


# 一颗打破常规全新的RISC-V芯片 -Cuzco 核心概述

嵌入式系统专家之声 2025年09月13日 08:00

以下文章来源于半导体行业观察公众号



半导体行业观察

半导体深度原创媒体，百万读者共同关注。搜索公众号：半导体芯闻、半导体产业洞察， ...  
3930篇原创内容

公众号



来源：内容编译自chipsandcheese。

Condor Computing 是晶心科技的子公司，致力于开发可授权的 RISC-V 内核，其商业模式与 Arm（该公司）和 SiFive 类似。晶心科技于 2023 年成立 Condor，因此 Condor 在 RISC-V 领域相对年轻。然而，晶心科技在 Condor 成立之前就拥有 RISC-V 设计经验，并在过去几年中开发了一些 RISC-V 内核。

Condor 将在 Hot Chips 2025 上展示其 Cuzco 核心。该核心是 RISC-V 领域的重量级产品，拥有广泛的乱序执行功能、先进的分支预测器以及一些新的基于时间的技巧。它与 SiFive 的 P870 和 Veyron 的 V1 等高性能 RISC-V 设计处于同一级别。与这些核心一样，Cuzco 应该会比目前已投入硅片的 RISC-V 核心（例如阿里巴巴 T-HEAD 的 C910 和 SiFive 的 P550）更胜一筹。

### Cuzco Processor Summary

- Innovative, time-based O-O-O core designed for high performance application processors
- Support for up to 8 cores with private L2s in a coherent cluster with a shared L3
- Much better performance than other high performance licensable CPUs with similar power consumption
- Latest RISC-V profile support (RVA23) for maximum software compatibility
- Full support for ISA customization

除了采用宽乱序设计外，Cuzco 还在后端主要使用静态调度来节省功耗并降低复杂性。Condor 称之为“基于时间”的调度方案。我稍后会详细介绍，但需要注意的是，这纯粹是一个实现细节。它不需要修改 ISA 或编译器进行特殊处理即可获得最佳性能。

## 核心概述

Cuzco 是一款 8 位宽的乱序核心，拥有 256 个 ROB 条目，在台积电 5nm 工艺上，目标时钟速度约为 2 GHz SS（慢速-慢速）至 2.5 GHz（典型-典型）。该流水线从指令提取到

数据缓存访问完成，共有 12 个阶段。然而，10 个周期的错误预测惩罚可能更准确地描述了该核心相对于竞争对手的流水线长度。

## Cuzco Core Pipeline

IF0	IF1	IF2	IF3	ID0	ID1	ID2	EX0	EX1/DC1	DC2	DC3	DC4
Calculate next address & access ILLB	Access IC tag array, IC hit/miss, BTB, and GHT, TAGE-SC-L	Access IC data array access, BTB hit/miss and taken/not-taken	Write IC cache line to ICQ & bypass/read N instructions to XIQ	Read N insts from XIQ, 1 <sup>st</sup> instr. decode, & access RFL & RAT	Access RSB to calculate execution times, 2 <sup>nd</sup> (read_time and read_time+1) for each instruction	Access TRM to issue instruction, secondary instruction decode, write issued instructions to XEQ	Read RF/rdvd data, check RSB, send instr from XEQ to functional unit	Execute instrs, write result to PRF, AGU and access dTLB	Access DC tag array (DC hit/miss)	Access DC data array	Align and send load data to write back to PRF

作为授权核心，Cuzco 旨在实现高度可配置性，以拓展其目标市场。该核心由数量可变的执行片构成。定制选项还包括 L2 TLB 大小、簇外总线宽度以及 L2/L3 容量。Condor 还可以调整各种内部核心结构的大小，以满足客户的性能需求。Cuzco 核心被排列成簇，最多可容纳 8 个核心。簇通过 CHI 总线与系统连接，因此客户可以自带片上网络 (NoC)，通过多簇设置实现更高的核心数量。

### Cuzco Feature Overview

- 64-bit, RV64GCBKV\* + CMO
- RVA23 profile compliant, with Hypervisor
- Innovative time-based microarchitecture
- 12-Stage Pipeline
- 8-Wide Frontend Decode
- 256-Entry Reorder Buffer (ROB)
- 8 Execution Pipelines
- RISC-V Vector 1.0 + Vector Crypto, 512b VLEN
- 2-Level Branch Target Buffer (BTB)
- TAGE-SC-L & Tournament Branch Predictor
- 1K/2K/4K 4-Way L2 TLBs
- 64 KB, 8-Way Private I/D Caches
- Up to 8 MB, Private L2\$
- 8-Core Multiprocessor w/ Shared L3\$ (up to 256MB)
- 256/512-bit CHI and 64b/512b MMIO Buses

The diagram illustrates the Cuzco core architecture. At the top, an Interrupt Controller and Debug Support block are connected to the core. The core itself consists of multiple Cuzco Core units, each with Private L1/2\$ caches. These are connected to a Coherence Manager (CM) and a Shared L3 Cache. The system interfaces include MMIO (AXI) and Memory/Coherent IO (CHI). A watermark 'Copyright © 2025 at 10:25 AM' is visible across the diagram.

