SYNOPSYS®

WHITE PAPER

利用ARC nSIM NCAM實現快速週期近似模擬

關鍵事實—方法論—用例

作者

Igor Böhm
新思科技軟體架構師與技術主管

Alexander Chuykov
新思科技應用工程師

成功開發軟體（如韌體/應用）的關鍵因素之一是在目標硬體尚未完善的情況下快速運行和分析軟體的能力。在設計過程的早期階段比如晶片生產前階段，越早開展這些活動，對軟體開發越有利。

通常，晶片生產前階段主要包括三項活動，每項活動都面臨不同的挑戰：

- 探索硬體/軟體設定空間
  - 挑戰：從較大的設計空間中尋找近似最優的配置集
- 軟體棧的功能啟動
  - 挑戰：找到並消除功能性軟體問題
- 軟體棧的晶片生產前階段最佳化
  - 挑戰：找到並最佳化軟體棧中的關鍵熱點

DesignWare® ARC® nSIM指令集模擬器旨在透過提供兩種模擬模式而應對這些挑戰：快速功能模擬模式和近週期精確模擬模式。快速功能模擬模式可以應對晶片生產前階段軟體功能啟動的需求。近週期精確模擬模式旨在幫助解決探索硬體/軟體配置空間的挑戰，並實現晶片生產前最佳化。

最終目標是透過提供適當的工具，使客戶能夠在晶片可用之前即獲得可產品化的軟體代碼，從而進一步縮短上市時間，並降低由於後期設計變更而導致的成本高昂的返工風險。

ARC nSIM NCAM—硬體效能建模

儘管nSIM本身是早期軟體開發和啟動的關鍵，但是，由於它是一個純粹的指令精確的行為級模擬器，因此無法預測微架構效能。然而，預測客戶軟體棧的目標效能非常重要—它會影響如CCM大小、快取關聯性和分支預測器配置等硬體配置決策。

為了提取此類資訊，客戶通常會使用週期精確模型，或者使用FPGA原型驗證平臺。

儘管這些方法已被業界接受並廣泛使用，但也存在一些風險和挑戰：

- 雞生蛋還是蛋生雞」的問題：
  - 最終硬體配置選項取決於在目標硬體上執行的軟體棧
  - 而，在最終確定硬體配置選項時，客戶通常沒有RTL，更不用說能夠評估和探索配置選項的穩定軟體棧了。使用在特定領域的專業知識和推斷方法來確定硬體配置的情況很常見，但是存在一定風險。
- 模擬時間過長：
  - RTL模擬和從RTL導出週期精確模型的模擬速度非常慢，因此，這僅適用於小規模的內核測試。然而，最終的軟體棧要比那些內核複雜得多。這增加了此類基於不符合現實條件做出的決策的風險，從而導致無法做出最優硬體配置選擇，而且隨後由於進度延遲會進一步增加開發成本。
  - 使用RTL模擬或週期精確模型嘗試不同硬體配置選項的周轉時間太長。此外，使用者還必須在修改配置選項時重建RTL模擬模型，這使得挑戰進一步加劇。

nSIM近週期精確模擬模式（NCAM）旨在彌合以下兩方面的差距：

- 快速指令精確模擬 - 提供極為有限的微架構效能細節
- 週期精確模擬方法—速度太慢（模擬速度）、缺乏靈活性（重新配置成本高），或者無法運行整個軟體棧（資源受限）

NCAM是什麼?

