## 微架构性能分析方法-从上到下的系统架构分析方法

现代 CPU 大多具有性能监控单元（Performance Monitoring Unit, PMU)，用于统计系统中发生的特定硬件事件，例如缓存未命中（Cache Miss）或者分支预测错误（Branch Misprediction）等，这种硬件事件有数百个，在发现和解决性能问题时，不可能每个事件都去分析，这样会做很多无用功，因为一个特定程序的性能问题可能与大多数硬件事件关系不大，所以需要有一种更行之有效的方法进行分析。

自顶向下的微体系架构分析方法（Top-Down Microarchitecture Analysis Method, TMAM）可以在乱序执行的内核中识别性能瓶颈，其通用的分层框架和技术可以应用于许多乱序执行的微体系架构。

### 流水线slot（槽）与stall（停顿）

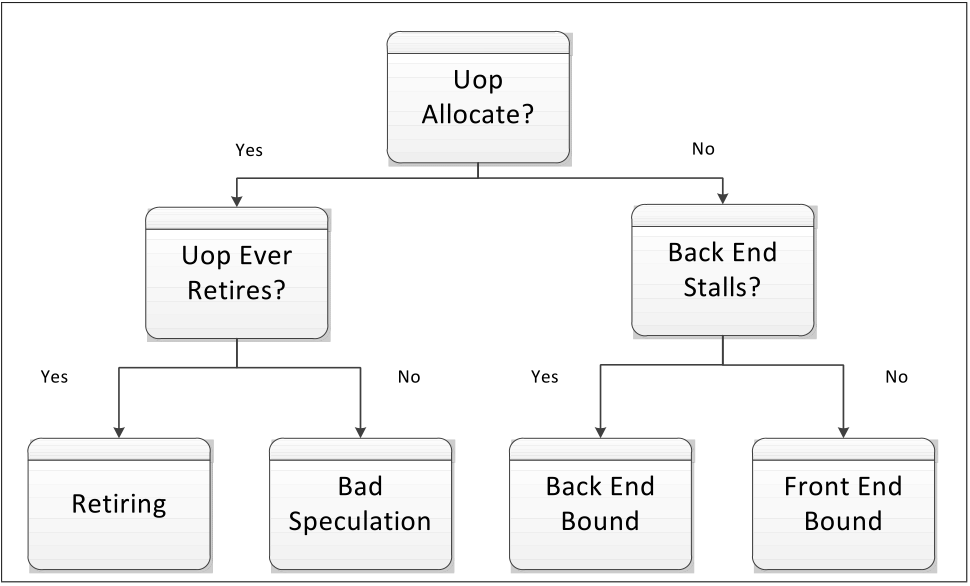
流水线槽（pipeline slot）代表处理一个 uop 所需的硬件资源。TMAM 假定一个cpu核心一个周期有4个pipline slot可用，如果有一个slot未被uops填充，也就是说未被利用，这就被看做是一次停顿。流水线中不同部件都可以有各自的pipeline slot，比如前段和后端可以各自有一个pipeline slot，测试他们的slot停顿数以确定程序的性能瓶颈主要在哪个方面。继续细分到各个部件，比如Loop Stream Detector(LSD)和Decoded I-cache (DSB)，它们也有自己的pipeline slot，我们也可以分析它们的slot停顿数。

### TMAM分类方法

除了上面提到的slot停顿数可以分析程序性能的瓶颈在哪里，分析那些已经被uops填充的slot也能有帮助。

在最顶层，pipeline被分为4个主要类别：Front-end Bound，Back-end Bound，Bad Speculation，Retiring。根据uops是否分配分为两大类：Front-end Bound，Back-end Bound与Bad Speculation，Retiring这两类。对于未分配uops的前者，根据是否是后端停顿继续划分成Front-end Bound与Back-end Bound；对于已经分配uops的后者，根据uops是否正常退出分为：Bad Speculation与Retiring。

根据一次一次判断而缩小范围，有选择地探索可能的性能瓶颈区域，使得我们能够将分析的重点放在确实重要的问题上，同时无视那些不重要的问题。



图TMAM的自顶向下流程图

### Front-end Bound

在高IPC情况下，Front-end指令带宽可能会影响性能，一些专用单元被引入（如Loop Stream Detector (LSD) 以及 Decoded I-cache (DSB)），用来隐藏流水线 Fetch 指令延迟以及维持所需的带宽，fetch指令延迟的原因有ICache miss，Fetch Bandwidth问题表示指令解码器的低效。所以Front-end问题被进一步分为Fetch Latency和Fetch Bandwidth。