4

© 2023 Condor Computing, Inc. - An Andes Company. All rights reserved.

CONDOR

Using RISC-V to New Heights

## 前端

Cuzco 的前端始于一个复杂的分支预测器，这对于以任何合理性能水平为目标的现代核心来说都是典型的。条件分支通过 TAGE-SC-L 预测器处理。TAGE 代表“标记几何”，这是一种使用多个表来处理不同历史长度的技术。它力求通过为每个分支选择最合适的历史长度来高效利用分支预测器的存储空间，这与使用固定历史长度的旧技术截然不同。SC（统计校正器）部分负责处理 TAGE 无法良好运行的一小部分分支，如果发现 TAGE 在某些情况下经常出错，则可以反转预测。最后，L 表示循环预测器。循环预测器只是一组计数器，用于对执行了一定次数但一次未执行的分支进行计数。如果分支预测器检测到这种循环行为，循环预测器可以避免在循环的最后一次迭代中出现错误预测。基本上，TAGE-SC-L 是基本 TAGE 预测器的增强版本。

AMD 的 Zen 2、Ampere 的 AmpereOne 和高通的 Oryon 也使用了某种类型的 TAGE 预测器，并实现了出色的分支预测精度。AMD、Ampere 和高通也可能以某种方式增强了基本的 TAGE 预测策略。Cuzco 的 TAGE 预测器的性能取决于其历史表的大小以及预测器的调优程度（索引位与标签位的选择、历史长度、TAGE 表间存储预算的分配等）。就 Cuzco 而言，他们披露了 TAGE 预测器的基础组件使用了一个 16K 的双峰计数器条目表。

## 5

## 8

与此类似的是 Nvidia 在 Kepler 及其后续 GPU 架构中的静态调度。两者都通过告知指令未来执行一定数量的周期来简化调度，而不是让硬件动态检查依赖关系。但由于 GPU ISA 尚未标准化，Nvidia 在其编译器中执行此操作。Cuzco 仍然使用硬件来创建动态调度，但将该作业移至重命名/分配阶段，而不是后端的调度程序。在传统的乱序 CPU 中，调度程序可能是一种昂贵的结构，因为它们必须在每个周期检查指令是否已准备好执行。在 Cuzco 上，后端调度程序只需等待指定数量的周期，然后发出一条指令，因为知道依赖关系届时将已准备就绪。

## Time Resource Matrix (TRM)

Time	Scheduled		Read ports			Write ports		Br	ALU		LSU
83	O	P	A	A	J						
84	Q	R	L			J		A	J		
85	S	T	N			M		M			L
86	U	V	H	I	T	N			N		
87	W	X	K	K		H	I		H	I	T
88	Y		B	B		K			K		
89	a	b	C	C	D	D			B		
90	c	d	E	E	F	F	D		C	D	
91	e	f	G	G	P		F		E	F	
92	g	h	U						G		P
93	i	j	O	O	V	c					U
94	k	l	X			O	c		O	c	V

- Instructions are efficiently scheduled in the TRM, after all required inputs operands are predicted to become available by Scoreboard, taking advantage of all available resources
- In this example snippet from Dhrystone, instruction O is scheduled in cycle 83, issued in cycle 93, and executed in cycle 94

9

Condor Computing. All rights reserved. Copyright 2023

CONDOR

为了执行基于时间的调度，Cuzco 维护着一个时间资源矩阵 (TRM)，该矩阵跟踪未来一定数量周期内各种资源（如执行端口、功能单元和数据总线）的利用率。TRM 可以预测未来 256 个周期，从而控制存储需求。由于在硬件中搜索 256 行矩阵的成本极其高昂，因此 Cuzco 仅在预测指令的依赖项已准备就绪后，在小窗口中查找可用资源。Condor 发现搜索八个周期的窗口是一种很好的权衡。由于重命名器最多可以处理八条指令，因此它每个周期最多必须访问 TRM 中的 64 行。如果重命名器在搜索窗口中找不到可用资源，则指令将在 ID2 阶段停滞。

