SYNOPSYS®

WHITE PAPER

利用ARC nSIM NCAM实现快速周期近似模拟

关键事实—方法论—用例

作者

Igor Böhm
新思科技软件架构师与技术主管

Alexander Chuykov
新思科技应用工程师

成功开发软件（如固件/应用）的关键因素之一是在目标硬件尚未完善的情况下快速运行和分析软件的能力。在设计过程的早期阶段比如硅前阶段，越早开展这些活动，对软件开发越有利。

通常，硅前阶段主要包括三项活动，每项活动都面临不同的挑战：

- 探索硬件/软件配置空间
  - 挑战：从较大的设计空间中寻找近似最优的配置集
- 软件栈的功能启动
  - 挑战：找到并消除功能性软件问题
- 软件栈的硅前阶段优化
  - 挑战：找到并优化软件栈中的关键热点

DesignWare® ARC® nSIM指令集模拟器旨在通过提供两种模拟模式而应对这些挑战：快速功能模拟模式和近周期精确模拟模式。快速功能模拟模式可以应对硅前阶段软件功能启动的需求。近周期精确模拟模式旨在帮助解决探索硬件/软件配置空间的挑战，并实现硅前优化。

最终目标是通过提供适当的工具，使客户能够在硅片可用之前即获得可产品化的软件代码，从而进一步缩短上市时间，并降低由于后期设计变更而导致的成本高昂的返工风险。

ARC nSIM NCAM—硬件性能建模

尽管nSIM本身是早期软件开发和启动的关键，但是，由于它是一个纯粹的指令精确的行为级模拟器，因此无法预测微架构性能。然而，预测客户软件栈的目标性能非常重要 - 它会影响如CCM大小、缓存关联性和分支预测器配置等硬件配置决策。

为了提取此类信息，客户通常会使用周期精确模型，或者使用FPGA原型验证平台。

尽管这些方法已被业界接受并广泛使用，但也存在一些风险和挑战：

- 鸡生蛋还是蛋生鸡”的问题：
  - 最终硬件配置选项取决于在目标硬件上执行的软件栈
  - 然而，在最终确定硬件配置选项时，客户通常没有RTL，更不用说能够评估和探索配置选项的稳定软件栈了。使用在特定领域的专业知识和推断方法来确定硬件配置的情况很常见，但是存在一定风险。
- 模拟时间过长：
  - RTL模拟和从RTL导出周期精确模型的仿真速度非常慢，因此，这仅适用于小规模的内核测试。然而，最终的软件栈要比那些内核复杂得多。这增加了此类基于不符合现实条件做出的决策的风险，从而导致无法做出最优硬件配置选择，而且随后由于进度延迟会进一步增加开发成本。
  - 使用RTL模拟或周期精确模型尝试不同硬件配置选项的周转时间太长。此外，用户还必须在修改配置选项时重建RTL仿真模型，这使得挑战进一步加剧。

nSIM近周期精确模拟模式 (NCAM) 旨在弥合以下两方面的差距：

- 快速指令精确模拟 - 提供极为有限的微架构性能细节
- 周期精确模拟方法 – 速度太慢（模拟速度）、缺乏灵活性（重新配置成本高），或者无法运行整个软件栈（资源受限）

NCAM是什么？

除功能（指令集架构层次）模拟之外，ARC nSIM还提供NCAM，这是一种近似微架构模拟模式，能够预估目标处理器的性能（例如执行周期、每个指令的时钟数 (CPI)、分支预测失误率等）。这种模拟模式使客户能够在签核硬件配置之前，估计并优化在ARC处理器硬件上运行的最终软件栈的性能。这些活动发生在硅片可用之前，而且不需要FPGA原型设计和模拟。

- NCAM并非独立的模拟器：
  - NCAM是ARC nSIM的一种硬件性能模拟模式
  - NCAM并非源自RTL
- NCAM来自于微架构规范，因此它是周期近似模式
- NCAM预测硬件/软件性能
  - NCAM是显示了关键性能指标 (KPI)的硬件计时模型
  - 软件执行周期数
  - 缓存未命中次数
  - 分支预测错误数
- NCAM速度快
  - 使用NCAM运行CoreMark（周期近似模式）耗时3秒
  - 使用xCAM运行CoreMark（周期精确）耗时5个小时

Benchmark	ARC nSIM functional simulation	ARC nSIM NCAM cycle approximate simulation	ARC nSIM NCAM cycle accurate simulation
CoreMark	12 MIPS	3.5 MIPS	0.005 MIPS