还有一种Fetch Latency 叫做BranchResteers，Branch Resteers 表示流水线刷新（pipeline flush）之后的指令提取延迟。从预取单元到微指令队列这个过程有几种不同的解码器，普通的指令解码器，MSROM解码器。（64-ia-32-architectures-optimization-manual的附录B.1.2，搜索”The methodology further classifies”）。

### Back End Bound

除法单元过载和数据cache未命中会导致Back End Bound，体现在pipeline slot没有被交付uop，如果在稳定状态下只能执行三个或更少的 uOps，就不能达到最佳状态，即 IPC 等于4，这些次优周期称为 Execution Stalls。Back-end Bound 分为 Memory Bound 和 Core Bound。Memory Bound的真正代价是调度程序没有其他准备好的uop给执行单元，后面的 uOps 可能正在等待进行中的内存访问。Core Bound对应于执行单元的压力或者程序中指令集并行的缺乏，Core Bound停顿可能表现为较短的执行饥饿周期或者执行端口利用率不佳，比如连续的除法指令只使用特定的端口，而该端口一直在处理除法指令，虽然该端口很忙但是并不能达到很高的带宽。

### ****Bad Speculation****

Bad Speculation 表示由于不正确的预测而浪费的 pipeline slot，主要包括两部分：

1. 执行了最终不会被提交的 uOps 的slots
2. 从错误预测中恢复而导致流水线被阻塞的slots

### ****Retiring****

理想情况下，我们希望看到所有的 slots 都被标记为 Retiring 类别。尽管如此，Retiring 比例高并不意味着没有更多的性能提升空间。微码序列，如Floating Point (FP)，通常会损害性能，可以避免。

### 主要内容

1. PMU是什么
2. 什么是从上到下的系统架构分析方法，TMAM
3. TMAM顶层的四种bound

## 前端

CPU内部指令处理单元（execution unit）以及端口（port）增多，在Pentium 4的时候，发出到Execution Unit的μops的throughput可以高达6（6 μops/clock cycle），这时候，流水线中的瓶颈会出现在register renaming（RAT）以及retirement（RRF），这两部分的throughput为3。为了突破这部分的瓶颈，Intel从Pentium M处理器开始引入了micro-fusion技术。这个融合后的微操作在流水线的大多数处理过程中只是用一个微指令的资源，如ROB中只占用一个条目，但是这个条目必须要放到两个不同执行单元去执行即融合后的ROB条目被分派到两个不同的执行端口，退出时作为一个单元退出。

参考： <http://www.agner.org/optimize/microarchitecture.pdf>（在7.6 Micro-op fusion中）

### uop fusion应用场景

在register renaming（RAT）和retirement（RRF）阶段，fusion-uop一般不会被分解，而是直接被当成一个指令处理，而在EU阶段，复杂μop会被多次发送到EU中进行处理，表现得像是有多个已被分解的μops一样。uop fusion技术只被应用于两种类型联合：内存写，读修改操作。可以被micro-fused的指令有如下：

1.写回内存的store指令 a两步

2.读内存和运算的混合指令

3.读内存和跳转的混合指令

<https://easyperf.net/blog/2018/02/15/MicroFusion-in-Intel-CPUs>

### 两种fusion

前面说了pipeline中的两个部件吞吐量只有3条微指令，为了突破该瓶颈，设计者将前面分割的指令在经过这两个部件的时候合并处理，只不过两者融合对象不一样：

1. Micro-fusion：同一个汇编指令的微指令被整合成一个微指令
2. Macro-fusion：来自不同的汇编指令的微指令被整合成一个微指令

<https://easyperf.net/blog/2018/02/04/Micro-ops-fusion>

https://cloud.tencent.com/developer/article/1358004

## Lagacy Decode Pipeline（传统解码流水线）

由ITLB，Icache，instruction predecode，instruction decode units组成，

### Length-changing prefixes 变长前缀与Instruction PreDecode

PreDecode处理指令并决定每个指令的长度，处理的指令包括LCPs指令，predecode unit 接收从指令cache取得16bytes，然后决定指令长度，length changing prefixes (LCPs) 意思是指令长度不同于默认长度的指令，这种LCPs会造成开销损失，在长度解码阶段，一个LCP大概花费3个时钟周期。

