O-Chip：意圖—決策—執行解耦運算架構

v2.0 / 2026-07-30 / Neo.K（許筌崴） / 公開概念技術論文／可驗證運算架構提案

證據狀態：E0 架構提案；尚未完成 O-Chip 專用矽、封裝整合或第三方重現

https://thisoneisneok.com/html/papers/ccp-15.html

SECTION

從「維度代理人」到可驗證的跨層編排與硬體提示控制

O-Chip: Intent–Decision–Execution Decoupled Computing Architecture

From a “Dimensional Agent” to Verifiable Cross-Layer Orchestration and Hardware-Guidance Control

作者：Neo.K（許筌崴）

機構：一言諾科技有限公司（EveMissLab），台灣

原始版本：2025 年 12 月

公開修訂版：v2.0，2026 年 7 月 30 日

文件性質：公開概念技術論文／可驗證運算架構提案

建議全名：O-Chip（Orchestration Chiplet，編排晶粒／編排控制晶片）

系列定位：工作負載意圖、作業系統、SynCore 執行模式、DEO 運作包絡、DPCPS 供電、DryCore 熱管理與 SDMCA 空間拓撲之間的跨層協同控制面

證據狀態：E0 架構提案；尚未完成 O-Chip 專用矽、封裝整合或第三方重現

SECTION

修訂聲明

本文由《O-Chip 維度代理人：靈肉分離的運算革命》重構而來。原稿最有價值的洞察，是注意到現代運算系統的「做什麼」「在哪裡做」「何時做」「以何種功率與資料配置做」不必全部由同一個執行核心在最後一刻決定。編譯器、執行期、作業系統、裝置韌體、功率控制器與應用程式本來就掌握不同時間尺度的資訊；若能把意圖、決策與執行重新分層，系統可能減少資料搬移、錯誤資源配置、尾端延遲與跨裝置等待。

然而，原稿把這個洞察推進到數個不成立或證據不足的結論：外接 PCIe 卡可攔截並重組 CPU 動態指令流；O-Chip 可直接向 CPU L1 快取寫入資料；通用 CPU 可移除分支預測器與大部分亂序執行；外部 AI 可提前五至十秒精確預知任意程式的微指令；「高維坍縮」可取代具體的編譯、排程與一致性協議；並以未建立的硬體平台宣稱固定幀率、功耗與溫度改善。這些敘述混淆了 CPU 核心內的週期級微架構決策、封裝內的硬體提示、作業系統的執行緒排程，以及應用層的任務圖編排。

v2.0 因此進行以下重構：

- 將 O-Chip 從「維度代理人／統一主系列 AI」重新定義為 Orchestration Chiplet。它是受約束的跨層編排控制面，不是具有全知能力的外部心智。
- 建立決策時限分層。分支預測、暫存器相依、發射與旁路等週期級決策保留於核心內；微秒級資源提示可由封裝內控制層處理；毫秒以上的任務圖、執行緒、裝置、記憶體與功率編排才是 O-Chip 的主要範圍。
- 保留通用核心的分支預測、亂序執行與快取階層。O-Chip 可提供任務提示、軟體分支預解析、預取描述符或核心類型建議，但不能以外接控制器普遍取代核心內反射機制。
- 取消 PCIe 卡攔截 CPU 微指令流的路線。PCIe／DPU／FPGA 原型只處理明確提交的工作佇列、遙測、I/O、記憶體搬移與粗粒度任務，不假裝位於 CPU 取指與退休路徑之間。
- 取消任意寫入 L1 快取的介面。資料交換改用一致性記憶體、共享虛擬記憶體、受控 scratchpad、DMA／CXL 記憶體語義及明確擁有權轉移。
- 將「超指令包」改為任務束與命令圖。真正的微指令融合屬於編譯器、JIT、ISA 與核心前端；O-Chip 可提交的是具有依賴、資料位置、期限與回退資訊的粗粒度描述符。
- 將「預知未來」改為多時間尺度工作負載推斷。優先使用明確應用提示與任務圖，其次才使用規則、統計或機器學習；模型必須輸出信心、不確定性與校準域。
- 加入安全、權限、隔離與回退。O-Chip 不得凌駕硬體熱保護、IOMMU、作業系統隔離或即時安全控制；控制失敗時必須退回平台預設排程器與功率控制器。
- 正式接入系列其他架構。O-Chip 負責意圖與計畫；DEO 選擇合格運作包絡；SynCore 執行；DPCPS 與 DryCore 分別提供功率與熱證書；SDMCA 提供可用資源拓撲。