除功能（指令集架構層次）模擬之外，ARC nSIM還提供NCAM，這是一種近似微架構模擬模式，能夠預估目標處理器的效能（例如執行週期、每個指令的時鐘數（CPI）、分支預測失誤率等）。這種模擬模式使客戶能夠在簽核硬體配置之前，估計並最佳化在ARC處理器硬體上運行的最終軟體棧的效能。這些活動發生在晶片可用之前，而且不需要FPGA原型設計和模擬。

- NCAM並非獨立的模擬器：
  - NCAM是ARC nSIM的一種硬體效能模擬模式
  - NCAM並非源自RTL
- NCAM來自於微架構規範，因此它是週期近似模式
- NCAM預測硬體/軟體效能
  - NCAM是顯示關鍵效能指標（KPI）的硬體計時模型
  - 軟體執行週期數
  - 快取未命中次數
  - 分支預測錯誤數
- NCAM速度快
  - 使用NCAM執行CoreMark（週期近似模式）耗時3秒
  - 使用xCAM執行CoreMark（週期精確）耗時5個小時

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

圖1：NCAM和xCAM之間的CoreMark MIPS對比

NCAM的精確度如何?

NCAM源於微架構規範，這一規範在抽象層面描述了指令時序行為。與週期精確模擬器相比，在不同抽象層實現微架構時序模型是高速模擬的關鍵。

由於NCAM不是源自RTL，因此，它不具有週期精確性。它是週期近似的，即NCAM預測的效能可能與源自RTL的週期精確模擬器報告的效能不完全一致。換句話說，在週期精確模擬器上執行的程式在NCAM上可能產生不同的週期數。

那麼，為什麼說NCAM是週期近似模擬呢?抽象指令時序模型不太可能捕捉到底層微架構表現出的所有極端行為。當在次優條件下操作硬體（例如分支預測失誤率高）時，我們通常會遇到極端情況。在這種情況下，NCAM預測的效能與週期精確模擬器相比可能會出現更大的偏差。

應用代碼的多樣性使得NCAM預測模擬的精確性範圍缺乏普適性，但我們已經採取了保障措施和方法，以幫助識別這些情況，從而使結果具有一定的置信度。

為何要使用週期近似ARC nSIM NCAM技術?

- 高速模擬
  - ARC nSIM NCAM的速度比週期精確模型快幾個數量級（幾分鐘與幾天的差異）。它旨在模擬整個軟體棧和真實工作負載，而不僅僅是輸入值有限的小函數或內核
  - 高速模擬可直接加快反饋速度。這又縮短了設計和開發時間
  - 使用真實工作負載對整個軟體棧進行模擬大大降低了根據小內核的推斷做出次優硬體配置決策的風險
- 靈活整合
  - 由於NCAM是ARC nSIM的模擬模式，它可以整合到ARC nSIM支援的所有應用和用例中，例如SystemC（OSCI/Acellera和Synopsys Virtualizer）、Platform Architect和MetaWare Debugger等
  - 與MetaWare Debugger的整合可以透過MetaWare Debugger 腳本語言提供高級自動化功能，從而能夠自動提取任何感興趣的代碼區域的KPI
- 高級分析
  - NCAM提供豐富的分析能力，旨在幫您快速找到需要的資訊
  - 與MetaWare Debugger結合後，可以輕鬆地深入洞察並將重要KPI追蹤至函數、語句和單個指令。例如，只需點擊幾下滑鼠，就可以找到指令快取未命中數量最多的函數，並深入挖掘未命中的基礎指令塊。

處理不確定性和近似性

處理近似性和不確定性的關鍵因素是允許我們根據處理器微架構基本知識來判斷預測結果可信度的方法論。

NCAM方法論的基本指導原則是：

- 不要追蹤並依靠單個關鍵效能指標來檢測是否存在效能問題
- 相反，要追蹤並使用一組關鍵效能指標來發現效能問題

週期近似NCAM模擬方法論

為了證明遵循這些指導原則的必要性，我們來看這樣一個例子：不使用推薦的NCAM方法，即考慮多個KPI，而僅依賴一個KPI。在本例中，我們試圖確定，對某個硬實時系統，在一組指定的硬體配置上，當模擬多種工作負載時，其關鍵函數能否滿足執行時間要求。此資訊是確定哪種硬體配置最有可能適用於指定軟體棧的關鍵。