图1：NCAM和xCAM之间的CoreMark MIPS对比

NCAM的精确度如何？

NCAM源于微架构规范，这一规范在抽象层面描述了指令时序行为。与周期精确模拟器相比，在不同抽象层实现微架构时序模型是高速模拟的关键。

由于NCAM不是源自RTL，因此，它不具有周期精确性。它是周期近似的，即NCAM预测的性能可能与源自RTL的周期精确模拟器报告的性能不完全一致。换句话说，在周期精确模拟器上执行的程序在NCAM上可能产生不同的周期数。

那么，为什么说NCAM是周期近似模拟呢?抽象指令时序模型不太可能捕捉到底层微架构表现出的所有极端行为。当在次优条件下操作硬件（例如分支预测失误率高）时，我们通常会遇到极端情况。在这种情况下，NCAM预测的性能与周期精确模拟器相比可能会出现更大的偏差。

应用代码的多样性使得NCAM预测模拟的精确性范围缺乏普适性，但我们已经采取了保障措施和方法，以帮助识别这些情况，从而使结果具有一定的置信度。

为何要使用周期近似ARC nSIM NCAM技术?

- 高速模拟
  - ARC nSIM NCAM的速度比周期精确模型快几个数量级（几分钟与几天的差异）。它旨在模拟整个软件栈和真实工作负载，而不仅仅是输入值有限的小函数或内核
  - 高速模拟可直接加快反馈速度。这又缩短了设计和开发时间
  - 使用真实工作负载对整个软件栈进行模拟大大降低了根据小内核的推断做出次优硬件配置决策的风险
- 灵活集成
  - 由于NCAM是ARC nSIM的模拟模式，它可以集成到ARC nSIM支持的所有应用和用例中，例如SystemC（OSCI/Acellera和Synopsys Virtualizer）、Platform Architect和MetaWare Debugger等
  - 与MetaWare Debugger的集成可以通过MetaWare Debugger 脚本语言提供高级自动化功能，从而能够自动提取任何感兴趣的代码区域的KPI
- 高级分析
  - NCAM提供丰富的分析能力，旨在帮您快速找到需要的信息
  - 与MetaWare Debugger结合后，可以轻松地深入洞察并将重要KPI追踪至函数、语句和单个指令。例如，只需点击几下鼠标，就可以找到指令缓存未命中数量最多的函数，并深入挖掘未命中的基础指令块。

处理不确定性和近似性

处理近似性和不确定性的关键因素是允许我们根据处理器微架构基本知识来判断预测结果可信度的方法论。

NCAM方法论的基本指导原则是：

- 不要追踪并依靠单个关键性能指标来检测是否存在性能问题
- 相反，要追踪并使用一组关键性能指标来发现性能问题

周期近似NCAM模拟方法论

为了证明遵循这些指导原则的必要性，我们来看这样一个例子：不使用推荐的NCAM方法，即考虑多个KPI，而仅依赖一个KPI。在本例中，我们试图确定，对某个硬实时系统，在一组指定的硬件配置上，当模拟多种工作负载时，其关键函数能否满足执行时间要求。

此信息是确定哪种硬件配置最有可能适用于指定软件栈的关键。

仅考虑单个KPI可能会让您产生一个错误的印象，即认为时间要求总是能够被满足。这样，您可能会错误地认为没有必要进一步优化软件栈。在上面的举例中，单个关键性能指标的置信度很高，但没有任何额外的证据支持这种高置信度。一旦硅片可用，一个现实的风险是有时关键时间要求可能无法被满足，从而您必须推迟计划，在流程的后期尝试优化软件栈。

这个问题的解决方案是遵循我们推荐的NCAM方法论，即考虑多个KPI（例如周期数、分支预测错误、高速缓存未命中等），通过额外的证据而获得对预测性能的置信度。

如何通过评估多个KPI提高结果的置信度?