- 移除所有無原型支撐的固定百分比與時程承諾，改用公平基線、消融實驗、可否證命題與 E0–E5 證據分級。

因此，新版保留的不是「AI 靈魂控制 CPU 肉體」的字面工程宣稱，而是更精確的命題：不同時間尺度的決策應放在能及時觀察、合法控制且可安全回退的層級；計算執行與跨層編排可以解耦，但不能無視延遲、語義、權限與一致性。

SECTION

摘要

現代異質運算系統同時包含通用 CPU、效率核心、GPU、NPU、記憶體控制器、網路與儲存加速器、功率管理單元及多層軟體。每一層都做出部分決策：CPU 微架構預測下一條控制流；硬體回饋介面描述核心能力；作業系統決定執行緒放置；執行期建立任務依賴；應用程式知道即將到來的幀、批次或服務期限；電源與冷卻控制器掌握可用功率及熱容量。問題不在於「決策與執行從未分離」，而在於這些決策缺乏共同語義、時間尺度與可驗證協議，導致局部最佳化互相衝突。

本文提出 O-Chip，即意圖—決策—執行解耦運算架構。O-Chip 將工作負載表示為帶有資料位置、依賴、期限、品質需求與權限的任務圖，將硬體表示為帶寬、延遲、功率、溫度、健康度與故障域組成的資源圖，再由受約束的計畫器決定任務放置、裝置選擇、資料預置、佇列優先序、DEO 運作包絡請求及 SynCore 模式。控制器不直接生成任意 CPU 微指令，也不取代分支預測、亂序發射與快取一致性；其主要作用時間尺度是數十微秒至數秒，只有經過共設計的封裝內版本才可逐步進入更細粒度控制。

本文建立六層架構：核心反射層、硬體回饋層、O-Chip 即時編排層、作業系統與 O-Runtime 層、應用意圖層，以及長期模型與證據層。每個決策必須滿足「決策時限條件」：觀察、推斷、通訊、執行與回退總延遲，必須顯著短於該決策能產生價值的時間窗口。本文同時提出能力受限的命令描述符、可信遙測、計畫證書、IOMMU／一致性記憶體邊界、看門狗與平台預設回退。

O-Chip 可分四種實作輪廓。O-Soft 是可立即建立的 Linux 軟體原型，使用 sched_ext、效能計數器、cgroup、resctrl、CPUFreq 與明確應用提示進行任務放置和資源控制。O-Card 是 PCIe／DPU／FPGA 原型，適合遙測彙整、I/O、網路、儲存、命令佇列與粗粒度資料流，不攔截 CPU 指令。O-Die 是封裝內 chiplet，透過 UCIe、CXL 或專用一致性介面取得較低延遲的共享記憶體與硬體回饋。只有在 O-Core 共設計階段，才可能研究特定工作負載的軟體分支預解析、helper thread、runahead 或局部前端簡化。

本文的可檢驗主張不是「O-Chip 必然提高所有程式效能」，而是：對具有可觀察階段、顯式任務圖、異質裝置與資料搬移成本的工作負載，跨層意圖與受約束編排可能降低尾端延遲、無效搬移、資源震盪與功率—熱控制衝突。若 O-Chip 的通訊、推斷、遷移與回退成本大於其節省，或在嚴格基線下不能勝過作業系統預設、靜態配置與 profile-guided 方法，則相應控制層應被否證或縮減。

關鍵詞：硬體—軟體協同設計、異質運算、任務圖、硬體回饋、Thread Director、DPU、SmartNIC、UCIe、CXL、helper thread、runahead、預取、可擴展排程器、預測控制、SynCore、DEO

SECTION

1.1 現代處理器的決策從未集中於單一位置

原稿將分支預測、亂序執行、快取管理與作業系統排程統稱為「CPU 在猜下一步」，並認為這些控制邏輯可以整體搬到另一顆 AI 晶片。這種描述忽略了不同決策的時間尺度與資訊來源。

現代系統至少包含：

- 核心內週期級決策：取指、分支預測、相依追蹤、發射、旁路、暫存器重命名、快取存取與退休；
- 晶片或封裝內微秒級決策：核心能力回饋、功率分配、記憶體控制、硬體預取、服務品質與裝置佇列；
- 作業系統微秒至毫秒級決策：執行緒喚醒、CPU 放置、優先序、NUMA、時間片與資源隔離；
- 執行期毫秒至秒級決策：任務圖、批次、流水線、裝置選擇、資料位置與容錯；
- 應用與服務秒級以上決策：下一幀、下一批查詢、品質等級、截止時間與能源政策。