僅考慮單個KPI可能會讓您產生一個錯誤的印象，即認為時間要求總是能夠被滿足。這樣，您可能會錯誤地認為沒有必要進一步最佳化軟體棧。在上面的舉例中，賦予單個關鍵效能指標太高置信度，但沒有任何額外的證據支援這種高置信度。一旦晶片可用，一個現實的風險是晶片上測試表明有時關鍵時間要求可能無法被滿足，從而您必須推遲計劃，在流程的後期嘗試最佳化軟體棧。

這個問題的解決方案是遵循我們推薦的NCAM方法論，即考慮多個KPI（例如週期數、分支預測錯誤、高速快取未命中等），透過額外的證據支持而獲得對預測效能的置信度。

如何透過評估多個KPI提高結果的置信度?

通常，不太可能的一種情況是，所有KPI都在目標範圍內，但是程式效能不太理想。換句話說，如果一個KPI看起來可疑，則其他KPI的準確性可能會降低。因此，考慮更多KPI能夠為您提供重要的額外警示。

例如，考慮至少三個KPI：（1）程式執行週期數，（2）發生高速快取未命中的次數，以及（3）分支預測錯誤的數量。如果週期計數在範圍內，指定應用程式的快取未命中數也是合理的，但分支預測失誤率非常高，您必須調查並找出原因，並嘗試在這方面最佳化代碼。

在本例中，這可能是由於缺乏內聯以及存在許多小的函數調用（例如在C++中）而導致的：這些調用在標頭檔案中宣告，但在各自的實現（即.cc）檔案中定義（即實現）。許多良好的軟體工程指南都是這樣規定的。借助NCAM，這種情況很容易發現。一旦確定了效能問題的原因，就可以使用現成的措施糾正問題，包括使用特殊的編譯器最佳化（如鏈接時間最佳化），以及將關鍵的小函數實現轉移到程式標頭中，讓編譯器做出關鍵的內聯決策。

發現效能問題

成功的關鍵在於能夠快速找到代碼中存在潛在效能問題的區域。為此您必須藉助正確的工具，快速的「大海撈針」。下圖顯示了如何使用MetaWare除錯器（MDB）和nSIM在GUI（如下圖）或命令行/批次處理模式下對ARC應用程式進行除錯。在左下角，Profiling 窗格顯示了每個函數的NCAM計數器值。（它還可以構建運行時調用樹，當面對較深的調用棧，並且需要找出哪些子函數是瓶頸時，這一點特別有用）。下拉功能表顯示出一些可以啟用的計數器。

