，Decode Queue 深度从 16 增加到 24，Rename 是 8 宽度，总的 checkpoint 从 5 提升到 6，有利于从分支预测错误中快速恢复。这一功能在高性能处理器中很常见。

4.5 访存单元（LSU）

两个 load/store 混用的通道，一个 load 通道，每个周期可以处理三个内存访问，相比较 AMD/Intel 数据带宽略低。

DCache 依然是 64KB/4way 的设计，ARM 对这一设计非常执着，ARM 工程师对功耗的追求有种固有的偏执，好消息是 DCache 改变以往使用 PLRU，V2 采用 RRIP，至少在寻求 DCache 命中率上有追求。ARM 常用的替换算法，NRU/PLRU/RRIP，L1 Cache 使用 PLRU 更多，更重视 L1 Cache 的时候会牺牲更多资源在替换算法上。现在论文常讲的更“细粒度”的替换算法，在实际工程中见的更频繁了。例如初始化区分历史，将数据或者指令视作不等价等。简单讲，有一种观点是不全部强调命中率，更强调整体的性能，举个简单的例子，有些数据不命中，对其 miss 系统损失的代价更高，即使依据频繁访问原则“它”应该被踢掉，但由于“它”地位更高，所以不将“它”替换掉。或者有观点，识别数据本身的特性以及访问频率等情况综合去考量替换问题，这无疑会消耗更多的资源，对于路数更多的 L2 可能使用类似“细粒度”的替换算法收益更高。但现在 ARM L1 Cache 也开始逐步使用相对复杂的替换算法。

关于预取，从以往 ARM 核的设计来看，对预取的设计是比较精细化的，V2 也不例外，多个模块均使用了预取，对于预取引擎，ARM 官方演说 PPT 这样描述：Multiple prefetching engines training on L1 and L2 accesses：Spatial Memory Streaming，Best Offset，Stride，Correlated Miss Cache。当然相比较 V1 优化的点，例如将 PC 用在 L2 GSMS 的训练上。值得一提的是，本代 V2 相比前代提升影响因素最大的是各个模块的硬件预取。

4.6 二级缓存（L2 Cache）

L2 Cache 可以配置 1MB 和 2MB，使用 8 路组相联结构，和前代相比，在升级容量的同时没有增加 ld-to-use 的延迟，这是很优质的升级。

如果细心的观察会发现，服务器的 L2 和 ICache 是 includsive，在手册中 ARM 强烈推荐要打开 ICache 的一致性维护，ARM 做了 includsive 这种相对简单的机制来确保能便捷的通过 L2 来维护指令 Cache 的一致性，付出的代价是 L2 需要包含 L1 的内容，使之能额外缓存的内容变小，这是需要为一致性付出的一些开销，所以服务器系列的 L2 可以配的大小比对应的移动端和核更大，这也是核数量巨多服务器核和核数较小的移动端核重要的不同点和设计思路差异。其实除了 ARM 推出了 V/N 系列的服务器核，其互联系统也有面向移动端优化版本，这说明先进的 IP 厂商会在核心设计方向一致的情况下去为特定的领域去做对应的优化以迎合市场的差异化需求。当然，服务器核和移动端核的差异并非如此简单，即使是 ARM 也需要更多的时间去探索如何高效的接入设计的差异，毕竟服务器核原来是源自 A 系列的核。