另一个潜在的限制是 TRM 的大小，这可能是长延迟指令的限制。然而，最长延迟的指令往往是缓存未命中的加载指令。Cuzco 始终假设 TRM 调度的 L1D 命中，并使用重放来处理 L1D 未命中。这意味着由于 TRM 大小限制导致 ID2 的停顿也应该很少见。

## Pros of Time-Based Microarchitecture



- Build execution schedule with known/projected dependencies and latencies:
  - Activate operands, resources, and execution units only at scheduled times
  - No search or prioritized selection at reservation stations
  - Reduced scheduler complexity (logic and area)
  - Dynamic power reduction vs conventional scheduler
- Two instructions scheduled/slice/cycle, scalable with slices
  - TRM2: Identify functional unit slots within 2 cycles of operand availability
  - TRM8: Within the next 8 cycles

12

© 2025 Condor Computing, Inc. – An Anlogic Company. All rights reserved.

CONDOR

与假设的“贪婪”设置相比，在这种设置下，核心能够在考虑执行资源限制的情况下创建完美的调度，而限制 TRM 搜索窗口会导致性能降低几个百分点。Condor 指出，创建一个能够达到“贪婪”程度的核心甚至可能根本不可能。传统的乱序核心不会受到 TRM 相关的限制，但可能会因为其他原因而难以创建最佳调度。例如，分布式调度器可能在一个调度队列中让多个微操作准备就绪，即使其他调度队列中可能有空闲的执行单元，也会面临“假”延迟。



## Dealing With Dynamic Behaviors

- TRM assumes optimal latencies and dependences, and adds dynamic runtime updates
  - Core: Variable exec latencies, misprediction initiated flushes
  - Memory: Load data L1 cache and TLB misses, bank conflicts
    - Dynamic memory response: L2/L3/Mem latency prediction
    - Load/Store RAW hazard prediction

Metric/Config	Default	Mem Dep Predictor	Reg based Predictor	No RAW hazards
Specint2k6/GHz (%diff)	100%	-1.0%	+1.4%	+4.2%
RAW Flush PKI	1.2	0.97	0.68	0

- Rescheduling of dependency chains
  - Instructions (L/S) will replay until successful completion (dynamic exec power)
  - Reschedule (ALU) can result in wasted issue slots (scheduling power)

Config	trm2	trm4	trm8	greedy
Replays in PKI	68.12	68.75	70.07	99.74
Replays (%diff)	-31.7%	-31.0%	-29.4%	0%

13

© 2023 Condor Computing, Inc. - An Andes Company. All rights reserved.

CONDOR

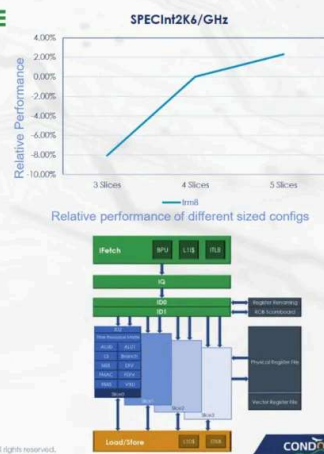
静态调度仅在指令延迟预先已知的情况下有效。某些指令具有可变延迟，例如可能错过缓存或 TLB、遇到存储体冲突或需要存储转发的加载指令。如前所述，Cuzco 使用指令重放来处理可变延迟指令及其相关的动态行为。重命名器确实采取了一些措施来减少重放，例如检查加载指令是否从与先前存储指令相同的寄存器获取地址。但是，它不会像英特尔酷睿 2 那样尝试预测内存依赖性，也不会尝试预测加载指令是否会错过缓存。

## 无序后端

Cuzco 中的乱序执行相对简单，因为重命名/分配阶段负责确定指令的执行时间。每条指令只需在调度程序中保留，直到经过指定数量的周期后，才会被发送执行。如果重命名/分配阶段猜测错误，则会通过“毒药”位进行重放。错误执行指令的结果数据实际上会被标记为“毒药”，任何使用该数据的指令都将被重新执行。重放指令会消耗电力并浪费执行吞吐量，因此理想情况下重放应该很少发生。每 1000 条指令重放 70.07 次似乎有点高，但这可能不是什么问题，因为执行资源很少会成为乱序核心的限制因素。考虑到大多数现代芯片很少以持续的方式使用其核心宽度，多占用大约 7% 的执行资源或许是一个可以接受的权衡。

## Slice-based Microarchitecture

- Baseline: 4 slices, 2 instruction pipelines/slice
- Uniformly scalable design IP
  - Each slice implements a fully compatible RISC-V CPU
  - Each slice adds symmetric set of resources to the machine
- Static PPA control through IP configuration
- Dynamic power vs. throughput control with slice-based enables and scheduling



14

© 2023 Condor Computing, Inc. - An Andes Company. All rights reserved.