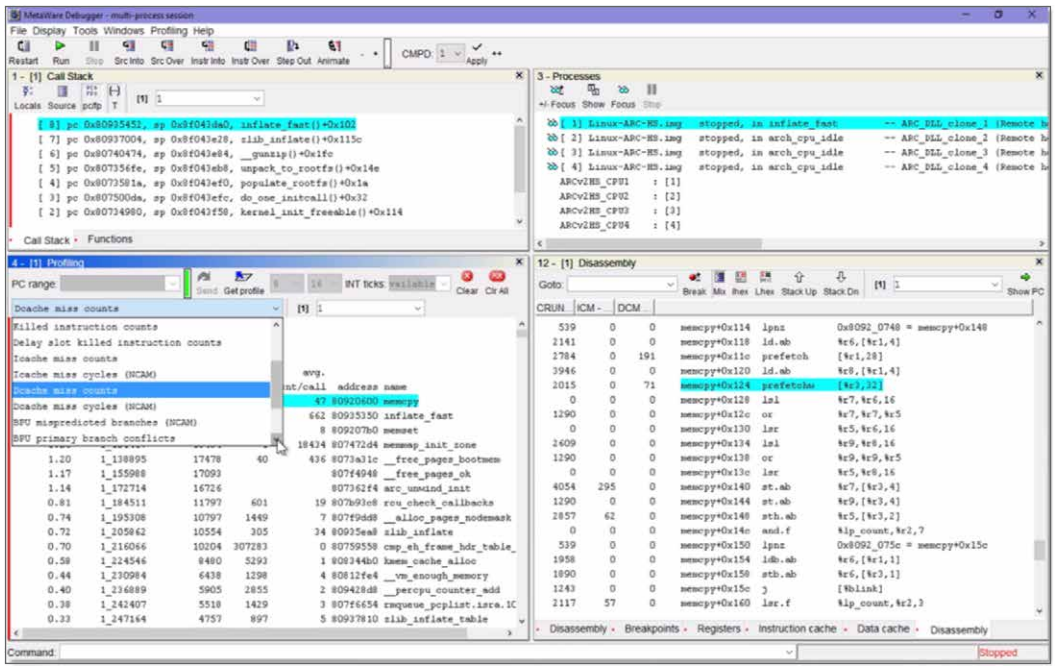

圖2：MDB和NCAM的套用

另一個特性是Source窗格和Disassembly窗格。Disassembly窗格位於右下角。除了實際的反組譯之外，它還可以顯示歸因於單個指令的計數器值。左側的CRUN、ICM和DCM列分別表示週期計數、指令和資料快取未命中。

我們來看另一個KPI，這是預測錯誤的分支，可以從Profiling和Disassembly窗格中的下拉清單中找到。

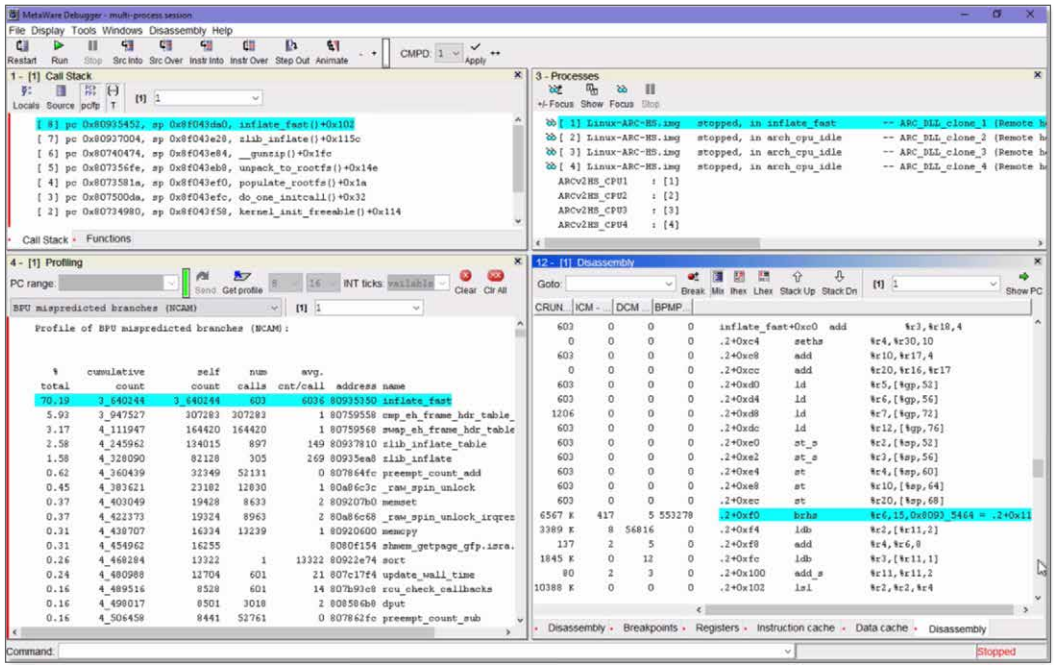

圖3：使用MDB和NCAM檢視分支預測錯誤數

這需要執行兩個步驟的操作：首先在Profiling窗格中查看貢獻最大的函數。然後使用函數名稱符號和Disassembly窗格進一步縮小範圍。大家可以看到，在一個分支指令周圍明顯存在一個瓶頸，它顯示出極高的BPMP（即分支預測錯誤）數。因此，您只需點擊三下即可找到有問題的代碼序列。下一步是找出問題發生的原因，並調整C/C++或組合語言代碼，以消除該特定指令上的停頓（可能是查看if/else或for/while語句，應用一些編譯器控制選項，重新排列代碼等）。對其他經常執行的函數，重複這種做法可以確保效能良好，並且在晶片上測試相同的代碼時不會出現意料之外的結果。