這些決策不是同一個問題。將它們全部集中到外部 AI，不會自動降低複雜度，反而可能產生更長的觀察—決策—執行閉環。

SECTION

1.2 延遲階層決定控制器可以做什麼

令某項決策 d 的有效時間窗口為 H_d，完整控制閉環延遲為：

$$
L_d
=
L_{observe}
+
L_{infer}
+
L_{communicate}
+
L_{actuate}
$$

若：

$$L_d \geq H_d,$$

則控制器即使做出完美決策，也已經錯過時機。

對通用 CPU 的下一個分支或發射槽，H_d 常接近數個時脈週期。外部 PCIe 卡、作業系統程序或大型神經網路無法在這個窗口內完成往返。相反地，對下一個遊戲幀、推論批次、編譯階段、I/O 管線或數秒級熱預算，較高層控制器有足夠時間整合更多資訊。

本文使用較保守的設計條件：

$$L_d \leq \eta H_d, \quad 0<\eta<1,$$

並要求扣除控制成本後的預期收益為正：

$$bb\text{-}E[B_d] > C_{observe} + C_{infer} + C_{move} + C_{switch} + C_{rollback}.$$

這兩個條件共同決定決策應放在核心、封裝、作業系統、O-Runtime 或應用層。

SECTION

1.3 為何不能普遍移除分支預測與亂序執行

分支預測不是一個可慢速外包的排程服務。核心前端必須在分支結果尚未算出前，選擇下一段取指位置；若等待外部控制器回覆，前端會直接停頓。亂序執行則利用已知相依關係，在某些指令等待記憶體或運算結果時執行其他已就緒指令。它同樣是低延遲、與暫存器和旁路網路緊密耦合的機制。

這不表示既有微架構不能改善。研究已展示：

- 軟體可在特定迴圈中預先計算難預測分支結果；
- helper thread 可提前執行資料位址或預取；
- runahead execution 可在長延遲停滯期間探索未來記憶體存取；
- decoupled access/execute 可把資料存取與計算拆成協同流。

但這些方法的共同點，是針對特定控制流或記憶體問題進行硬體—軟體共設計，不是把整個通用核心變成沒有控制能力的「執行殭屍」。

SECTION

1.4 O-Chip 的合法範圍

O-Chip 的主要研究範圍是：

- 應用意圖與任務圖；
- 執行緒、程序、容器與裝置放置；
- CPU／GPU／NPU／DPU 工作分配；
- 資料預置、記憶體層級與 I/O 管線；
- DEO 運作包絡與 SynCore 模式請求；
- 功率、熱、健康度與故障域協同；
- 計畫驗證、回退與長期證據累積。

其不直接負責：

- 每週期分支預測；
- 暫存器重命名；
- 任意微指令注入；
- 未經一致性協議的 L1 快取寫入；

- 凌駕硬體保護的電壓與時脈控制；
- 無限制監看使用者行為。

SECTION

2.1 硬體回饋與作業系統排程

Intel Thread Director 已展示一種重要模式：硬體持續觀察執行緒指令混合與核心狀態，再向作業系統提供放置建議；最終排程決策仍由作業系統完成。這說明硬體與排程器可以共享比靜態核心類型更細緻的執行資訊，但也說明「硬體提示」與「硬體接管作業系統」是兩件不同的事。[1][2]

Linux sched_ext 進一步提供可由 BPF 定義的排程器類別，並內建錯誤回退：當 BPF 排程器失效或任務停滯，核心可恢復預設排程行為。這種可替換、可觀察、可回退的介面，非常適合作為 O-Soft MVP 的起點。[3]

O-Chip 的差異不能只是「另一個 Thread Director」。它必須把硬體回饋與以下資訊結合：

- 應用明確任務圖；
- 資料位置與搬移成本；
- 加速器能力；
- 功率和熱證書；
- 故障域與可靠度；
- 任務期限、品質與回退策略。

SECTION

2.2 DPU 與 SmartNIC：控制面分離的現實案例

NVIDIA BlueField 等 DPU 已將網路、儲存、安全與基礎設施控制的一部分從主機 CPU 移到具有 Arm 核心、可程式化資料路徑和專用加速器的裝置。這證明「主機執行應用，另一顆處理器管理資料與基礎設施」是可行產品方向。[4]

但 DPU 不位於 CPU 的取指、解碼與退休路徑。它能處理的是：

- 網路封包與虛擬交換；

- 儲存與資料服務；
- 安全、隔離與遙測；
- 明確提交的加速任務；
- DMA 與裝置記憶體管理。

因此，O-Card 可以借鑑 DPU，但不能宣稱透過普通 PCIe 驅動「攔截 CPU 發出的所有指令」。若工作沒有被應用、編譯器或執行期明確提交，外接卡通常看不到其完整語義。