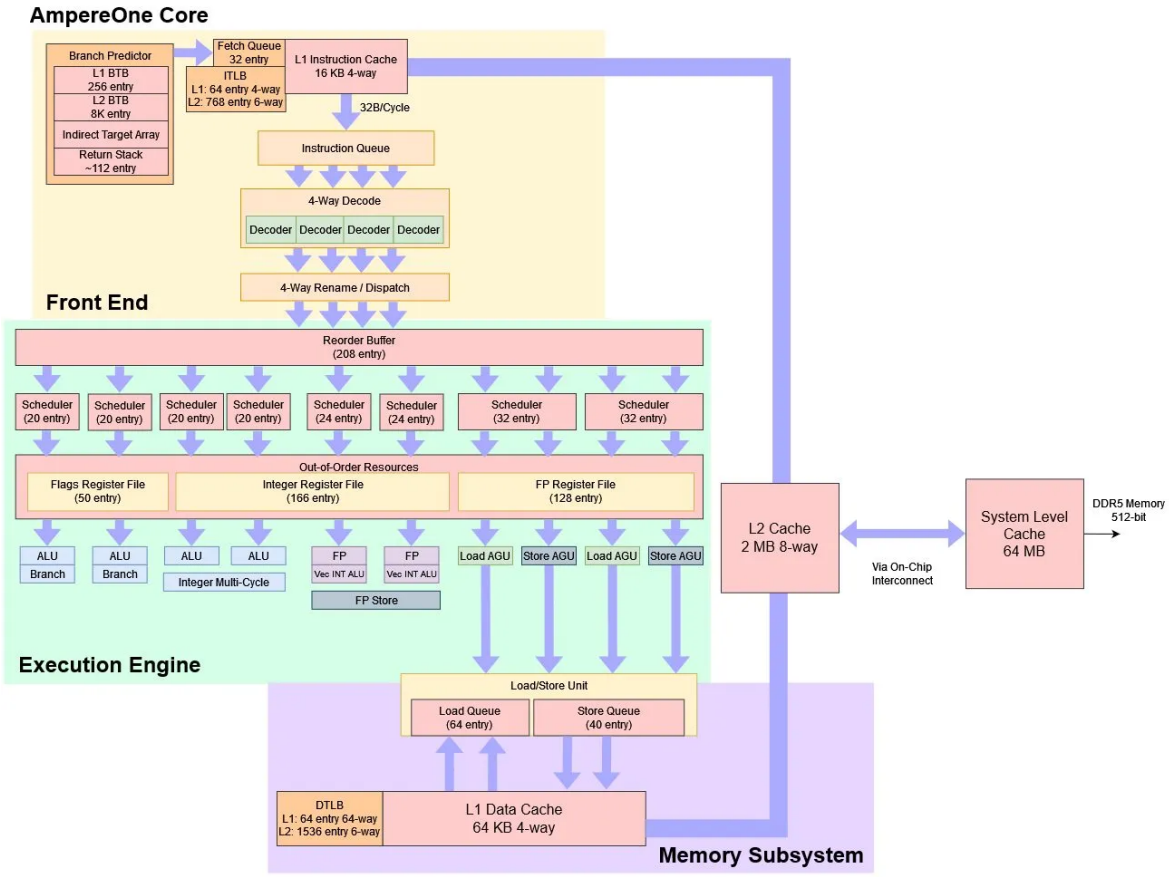
除此之外，值得聊的点，带宽：64B read or write per 2 cycles per bank = 128B/cycle total，这是官方的说法，因为是 4 个 bank，所以官方简单的相乘算出 128B/cycle 的带宽，这是比较极端的情况，每次都能顺利访问 4 个 bank，实际情况应该会差。

替换策略采用 6-state RRIP，L1 的 DCache 也是采用同样的算法，这个替换算法不是一个新技术，只是看设计人员是否在合适的场景下愿意付出较大的面积去获得性能上的提升。

第五章　Ampere AmpereOne 微架构设计

5.1 简介

Ampere 目前是一家专注于服务器设计的芯片研发公司，主要是 ARM 架构，目前该公司产品已经在一些大厂商用，本文主要简单分析一下该公司的`AmpereOne`系列的微架构，采用 ARM 架构，`5nm`工艺，最高`3.7GHz`，最多`192`核。各项参数都处在一线的水平，并且据说商业反馈比较好，所以就介绍一下这个系列的微架构。



5.2 取指单元（IFU）

前端直接预测器使用 8-table 的 TAGE 算法，目前 8-table 在大核中比较主流了。BTB 方面 L1 的 0 气泡是 256entry，没有说是什么结构，我猜测全相联可能性不大，但依然可能是 DFF 做的，也不排除 SRAM，这个核的工艺是 5nm，时序方面应该不错。L2 BTB 是 8Kentry，2-cycle 延迟，这里延迟保持的比较小。

分支预测精度方面和 Zen5 相比差距不大，仅在压缩场景和 Zen 有一定的差距。但请注意，这里的工作负载没有那么大，比较单一，数值仅能简单参考。Zen5 对预测

器投入的资源比较大，因为 Zen5 的 L1 BTB 是 16K entry 和 Ampere 的 256 相比不是一个数量级，所以预估在一些复杂场景，Zen5 优势会明显点。仅孤立的谈论预测器性能，Zen5 在几个场景表现更优秀一点。