ARC nSIM NCAM—用例

下面，我們透過新思科技經常面對的兩個真實示例來概括和總結NCAM用例。

硬體配置設計空間探索

借助ARC nSIM NCAM，您可以快速查看哪些硬體配置選項會影響目標CPI（每條指令的週期數），並以試算表或其他方便的形式進行匯總，用於估計硬體配置變更對您感興趣的程式的影響。這是關鍵的早期輸入資料，有助於推動決策，以確定指定的硬體配置。

- 例：探索如何對指令快取（I$）、資料快取（D$）和二級快取（L2$）確定大小/組態
  - 容量不足導致的快取未中：
- 增加快取大小
  - 衝突導致的快取未中：
- 增加快取大小
- 增加關聯性（更多路）
- 減少快取行大小（更多群組）

			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%

圖4：使用NCAM追蹤硬體設定選項變更

- 例：探索如何對BPU確定大小/組態
  - 容量導致的分支方向/目標快取未中
    - 增加分支快取大小
  - 更大的轉回位址堆疊（RAS）意味著可以正確預測更深的子函數嵌套等級
  - 更大的模式表意味著可以同時追蹤更多分支或跳轉指令的全域分支歷史
  - 例：探索即時系統中的關鍵處置器常式是否需要
  - CCMs
    - 可預測的效能—費用高
  - 快取架構
    - 較不可預測的效能—費用低

編譯器最佳化空間探索

除了顯而易見的編譯器最佳化選項之外，例如-O1、-O2和-O3，您還可以透過許多更專業的選項來專門調整：

- 代碼密度
- 效能（內聯、展開）

這些選項的交互對於大型複雜軟體棧並非顯而易見，特別是在一些函數具有即時約束（即必須在時間預算內完成）而有些函數則可以隨時被中斷的硬實時系統中。

透過使用ARC nSIM NCAM，您可以執行完整的應用程式以系統地探索編譯器最佳化空間，以深入瞭解最佳化選項的最佳組合，以便獲得如下表所示的資訊（表中顯示了編譯和連結時最佳化的影響）。