CONDOR

执行资源被分组为多个切片，每个切片都有一对流水线。一个切片可以执行内核支持的所有 RISC-V 指令，因此通过更改切片数量可以轻松扩展执行资源。每个切片由一组执行队列 (XEQ) 组成，这些队列保存着等待功能单元的微操作。Cuzco 的每个功能单元都有 XEQ，这与传统设计不同，传统设计往往有一个调度队列，为连接到执行端口的所有功能单元提供数据。四个寄存器读取端口为切片提供操作数，两个写入端口处理结果写回。总线冲突也由 TRM 处理。一个切片每个周期不能执行超过两个微操作，即使这样做也不会使寄存器读取端口超额认购。例如，一个切片不能在同一个周期内发出整数加法、分支和加载，即使这只需要四个寄存器输入。

XEQ 的大小可根据工作负载特性进行调整，就像调整分布式调度器一样。虽然可以根据客户需求设置 XEQ 的大小，但 Condor 提供了一些基准配置的数据。ALU 有 16 个入口队列，而分支和地址生成单元 (LS) 有 8 个入口队列。XEQ 的大小可以按 2 的幂次方进行调整，从 2 个到 32 个入口。片间转发通常只有一个周期的延迟。核心可以配置为实现零周期跨片转发，但这将非常困难。

在矢量方面，Cuzco 通过多个微操作支持 256/512 位 VLEN，这些微操作分布在各个执行片上。执行单元原生宽度为 64 位。每个片上有一个 FMA 单元，因此 FP32 峰值吞吐量为每周期 8 次 FMA 操作，如果将加法和乘法单独计算，则为 16 FLOPS。FP 加法的执行延迟为 2 个周期，而 FP 乘法和乘加法的延迟为 4 个周期。两周期 FP 加法延迟令人欣喜，与 Neoverse N1 和英特尔 Golden Cove 等最新核心相当，尽管时钟频率要低得多。

加载/存储

Cuzco 的加载/存储单元包含一个 64 项加载队列、一个 64 项存储队列和一个 64 项数据缓存未命中队列。访问数据缓存后，加载操作可能会离开加载队列，这可能会导致类似于 AMD Zen 系列处理器的行为。在 Zen 系列处理器中，乱序后端的待处理加载操作数量可能远超官方公布的加载队列容量。该核心采用四片式配置，拥有四条加载/存储流水线，或者说每片式配置一条流水线。最大加载带宽为 64B/周期，可通过矢量加载实现。

	Configuration	Replacement	Latency	Bandwidth
L1D	64 KB 8-way	pLRU	4 cycle	64B/cycle
L2	Up to 8 MB, 16-way	Pseudo-Random	18 cycle (+14 vs L1D)	64B/cycle
L3	Up to 256 MB Up to 16-way Cluster Shared	Pseudo-Random	38 cycle (+20 vs L2)	64B/cycle per core, per slice

L1D 采用物理索引和物理寻址 (PIPT) 机制，因此地址转换必须在 L1D 访问之前完成。为了加快地址转换速度，Cuzco 配备了一个包含 64 个条目的全相联数据 TLB。L2 TLB 采用 4 路组相联机制，可以包含 1K、2K 或 4K 个条目。Cuzco 的核心私有统一 L2 缓存也具有可配置的容量。例如，在台积电 5nm 工艺上，2MB L2 占用 1.04 平方毫米的面积。

### Cuzco Memory System Microarchitecture

#### Private L1 Caches

- 64KB I\$/D\$, 64B line size, 8-way, pLRU
- PIPT (Physical Index and Physical Tag)
- 64-byte cache line size
- I\$/D\$ prefetch and D\$ writearound
- SECCDED ECC error protection
- Up to 64 pending miss requests
- 4 cycle load->use penalty

#### Private L2 Cache

- Up to 8MB, 64B line size, up to 16-way, pseudo-random replacement
- I/D prefetch
  - Preset and configurable prefetch policies
- Configurable multi-cycle SRAM accesses
- SECCDED ECC error protection
- +14 cycle delay on L2 hit