SECTION

2.3 Decoupled Access/Execute、runahead 與 helper thread

Decoupled Access/Execute 架構將位址產生與資料存取從運算流分開，以佇列協調兩者，使記憶體操作可提前進行。Runahead execution 則在核心因長延遲存取停頓時，暫時向前執行以產生未來快取缺失。Helper thread 研究進一步讓編譯器或執行期產生輔助工作，提前預取資料或計算控制結果。[5][6][7]

這些研究為 O-Chip 提供三個可靠原則：

- 提前做什麼必須有可辨識的依賴鏈；
- 輔助工作本身會占用算力、頻寬與能源；
- 預執行、錯路徑與共享微架構狀態會帶來安全風險。

因此，O-Chip 的「預測」必須通過收益估計、資源預算與安全策略，不是預測越多越好。

SECTION

2.4 UCIe 與 CXL：封裝內與一致性裝置的合理接口

UCIe 提供開放 die-to-die 互連與 3D 封裝支援；其後續規格也強化管理、測試、除錯與系統級功率能力。CXL 則在 PCIe PHY 上提供 I/O、快取一致性與記憶體協議，使加速器與記憶體裝置可以在受控一致性模型下共享資料。[8][9][10]

這些介面讓 O-Die 可以：

- 讀取被授權的共享記憶體；

- 維護裝置快取或使用 CXL 記憶體；
- 接收命令佇列和遙測；
- 與其他 chiplet 進行低延遲資料交換；
- 參與封裝管理、測試與故障回報。

但它們仍不等於「任意寫入 CPU L1」或「直接控制核心每一條微指令」。一致性、權限、順序與錯誤處理必須由協議與 home agent 管理。

SECTION

2.5 尚未被統一的缺口

既有技術分別處理：

- 核心內推測與記憶體延遲；
- OS 執行緒放置；
- DPU 基礎設施卸載；
- chiplet 一致性互連；
- 功率與熱控制；
- 應用任務圖。

O-Chip 的研究缺口是：建立一個跨層但權限受限的意圖與計畫協議，使這些控制器不再互相猜測，而能交換可驗證提示、資源證書、期限與回退條件。

SECTION

3.1 決策安置原則

對決策 d，定義：

- H_d：有效時間窗口；
- O_d：可觀察資訊集合；

- A_d：合法可執行動作集合；
- L_d：閉環延遲；
- R_d：錯誤決策風險；
- G_d：預期淨收益。

  決策層 ℓ 合格的必要條件為：

> O_d⊆ O_(ℓ),
>
> A_d⊆ A_(ℓ),
>
> L_(ℓ,d)<H_d.

並要求：

> G_(ℓ,d)
> =
> bb-E[B_(ℓ,d)]
> -
> bb-E[C_(ℓ,d)]
> -
> λ R_(ℓ,d)
> >0.

若高層模型擁有更多資訊，但延遲太長，則應把策略壓縮成較低層可快速執行的規則，而不是讓高層模型介入每個事件。

SECTION

3.2 六個決策時間尺度

時間尺度	典型決策	合理位置	O-Chip 角色
亞奈秒至數奈秒	分支、發射、旁路、快取命中	CPU 核心	不介入；只可離線共設計

數十奈秒至數微秒　預取、硬體佇列、核心能力回饋　核心／SoC／封裝內控制器　接收或下發窄義提示

數十微秒至數毫秒　執行緒放置、裝置佇列、資料預置　O-Die／核心／OS　主要控制區間

數毫秒至數秒　幀、批次、服務階段、DEO 包絡　O-Runtime／O-Chip　任務圖與資源編排

數秒至數分鐘　熱容量、模型校準、故障降級　系統控制面　跨模組協同

小時至生命週期　老化、策略更新、證據累積　離線／管理平面　不在線直接控制關鍵路徑

O-Chip 不追求把所有決策搬到單一層，而是提供一組跨層翻譯與承諾機制。

SECTION

3.3 「靈肉分離」的工程化翻譯

原稿的隱喻可以保留為概念史，但新版將其翻譯為：

- 意圖層：描述需要完成的工作與品質；
- 決策層：在約束下選擇資源、資料位置和控制策略；
- 執行層：由 CPU、GPU、NPU、DPU、記憶體與 I/O 裝置完成具體運算；
- 證據層：證明計畫是否按假設執行，並決定是否回退。

  因此：

> Intent
> ≠
> Plan
> ≠
> Execution
> ≠
> Evidence.