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%

圖5：使用NCAM追蹤編譯器選項變更（表檢視）

這對於追蹤和以可視化形式呈現關鍵函數效能隨時間變化的實驗尤其有用，可以查看在所有應用階段（即冷啟動、暖機、工作負載峰值）是否仍能滿足效能要求。

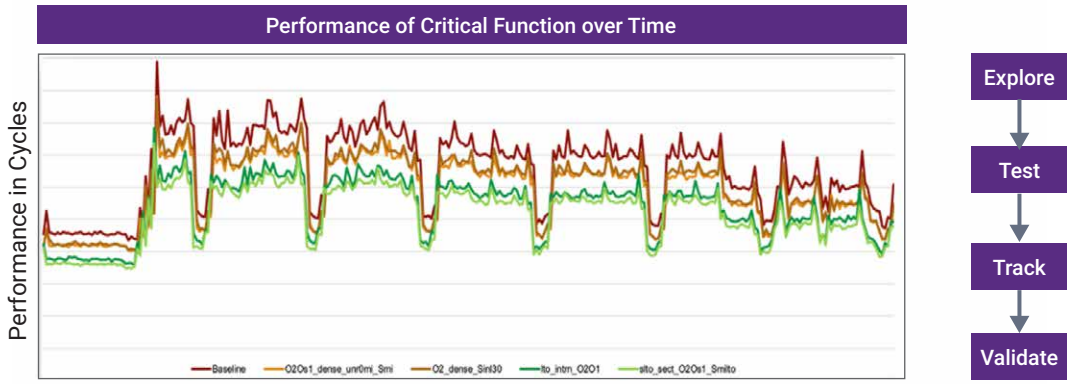

圖6：使用NCAM追蹤編譯器選項變更（圖形檢視）

在上圖中，可以看到-O3最佳化並沒有提供最適最佳化以提高效能。顯然，編譯器最佳化器為-O3產生盡可能快的代碼。通常，效能和代碼大小之間需要折衷考慮—激進的迴圈展開、函數內聯等。如果在純nSIM（指令精確模型）上運行此應用，我們可以為-O3打出最佳分數—更少的子函數呼叫開銷。但是，NCAM考慮更多方面，包括BPU和快取。因此，激進的迴圈展開不適用於這個函數。該函數具有較大的代碼大小，指令快取未命中的代價可能比子函數呼叫或維持迴圈要高。總之，對於複雜軟體的最佳化方法，它可能並不明顯，而且需要多輪才能找到最佳選項。此外，同一應用中的不同函數可能對最佳化有相反的要求。

您可以看到，微調編譯器選項需要大量的工作和測試用例評估，但NCAM可能會揭露一些不明顯的情況，例如在增加指令數確實能夠提高性能時。然後您可以選擇最佳選項，並在較慢的週期精確模型上運行這些選項，以驗證並簽核最佳選擇。

確定最佳編譯器選項集的一般流程

- 選擇最佳化選項的基準
  - 限制應用程式效能預算和代碼大小
  - 為應用程式選擇KPI
- 運行分析
  - 取得KPI的分數
  - 查找瓶頸函數
- 調整最佳化選項。最佳化方法取決於KPI和應用目標。例如：
  - 大量指令快取未命中—最佳化快取設定（硬體最佳化）；降低展開/內聯級別或變更函數位址/重新排列函數（軟體最佳化）
  - 大量資料快取未命中—最佳化快取未命中（硬體最佳化）；變更資料物件位址/重新排列資料物件，使用預取（軟體最佳化）
- 重複這些步驟
- 如果您的應用具有不同的操作模式，請切換到其他模式，並重複所有步驟

該流程可讓您找到一組接近最佳的硬體配置（如果適用）或工具鏈和軟體最佳化選項。該流程能夠與NCAM一起模擬全功能應用或大型函數集合。對於較小的單個函數和內核（例如微基準測試），最好使用週期精確的ARC xCAM或其他週期精確的模擬策略（例如FPGA）。

ARC nSIM NCAM—結語

實踐證明，週期近似NCAM模擬的功能非常強大，已成為硬體/軟體協同設計和軟體開發過程中非常有價值的一部分。儘管它如此有效，但瞭解其局限性也很重要，因為在某些情況下，如果要求獲得的結果的置信度很高（例如，幾乎等同於週期精確），則不建議使用NCAM。以下簡單規則有助於瞭解NCAM是否適用：

何時使用NCAM

探索硬體配置空間

- 快速找到適用於您的應用的良好硬體配置集
- NCAM可輕鬆配置，以供快速進行試驗

- 應用程式最佳化（晶片生產前和晶片生產後）
  - 快速找到運行時佔時最多的函數
  - 集中精力在有可能改善的地方
- 編譯器最佳化
  - 系統地探索編譯器巨大的最佳化空間
  - 找到良好的最佳化選項

何時不使用NCAM

- 分析小型的微基準
  - 微基準通常探索硬體的極端情況，需要的精度較高
  - NCAM不是為該用例而專門設計的—應該使用ARC xCAM、RTL模擬或FPGA
- 效能簽核
  - 如果未在周期精確模式下執行效能簽核，不得進行流片
  - NCAM是近似模擬，簽核要求確定性和準確性—應該使用ARC xCAM、RTL模擬或FPGA

01/13/22.CS817256242 SG-NCAM WP-trad-chinese.

Pub: May 2021