## 参考资料

从上到下的系统架构分析方法-Intel PMU <https://cloud.tencent.com/developer/article/1442444>

[Intel微处理架构优化手册](Intel微处理架构优化手册https://software.intel.com/sites/default/files/managed/9e/bc/64-ia-32-architectures-optimization-manual.pdf)<https://software.intel.com/sites/default/files/managed/9e/bc/64-ia-32-architectures-optimization-manual.pdf>

Microbenchmarking fused instruction

<https://easyperf.net/blog/2018/02/04/Micro-ops-fusion>

MicroFusion in Intel CPUs <https://easyperf.net/blog/2018/02/15/MicroFusion-in-Intel-CPUs>

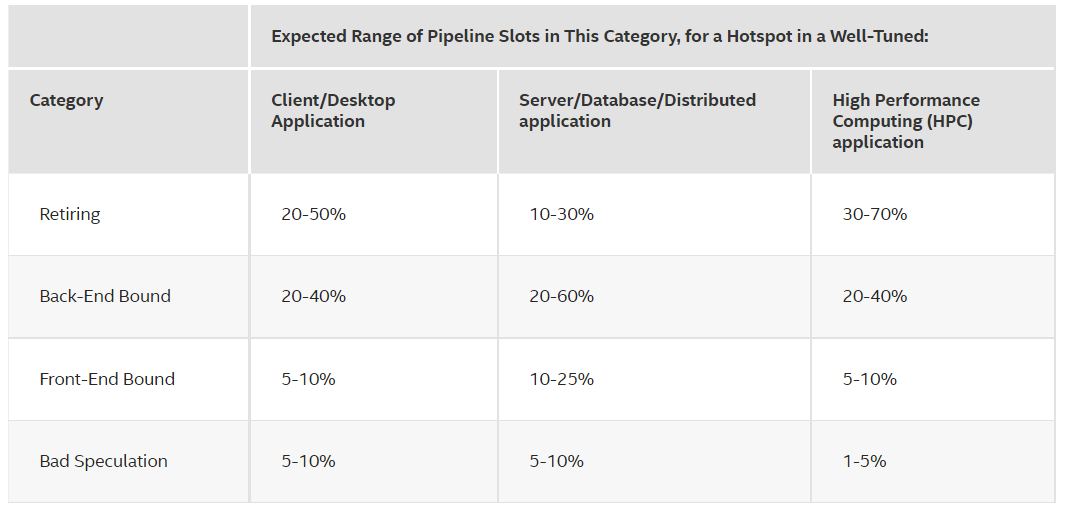
1. gcc编译器，面对的微架构是什么；
2. 针对的优化策略是什么；
3. 有自己的测试用例；
4. 通用的测试用例；
5. 在这个基础上，其他的开源设计代码；
6. 做成什么微架构，gcc支持的微架构，通用验证的效果。

## 微架构优化方法

r <https://software.intel.com/content/www/us/en/develop/documentation/vtune-cookbook/top/methodologies/top-down-microarchitecture-analysis-method.html> k Microarchitectural Tuning Methodology（标题）

To obtain maximum benefit from microarchitectural tuning, ensure that algorithmic optimizations such as adding parallelism have already been applied. Generally system tuning is performed first, then application-level algorithm tuning, then architectural and microarchitectural tuning最大获取微架构优化时，确保使用了并行；首先考虑通用系统优化，然后是应用层面的算法，最后是微架构优化

VTune Profiler如果超出预定阈值，并且在热点中发生，则会在GUI中自动突出显示度量值。VTune Profiler认定超过所有时钟周期5％的函数为热点。可以根据实际的工作负载确定特定类别的给定分值是否构成瓶颈，但对于通用的程序，可以参考下面指标。



如果程序的各项特定种类的指标在上面给定的范围内，则说明该程序是经过良好优化的。这些阈值基于对英特尔实验室的一些工作负载的分析。

## Tune for the Back-End Bound Category

The majority of un-tuned applications will be Back-End Bound. Resolving Back-end issues is often about resolving sources of latency, which cause retirement to take longer than necessary.大多数未优化的应有都是后端限制，解决后端的问题经常是解决延迟的来源，这些延迟会导致retirement超过必要的时间。

Memory Bound 对应缓存和内存子系统相关的 Execution Stalls。这些停顿通常表现为执行单元在短时间内饥饿，例如 load 操作没有在缓存中命中。（下面这一句的翻译）The Memory and Core Bound sub-metrics是Back-End Bound metrics下的。