通常，不太可能的一种情况是，所有KPI都在目标范围内，但是程序性能不太理想。换句话说，如果一个KPI看起来可疑，则其他KPI的准确性可能会降低。因此，考虑更多KPI能够为您提供重要的额外警示。

例如，考虑至少三个KPI：(1)程序执行周期数，(2)发生高速缓存未命中的次数，以及 (3) 分支预测错误的数量。如果周期计数在范围内，指定应用程序的缓存未命中数也是合理的，但分支预测失误率非常高，您必须调查并找出原因，并尝试在这方面优化代码。

在本例中，这可能是由于缺乏内联以及存在许多小的方法调用（例如在C++中）而导致的：这些调用在头文件中声明，但在各自的实现（即.cc）文件中定义（即实现）。许多良好的软件工程指南都是这样规定的。借助NCAM，这种情况很容易发现。一旦确定了性能问题的原因，就可以使用现成的措施纠正问题，包括使用特殊的编译器优化（如链接时间优化），以及将关键的小方法实现转移到程序头中，让编译器做出关键的内联决策。

发现性能问题

成功的关键在于能够快速找到代码中存在潜在性能问题的区域。为此您必须借助正确的工具，快速的“大海捞针”。下图显示了如何使用MetaWare调试器 (MDB) 和nSIM在GUI（如下图）或命令行/批处理模式下调试ARC应用程序。在左下角，Profiling 窗格显示了每个函数的NCAM计数器值。（它还可以构建运行时调用树，当面对较深的调用栈，并且需要找出哪些子函数是瓶颈时，这一点特别有用）。下拉菜单显示出一些可以启用的计数器。

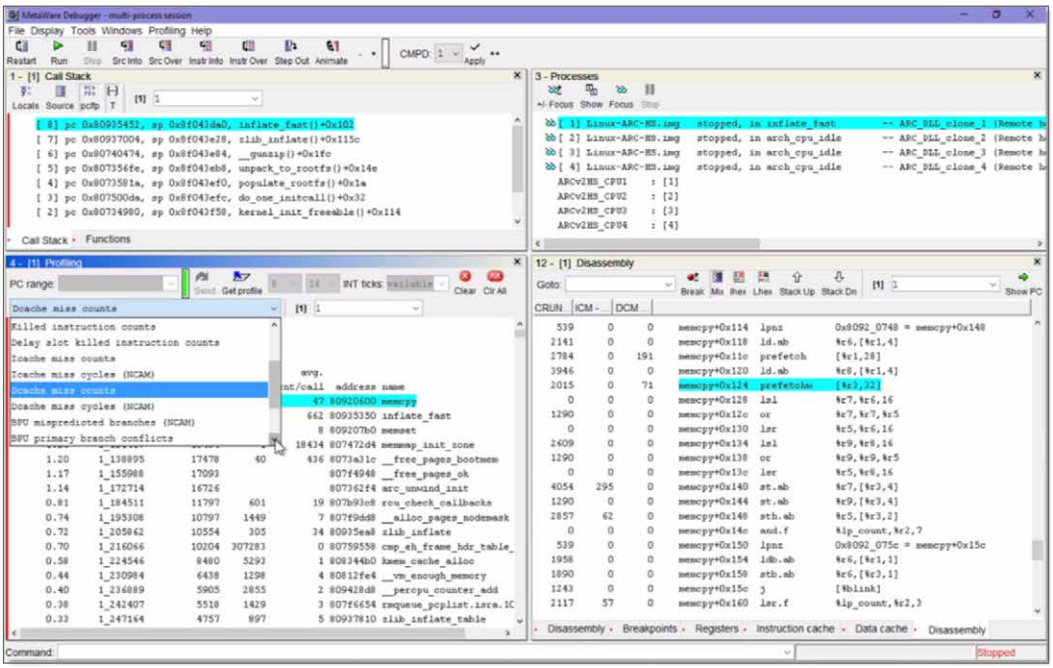

图2：MDB和NCAM的应用

另一个特性是Source窗格和Disassembly窗格。Disassembly窗格位于右下角。除了实际的反汇编之外，它还可以显示归因于单个指令的计数器值。左侧的CRUN、ICM和DCM列分别表示周期计数、指令和数据缓存未命中。