四者解耦後仍需明確協議連接，不能以「坍縮」一詞跳過中間工程。

SECTION

4.1 工作負載任務圖

定義工作負載圖：

> cal-G_W(t)
> =
> bigl(V_W,E_W,m_W(t)bigr).

其中每個節點 v⊠∈ V_W 表示一個可調度任務，邊 e_(ij)∈ E_W 表示資料或控制依賴。節點中繼資料可以包含：

> m⊠
> =
> (c⊠,m⊠,b⊠,d⊠,q⊠,s⊠,r⊠),

分別表示估計計算量、記憶體需求、輸入輸出位元組、期限、品質等級、安全域與可回退策略。

任務可以來自：

- 編譯器產生的基本任務；
- GPU／NPU command graph；
- 遊戲引擎 frame graph；
- AI 推論 pipeline；
- 儲存與網路服務圖；
- 使用者顯式 API；
- 執行期依 trace 推斷的階段。

SECTION

4.2 資源圖

定義動態資源圖：

> cal-G_R(t)
> =
> bigl(V_R,E_R,m_R(t)bigr).

節點包括 CPU 核心群、GPU、NPU、DPU、記憶體層、儲存、網路、電源區與冷卻區；邊表示可用互連。每個資源節點 r□ 具有：

$$
\begin{aligned}
&m_{r□}(t) \\
&= \\
&\bigl(\\
&C□, \\
&M□, \\
&B□, \\
&L□, \\
&P□, \\
&T□, \\
&H□, \\
&F□ \\
&\bigr),
\end{aligned}
$$

分別表示計算能力、記憶體容量、頻寬、延遲、可用功率、熱狀態、健康度與故障域。

SECTION

4.3 意圖向量

應用或系統提交意圖：

$$
\begin{aligned}
&i \\
&= \\
&\bigl(\\
&\tau_{max}, \\
&q_{min}, \\
&E_{max}, \\
&R_{min}, \\
&\text{cal-S}, \\
&\text{cal-P} \\
&\bigr),
\end{aligned}
$$

其中：

- textau_(max)：延遲或期限；
- q_(min)：最低品質或吞吐要求；
- E_(max)：能耗或功率預算；
- R_(min)：可靠度要求；
- cal-S：安全與資料主權限制；
- cal-P：允許的降級與回退政策。

  應用不應直接要求危險電壓或未經認證的硬體狀態，而應表達可驗證的目標。

SECTION

4.4 部分可觀察狀態與信念

O-Chip 無法完整知道未來工作與所有硬體內部狀態。令真實狀態為 s□，觀察為 o□，則控制器維持信念分布：

> b□(s)
> =
> P(s□=smid o_(0:t),a_(0:t-1)).

信念可由明確任務圖、效能計數器、佇列深度、快取／記憶體遙測、溫度、功率與歷史階段更新。若觀察不足，系統應輸出「未知」而不是補成確定預測。

SECTION

4.5 計畫

O-Chip 生成計畫：

> Π□
> =
> bigl(
> μ□,

其中：

- μ⊠：任務到資源的映射；
- ρ⊠：資料位置與搬移方案；
- κ⊠：佇列、優先序與同步策略；
- δ⊠：DEO 運作包絡請求；
- ε⊠：SynCore 模式與執行提示；
- φ⊠：失敗、逾時與回退方案。

SECTION

4.6 多目標函數

計畫器最小化：

$$
J(\Pi) = \alpha L_{p99} + \beta E_{sys} + \gamma D_{move} + \delta C_{switch} + \eta V_{thermal} + \zeta R_{failure} + \xi U_{uncertainty}.
$$

其中：

- L_(p99)：尾端延遲；
- E_(sys)：計算、記憶體、互連、供電與冷卻總能耗；
- D_(move)：資料搬移成本；
- C_(switch)：遷移、模式與包絡切換成本；
- V_(thermal)：熱違規與熱耦合；
- R_(failure)：失敗和可靠度風險；
- U_(uncertainty)：模型不確定性。

SECTION

4.7 機率約束與拒絕權

對關鍵限制 g⊠，要求：

> Pbigl(g⊠(s⊠,Π⊠)≤ 0bigr)
> ≥
> 1-ε⊠.