间接预测器不清楚什么参数，但做了单独的结构。值得关注的是，分支预测惩罚是 10-cycle，这是目前我见过的大核中明确谈论的数值中最小的，这个参数对大核来讲是个挺重要的性能点的。之前聊的 SIFIVE（搞 RV 架构）在 PPT 的图中显示分支错误惩罚是 10-11cycle，但没有明确说。能将频率最高跑到 3.7，还能保持这么低的分支预测惩罚，说明设计水准较高。

指令侧是 16KB，4 路组相联结构（注意每路是 4KB，像我之前在 AMD 和 Intel 的文章中所说的原因），这个设计很有意思，这里相比较 ARM 的一些配置比较低，即使是它们自己另一款微架构 Altra 也是典型的 64KB 指令缓存。不清楚是预取做的太好还是 L2 延迟优化比较好的原因。还有可能是因为分支预测惩罚做的很低（10cycle），导致对 Cache 的访问无法接受更高的延迟，这里用更小的 Cache 来掩盖分支预测错误的惩罚延迟（压低流水线级），这种可能性也比较大。也可能都有，原因不孤立，总之是个设计的折中，最终也是看架构师想要什么目标。我更倾向于第二种原因是主要因素，因为从预测器结构来看架构师很重视分支预测惩罚的 cycle 数。目前很少看到大核做这么小的指令 Cache 了，分支预测器目前业内没有特别大的突破，改变一下思路可能豁然开朗。但是呢，如果指令 Cache 频繁 MISS 的场景，指令预取和 L2 延迟就太重要了，否则可能适得其反，总之一个不那么典型的设计需要更多的优化才能达到目标。不能因为降低分支惩罚延迟而导致前端成为瓶颈，这个核应该不会，因为前端瓶颈很好测出来，我个人认为这个设计应该比较成功，符合我对 CPU 微架构发展的预期。

5 路 decode 每个周期 4uop，带有指令融合技术，没有设计 uop cache，这里的流水线目标已经很低了，不选这个结构也挺合理的，一般能加大带宽或者这附近流水线可以降低到很低水平，uop cache 都没有太大必要，ARM 放弃 uop Cache 是带宽足设计的足够，同时又平衡了流水线深度，并且 ARM 有预解码的设计，所以解码流水线很浅，Ampere 这个应该是流水线做的很紧凑，完全没必要加这个结构。目前 RISC 的一些微架构都很倾向于降低流水线，尤其是分支预测错误惩罚的流水线，代价是频率相比 AMD/Intel 的深流水线低。

5.3 执行单元（EXU）

8 个调度器分配给 12 个执行流水线，2Load，2Store 流水线，这里没有做过相关内容，仅从几行文字看不出好坏。

5.4 访存单元（LSU）

LSU 看这个参数和 ARM 的 V2/V3 差点意思，64KB 的 4 路组相联，数据带宽似乎做的一般。有外网描述：在 LOAD 包含在先前 STORE 中的情况下，可进行快速转发（延迟时间为 6-7 个周期）。STORE 的延迟大概 6-7 周期，向量 STORE 大概 12 周期，这些延迟都有点高。在内存安全性上例如内存 TAG 在此微架构上也有应用。

5.5 内存管理单元（MMU）

这个模块的微架构之前写过不少文章都有提及，本次简略写。L1 ITLB 是 64entry，4 路，L1 DTLB 是 64 全相联。L2 TLB 指令侧和数据分开，从版图上看，MMU 都是分为数据和指令 2 个不同部分。最多有 8 个同时运行的 page walk，这个配比也是相对常规的，目前至少也是 4 个同时的 page walk。获取 PTE 是直接从 L2 而不是从 L1 来避免占用 Cache 空间。具体的一些参数大家参考下表，关于 MMU 的微架构设计我在多个文章都描述过相关设计的权衡。

5.6 二级缓存（L2 Cache）

二级缓存目前看到的大核常见的配置是 2MB，延迟 11 个周期，每周期可以同时执行一次读写。可以一个周期回填 64B 到 L1。其它参数大家参阅图片。

参考

1. Multiple-Block Ahead Branch Predictors
2. AMD 官网 https://www.amd.com/en.html
3. Chips and Cheese 网站 https://chipsandcheese.com/
4. HotChips 官网 https://hotchips.org/
5. VARIABLE-LENGTH INSTRUCTION STEERING TO INSTRUCTION DECODE CLUSTERS