我们来看另一个KPI，这是预测错误的分支，可以从Profiling和Disassembly窗格中的下拉列表中找到。

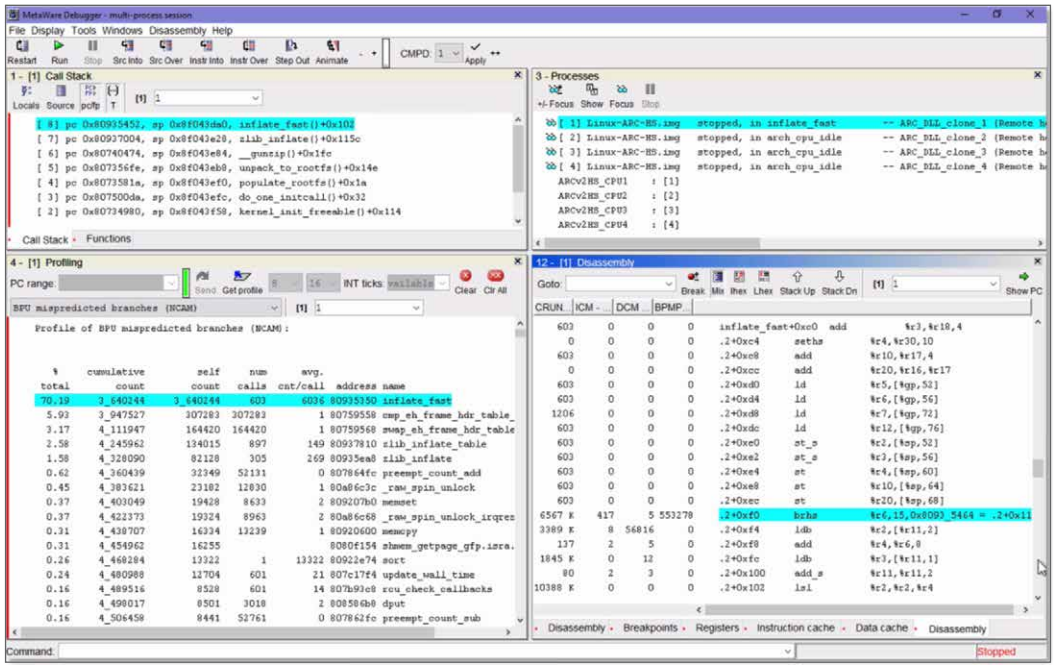

图3：使用MDB和NCAM查看分支预测错误数

这需要执行两个步骤的操作：首先在Profiling窗格中查看贡献最大的函数。然后使用函数名称符号和Disassembly窗格进一步缩小范围。大家可以看到，在一个分支指令周围明显存在一个瓶颈，它显示出极高的BPMP（即分支预测错误）数。因此，您只需点击三下即可找到有问题的代码序列。下一步是找出问题发生的原因，并调整C/C++或汇编代码，以消除该特定指令上的停顿（可能是查看if/else或for/while语句，应用一些编译器控制选项，重新排列代码等）。对其他经常执行的函数，重复这种做法可以确保性能良好，并且在测试硅上相同的代码时不会出现意外。

ARC nSIM NCAM—用例

下面，我们通过新思科技经常面对的两个真实示例来概括和总结NCAM用例。

硬件配置设计空间探索

借助ARC nSIM NCAM，您可以快速查看哪些硬件配置选项会影响目标CPI（每条指令的周期数），并以电子表格或其他方便的形式进行汇总，用于估计硬件配置变更对您感兴趣的代码的影响。这是关键的早期输入数据，有助于推动决策，以实现指定的硬件配置。

- 例：探索如何对指令缓存 (I$)、数据缓存(D$) 和二级缓存 (L2$) 进行尺寸标注/配置
  - 容量不命中：
    - 增加缓存大小
  - 冲突不命中：
    - 增加缓存大小
    - 增加关联性（更多路）
    - 减少缓存行大小（更多组）

			Benchmark	Benchmark 1		
			Function	Critical Function 1	Critical Function 2	Critical Function 3
#	Item	Configuration Experiment	Input	% change in CPI	% change in CPI	% change in CPI
1	ICCM	Increase ICCM size	Medium	-1%	0%	-7%
2	DCCM	Increase DCCM size	Medium	-5%	0%	-3%
4	I$	Increase cache size	Medium	-20%	-11%	-4%
5	I$	Increase associativity	Medium	-12%	-5%	-7%
6	I$	Decrease line size	Medium	17%	50%	32%
7	D$	Increase cache size	Medium	-4%	-2%	0%
8	D$	Increase associativity	Medium	-1%	0%	0%
9	D$	Decrease line size	Medium	5%	1%	1%
10	BPU	Increase branch cache size	Medium	-5%	-7%	-9%
11	BPU	Increase size of Return Address Stack	Medium	0%	0%	0%
12	BPU	Increase size of Pattern Table	Medium	0%	-1%	0%
13	CSM	Increase CSM size	Medium	1%	0%	0%
14	L2$	Increase cache size	Medium	0%	-1%	0%
15	L2$	Increase associativity	Medium	0%	0%	0%