若模型無法滿足約束或信心低於門檻，O-Chip 必須：

- 使用保守計畫；
- 向應用要求更多提示；
- 交回作業系統或硬體預設控制；
- 拒絕不安全的計畫。

SECTION

5.1 L0：物理安全與最終否決層

L0 包含：

- 熱關機與電流保護；
- PMIC、時脈與電壓護欄；
- ECC、RAS 與故障回報；
- IOMMU、記憶體保護與裝置重設；
- watchdog；
- 安全啟動與韌體信任根。

O-Chip 只能提出請求，不能關閉 L0。若 O-Chip 與 L0 衝突，L0 優先。

SECTION

5.2 L1：核心反射層

L1 保留：

- 分支預測；
- 取指與解碼；
- 暫存器重命名；
- 亂序發射與退休；
- 快取與 TLB；
- 核心內硬體預取；
- 短延遲中斷與例外。

O-Chip 可以提供低權限提示，例如工作階段標籤、預取描述符、核心類型偏好或被編譯器驗證的分支預解析資料；核心可以忽略或撤銷提示。

SECTION

5.3 L2：硬體回饋與遙測層

L2 將異質硬體狀態轉為統一遙測：

- 指令混合與停滯類型；
- IPC、快取／TLB 失誤；
- 記憶體和互連頻寬；
- 佇列深度；
- 裝置利用率；
- 功率、溫度與熱餘裕；
- 錯誤率與健康度；
- 故障域與可維修狀態。

遙測必須包含時間戳、來源、精度、權限域與失效狀態。缺測不能被默認為零。

SECTION

5.4 L3：O-Chip 即時編排層

L3 是 O-Die 或 SoC 控制 tile 的主要位置，處理數十微秒至毫秒級決策：

- 任務與核心群放置；
- 裝置命令佇列；
- 資料預置與記憶體層級；
- 短期階段辨識；
- DEO 包絡與 SynCore 模式請求；
- 逾時、健康度與快速回退。

其實作可以是小型 RISC-V／Arm 控制核心、硬體狀態機、特徵提取器與有限大小模型的組合，不要求大型 Transformer 常駐關鍵路徑。

SECTION

5.5 L4：O-Runtime 與作業系統層

L4 負責：

- 建立與更新任務圖；
- sched_ext 或平台排程整合；
- cgroup、cpuset、NUMA 與優先序；
- resctrl／MPAM 類快取與頻寬控制；
- CPUFreq／PM QoS 與 DEO 介面；
- GPU／NPU／DPU 命令圖；
- 程序、容器與租戶隔離；
- 計畫日誌與重現。

L4 可執行較複雜計畫器，但必須把快速策略編譯為 L3 可執行的有限規則。

SECTION

5.6 L5：應用意圖與編譯層

L5 由應用、編譯器與框架提供：

- 任務依賴；
- 預計批次、幀或階段；
- 資料大小與位置；
- 期限和服務品質；
- 可接受的品質降級；
- 可重試性與冪等性；

- 安全和隱私限制。

明確提示通常比從黑箱 trace 猜測更可靠。O-Chip SDK 應讓應用能逐步採用，不要求重寫所有程式。

SECTION

5.7 L6：模型、證據與長期治理層

L6 儲存：

- 平台校準模型；
- 工作負載階段模型；
- 策略版本與簽章；
- 實驗資料；
- 失敗與回退紀錄；
- 可靠度帳本；
- E0–E5 證據狀態。

L6 不應在未審核情況下自動改寫關鍵安全策略。學習模型的更新必須可回滾、可比較且可重現。

SECTION

6.1 意圖編譯器

意圖編譯器把應用 API、編譯器 metadata、OS 政策與服務等級轉為標準化任務描述。它必須區分：

- 硬約束；
- 軟偏好；
- 預測；
- 預設值；
- 未知。

例如「此幀必須在 16.7 ms 內完成」可以是硬期限；「優先使用效率核心」是軟偏好；「下一批可能為 32」是預測，不能混為一談。

SECTION

6.2 遙測融合器

遙測融合器對齊不同頻率、延遲與誤差的訊號。令第 k 個感測來源為：

> y□(t)
> =
> h□(s(t-Δ□))+n□(t),

其中 Δ□ 是遙測延遲，n□ 是噪聲。若控制器忽略 Δ□，可能根據過時溫度或佇列做出錯誤決策。

SECTION

6.3 階段辨識器

階段辨識可以使用：

- 明確事件與 API；
- 規則和有限狀態機；
- EWMA／變化點檢測；
- 隱馬可夫模型；
- 決策樹或小型神經網路；
- 圖模型或較大型離線模型。

  使用最簡單且能達成校準目標的方法。模型複雜度不是產品價值本身。

SECTION

6.4 計畫合成器

計畫合成器可使用：

- 啟發式；
- 整數規劃；
- 最短路徑／最大流；
- 模型預測控制；
- 貝葉斯最佳化；
- 學習型策略。

  任何學習型策略都必須通過安全投影：

> Π_(safe)
> =
> Proj_(cal-C_(safe))
> bigl(Π_(model)bigr).