#### Privilege and Memory Management

- Machine (M), hypervisor (H), supervisor (S) and user (U) privilege modes
- Memory management unit (MMU)
  - Bare, Sv39, Sv48, and Sv57 VA translations
  - Synaprot, Svpbmt, Svinval VM extensions
  - L1 I/D TLBs: 64-entry, fully associative
  - L2 TLB: up to 4K-entry, 4-way
  - PMP and ePMP support with 16 PMP entries
- 16 PMA regions

Diagram illustrating the Cuzco Memory System Microarchitecture. It shows four cores (Core 0, Core 1, Core 2, Core 3) arranged in a 2x2 grid. Each core has its own L1 and L2 cache. The L2 caches are connected to a shared L3 cache. The diagram is labeled 'Processor' and 'L3'.

15

© 2020 Conda Computing, Inc. - An Andes Company. All rights reserved.

每个集群的八个核心共享一个 L3 缓存，该缓存被拆分成多个切片以处理来自多个核心的带宽需求。每个切片可提供 64B/周期的带宽，切片数量与核心数量匹配。因此，Cuzco 在整个缓存层级结构中享有 64B/周期的负载带宽，当然，如果来自不同核心的访问与同一切片发生冲突，则 L3 带宽可能会降低。集群内的核心和 L3 切片通过交叉开关连接。L3 缓存的运行速度最高可达核心时钟频率。系统请求通过 64B/周期的 CHI 接口发出。集群之外的系统拓扑由实施者决定。

## Cuzco: Cluster Microarchitecture

### Shared L3 Cache

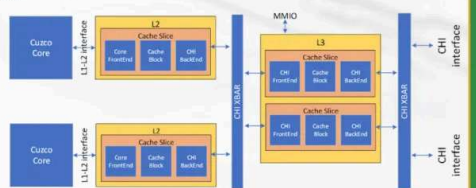
- Up to 256MB, 64B line size, up to 16-way, pseudo-random replacement
- I & D prefetch, configurable policies
- Max outstanding reads/writes (cacheable and uncacheable): 32-128
- Max 64 outstanding snoop transactions
- Configurable multi-cycle SRAM accesses
- SECDED ECC error protection

### Cluster with Multicore Cache Coherency

- Up to 8 cores+L2s in a cluster
- Coherence Manager and L3s

### Bus Interfaces

- 512-bit main memory CHI bus interface
- 256/512-bit memory mapped I/O (MMIO) interface
- Core+L2 vs. external-bus clock
- Asynchronous, and Synchronous N:1 clock ratios



16

© 2022 Condor Computing, Inc. — An Andes Company. All rights reserved.

CONDOR

缓存未命中的重放是通过将数据消费者重新调度到预计数据就绪的稍后时间来实现的。因此，L3命中会导致消费指令执行三次：一次是针对预测的L1D命中，一次是针对预测的L2命中，最后一次是针对具有正确数据的L3命中。

## 最后的话

高性能CPU设计在过去几十年中逐渐稳定下来，并最终集中于一种乱序执行模型。无可否认，乱序执行非常困难。多年来，人们尝试过许多替代方案，但都未能持久。英特尔的安腾处理器试图使用基于ISA的方法，但未能取代该公司自家使用乱序执行的x86内核。Nvidia的丹佛处理器尝试将ARM指令动态编译成微码包，但这种方法未能延续下去。如今所有成功的高性能设计通常都使用相同的乱序执行策略，尽管存在很多变化。这是由ISA兼容性的要求以及在广泛应用程序中提供高单线程性能的需求所驱动的。打破常规显然充满危险。

Condor力求打破常规，但其突破性之处在于其深入内核，从功能角度来看，它对软件来说是不可见的，从性能角度来看，它几乎不可见。Condor内核运行RISC-V指令，因此与Itanium不同，它受益于该软件生态系统。它不像Denver那样依赖于编译后的微码缓存，因此在处理代码局部性较差的问题时，其性能不会像典型的OoO内核那样下降。最后，指令重放能够有效地创建动态调度并处理缓存未命中。

### 参考链接

<https://chipsandcheese.com/p/condors-cuzco-risc-v-core-at-hot>

END

嵌入式系统专家之声推荐搜索

RISC-V | 嵌入式系统 | 物联网

【点击上方👆搜索词条可查看号内更多其他内容】



微信搜一搜

嵌入式系统专家之声

关注我们，了解更多精彩内容

转发，点赞，在看，安排一下？