4：使用NCAM追踪硬件配置选项变更

- 例：探索如何对BPU进行尺寸标注/配置
  - 容量不命中
    - 增加分支缓存大小
  - 更大的返回地址堆栈 (RAS) 意味着可以正确预测更深的子例程嵌套级别
  - 更大的Pattern表意味着可以同时追踪更多全局分支历史，以获取更多分支或跳转指令
  - 例：探索实时系统中的关键句柄程序是否需要
  - CCMs
    - 可预测的性能—费用高
  - 缓存架构
    - 不太可预测的性能—费用低

编译器优化空间探索

除了显而易见的编译器优化选项之外，例如-O1、-O2和-O3，您还可以通过许多更专业的选项来专门调整：

- 代码密度
- 性能（内联、展开）

这些选项的交互对于大型复杂软件栈非常重要，特别是在一些函数具有实时约束（即必须在时间预算内完成）而有些函数则可以随时中断的硬实时系统中。

通过使用ARC nSIM NCAM，您可以使用完整的应用系统地探索编译器优化空间，以深入了解优化选项的最佳组合，以便获得如下表所示的信息（表中显示了编译和链接时优化的影响）。

Impact of Compile Time Optimizations

Optimization Flags	% MAX	% AVG
Baseline	0.00%	0.00%
O2O1	-6.85%	-9.85%
O2	-6.16%	-10.00%
O3	-7.52%	-9.24%
O2_dense	-8.86%	-9.80%
O2_dense_Sinl0	-8.95%	-9.73%
O2_dense_Sinl15	-5.27%	-8.40%
O2_dense_Sinl30unr60_unr60mi	-10.11%	-11.03%
O2_dense_Sinl30unr60_unr0mi	-9.21%	**-11.72%**
O2_dense_Sinl30unr60	-10.35%	-9.90%
O2_dense_Sinl30unr0	-12.24%	-9.61%
O2_dense_Sinl30	**-12.65%**	-9.69%
O2_dense_Sinl60	-2.78%	-8.47%
O2_dense_Sunr0	-10.74%	-9.67%
O2_dense_Sunr15	-10.41%	-10.07%
O2_dense_Sunr60	-12.59%	-9.59%
O2_dense_Sunr100	-12.59%	-9.59%

Impact of Link Time Optimizations

Optimization Flags	% MAX	% AVG
Baseline	0.00%	0.00%
lto_intrn_O2Os	-11.37%	-11.25%
lto_intrn_O2O1	-12.61%	-11.42%
lto_intrn_O2_Sinl30	-14.31%	-11.42%
lto_intrn_O2	-7.51%	-11.35%
lto_intrn_O2O1_mi	-4.49%	-5.00%
lto_intrn_O2O1_inl60	-12.61%	-11.42%
lto_intrn_O2O1_unr100	-13.44%	-10.70%
lto_intrn_O2O1_unr60	-12.18%	-10.83%
lto_intrn_O2O1_unr0	-13.19%	-9.59%
slto_sect_O2_Sinl30lto	-17.66%	-14.41%
slto_sect_O2Os1_unr0mi_Smilto	-18.34%	-14.05%
slto_sect_O2Os1_Smilto	**-18.57%**	**-15.54%**
slto_sect_O2Os1	-15.04%	-12.47%
slto_sect_O2_Sinl30_inl100	-17.81%	-14.44%
slto_secO3O2_Sinl30unr60_unr0mi	-18.16%	-13.52%
lto_intrn_O2O1_inl15	-2.66%	-2.90%