SECTION

6.5 資源仲介器

資源仲介器處理所有權與承諾：

- 哪個任務可使用哪個裝置；
- 記憶體是否已映射；
- 功率與熱預算是否被接受；
- 頻寬是否預留；
- 計畫何時生效；
- 何時到期；
- 誰可以撤銷。

沒有資源承諾的計畫，只是建議，不是可執行契約。

SECTION

6.6 計畫證書與回退管理器

每份計畫證書至少包含：

plan_id
policy_version
workload_identity
resource_snapshot
assumptions
validity_window
authorized_actions
expected_cost
uncertainty
safety_guards
rollback_target
telemetry_requirements
若任何關鍵假設失效，證書立即過期，系統執行回退。

SECTION

7.1 四種訊息不可混用

O-Chip 協議區分：

- Hint：可忽略提示；
- Request：需由資源管理器接受或拒絕；
- Commitment：在期限內保證的資源；
- Command：已授權且具明確完成語義的動作。

  例如：

- 「此執行緒可能適合效能核心」是 Hint；
- 「請求 E3 爆發包絡 20 ms」是 Request；

- 「DPCPS 承諾 20 ms 內提供指定電流裕度」是 Commitment；
- 「將命令圖 42 提交至 NPU 佇列」是 Command。

SECTION

7.2 任務束取代「微指令注入」

新版定義 O-Bundle：

> cal-B
> =
> bigl(
> V_B,E_B,D_B,Q_B,T_B,F_B
> bigr),

其中包含任務、依賴、資料描述符、品質、期限與失敗策略。O-Bundle 可以映射到：

- CPU thread pool；
- GPU graph；
- NPU command queue；
- DPU pipeline；
- 儲存或網路工作佇列。

它不包含未經 ISA 與核心驗證的任意微指令。若要進行指令融合，應由編譯器、JIT、microcode 或正式 ISA 擴充完成。

SECTION

7.3 記憶體語義

O-Chip 可使用三種資料路徑：

顯式訊息與 DMA

適用 O-Card。應用或執行期提交 buffer 描述符，由 IOMMU 控制存取。優點是邊界清楚；缺點是搬移與同步成本高。

一致性共享記憶體

適用 CXL.cache／CXL.mem 或 SoC 一致性互連。O-Chip 可讀寫被授權的 cache line，但一致性仍由協議管理，不能繞過 home agent 或隨意更改核心私有 L1 狀態。

受控 scratchpad／mailbox

適用封裝內快速控制。資料結構固定、容量有限、容易驗證；適合遙測、計畫、提示與短描述符。

SECTION

7.4 分支與預取提示

O-Chip 可提供：

- 編譯器識別的難預測分支結果流；
- pointer-chasing 預取描述符；
- 任務即將進入記憶體密集階段的標籤；
- 預計使用的資料集合；
- 可取消的 helper thread 任務。

  但核心必須檢查時序與對應關係，錯誤提示不得破壞正確性。

SECTION

7.5 一致性與提交

O-Chip 計畫需要兩階段或可撤銷提交：

- Prepare：資源檢查、映射、功率與熱證書；
- Commit：計畫生效並記錄版本；

- Observe：監測偏差；
- Abort／Rollback：逾時或失效時恢復安全基線。

  對不可重試任務，必須使用更嚴格的事務、冪等或檢查點機制。

SECTION

8.1 O-Soft：軟體優先的最小版本

O-Soft 不需要新晶片。它由：

- O-SDK；
- O-Runtime；
- 遙測 daemon；
- sched_ext 排程器；
- cgroup／cpuset／NUMA 控制；
- resctrl／MPAM 類資源控制；
- CPUFreq／PM QoS／DEO 介面；
- GPU／NPU／DPU adapter；
- 實驗與證據記錄器；

  組成。

  O-Soft 可以驗證最核心的問題：應用意圖和跨層策略是否真的比 OS 預設與靜態調優更有價值。

SECTION

8.2 O-Card：PCIe／DPU／FPGA 粗粒度控制器

O-Card 可負責：

- 高速遙測彙整；

- I/O、儲存和網路資料流；
- 任務佇列與描述符驗證；
- 壓縮、加密、封包與資料轉換；
- DMA 預置與 buffer 管理；
- 粗粒度工作負載分類；
- 作為 O-Soft 的硬體在迴路控制器。