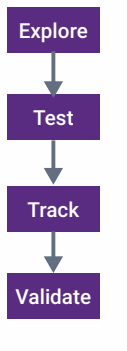

图5：使用NCAM追踪编译器选项变更（表视图）

这对于追踪和以可视化形式呈现关键函数性能随时间变化的实验尤其有用，可以查看在所有应用阶段（即冷启动、预热、工作负载峰值）是否仍能满足性能要求。

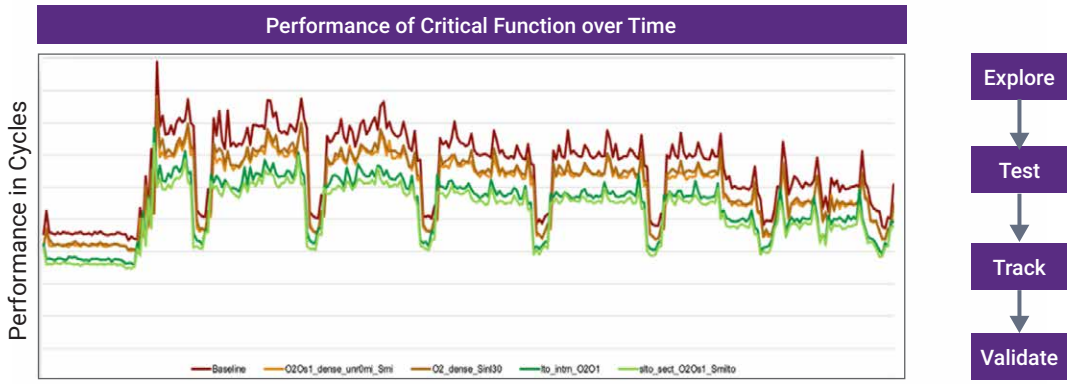

图6：使用NCAM追踪编译器选项变更（图形视图）

在上图中，可以看到-O3优化并没有提供最佳优化而提高性能。显然，编译器优化器以尽可能快的速度为-O3生成代码。通常，性能和代码大小之间需要折衷考虑— 激进的循环展开、函数内联等。如果在纯nSIM（指令精确模型）上运行此应用，我们可以为-O3打出最佳分数 — 更少的子例程调用开销。但是，NCAM考虑更多方面，包括BPU和缓存。因此，激进的代码展开不适用于这个函数。该函数具有较大的代码大小，指令缓存未命中的代价可能比子例程调用或保持循环要高。总之，对于复杂的软件优化方法，它可能并不明显，而且需要多轮才能找到最佳选项。此外，同一应用中的不同函数可能对优化有相反的要求。

您可以看到，微调编译器选项需要大量的工作和测试用例评估，但NCAM可能会发现一些不明显的情况，例如在增加指令数确实能够提高性能时。然后可以选择最佳选项，并在较慢的周期精确模型上运行这些选项，以验证并认可最佳选择。

确定最佳编译器选项集的一般流程

- 选择优化选项的基准
  - 限制应用程序性能预算和代码大小
  - 为应用程序选择KPI
- 运行分析
  - 获得KPI的分数
  - 查找瓶颈函数
- 调整优化选项。优化方法取决于KPI和应用目标。例如：
  - 大量指令缓存未命中—优化缓存设置（硬件优化）；降低展开/内联级别或重新定位/打乱函数（软件优化）
  - 大量数据缓存未命中—优化缓存未命中（硬件优化）；重新定位或打乱数据对象，使用预取（软件优化）
- 重复这些步骤
  - 如果您的应用具有不同的操作模式，请切换到其他模式，并重复所有步骤

该流程可让您找到一组接近最佳的硬件配置（如果适用）或工具链和软件优化选项。该流程能够与NCAM一起模拟全功能应用或大型函数集合。对于较小的单个函数和内核（例如微基准测试），最好使用周期精确的ARC xCAM或其他周期精确的模拟策略（例如FPGA）。

ARC nSIM NCAM—结语

实践证明，周期近似NCAM模拟的功能非常强大，已成为硬件/软件协同设计和软件开发过程中非常有价值的一部分。尽管它如此有效，但了解其局限性也很重要，因为在某些情况下，如果要求获得的结果的置信度很高（例如，几乎等同于周期精确），则不建议使用NCAM。以下简单规则有助于了解NCAM是否适用：

何时使用NCAM

- 探索硬件配置空间
  - 快速找到适用于您的应用的良好硬件配置集
  - NCAM可轻松配置，以供快速进行试验
- 应用程序优化（硅前和硅后）
  - 快速找到运行时主导函数
  - 关注带来优化的方面
- 编译器优化
  - 系统地探索大型编译器优化空间
  - 找到良好的优化选项

何时不使用NCAM

- 分析小型的微基准
  - 微基准通常探索硬件的极端情况，需要的精度较高
  - NCAM不是为该用例而专门设计的 —应该使用ARC xCAM、RTL模拟或FPGA
- 性能签核
  - 如果未在周期精确模式下执行性能签核，不得进行流片
  - NCAM是近似模拟，签核要求确定性和准确性—应该使用ARC xCAM、RTL模拟或FPGA

01/12/22.CS814264527-NCAM whitepaper Simplified Chinese translation.

Pub: May 2021