O-Card 不負責：

- 攔截 CPU 取指；
- 逐分支決策；
- 直接覆寫 L1；
- 取代核心亂序引擎；
- 對沒有明確提交的應用任務進行透明重組。

它最適合 I/O 密集、流水線清楚且工作單元大於通訊往返成本的場景。

SECTION

8.3 O-Die：封裝內編排 chiplet

O-Die 可透過 UCIe、CXL 或專用一致性互連接近 CPU、GPU、NPU 和記憶體。其硬體可包含：

- 遙測與事件聚合器；
- 命令佇列；
- 小型控制核心；
- scratchpad；
- 規則引擎；

- 低延遲階段分類器；
- 資源承諾表；
- 安全監督器；
- trace buffer。

O-Die 的價值必須來自更低延遲與更直接的硬體回饋，而不是單純把 O-Soft 搬到封裝內。

SECTION

8.4 O-Core：特定核心共設計

O-Core 是最遠期、風險最高的輪廓，只針對特定執行環境研究：

- 軟體分支預解析；
- helper thread engine；
- decoupled access queue；
- runahead／prefetch slice；
- 特定資料流核心；
- 可被 O-Die 配置的前端策略。

只有當量測證明某些核心內資源可由上層提示穩定替代，才考慮縮減硬體。通用 CPU 不應先移除成熟機制再期待 O-Chip 補救。

SECTION

9.1 明確提示優先於黑箱預測

O-Chip 的資訊優先序為：

- 應用明確任務圖與期限；
- 編譯器／JIT metadata；

- 作業系統與裝置佇列；
- 規則和階段模型；
- 統計預測；
- 機器學習補充。

若應用已知道「下一幀即將開始」或「下一批張量大小」，不應先用大型模型從硬體計數器猜測同一件事。

SECTION

9.2 多時間尺度預測

令預測地平線為 h：

- h≈ 10–100 μ s：佇列、短期停滯、資料預取；
- h≈ 0.1–10 ms：執行緒、裝置與幀階段；
- h≈ 10 ms–1 s：批次、功率與熱爆發；
- h>1 s：冷卻、容量、服務負載與故障趨勢。

不同地平線應使用不同模型。不存在一個模型同時精確預測下一個分支與五秒後完整工作流。

SECTION

9.3 預測信心與校準

對預測 $

\hat y，模型同時輸出信心c與域內分數d_{\mathrm{in}}$：

> (
> hat y,c,d_(in))
> =
> f_(θ)(o_(0:t)).

只有當：

$$
\begin{aligned}
&c \geq c_{(min)} \\
&\quad \wedge \\
&d_{(in)} \geq d_{(min)}
\end{aligned}
$$

時，系統才允許較積極計畫；否則使用保守策略。

SECTION

9.4 後悔與回退成本

定義單次決策後悔：

$$
\begin{aligned}
&r\boxtimes \\
&= \\
&C(a\boxtimes,s\boxtimes)-C(a\boxtimes^{(*)},s\boxtimes).
\end{aligned}
$$

實際系統不知道 $a\boxtimes^{(*)}$，可用事後最佳基線或 counterfactual 模型估計。O-Chip 不只追求平均預測準確率，也必須限制高成本錯誤：

$$
\begin{aligned}
&CVaR_{(\alpha)}(r\boxtimes) \\
&\leq \\
&r_{(max)}.
\end{aligned}
$$

SECTION

9.5 線上學習邊界

線上學習不得直接改寫：

- 熱關機門檻；
- IOMMU 規則；
- 租戶隔離；

- 電壓安全曲線；
- 不可回復資料操作。

新策略先在影子模式評估，再經 canary、限制權限與版本化部署。

SECTION

10.1 真正可以減少的是工作，不是物理過程本身

原稿的核心口號「不是做得更快，而是做得更少」可以保留，但必須具體化。可減少的工作包括：

- 重複資料格式轉換；
- CPU、GPU 與 NPU 間不必要的來回搬移；
- 已知不會被使用的預取；
- 錯誤裝置放置造成的重跑；
- 同一特徵在多層重複計算；
- 資源震盪與頻繁遷移；
- 過度品質和不必要精度；
- I/O 與計算沒有重疊造成的等待。

SECTION

10.2 資料搬移模型

對資料物件 d□，從資源 a 移到 b 的成本為：

C_(move)(d□,a,b)
=
L_(ab)
+
(|d□|)/(B_(ab))
+
E_(ab)|d□|

若遷移後節省的執行成本小於搬移成本，O-Chip 應保持資料原位並選擇相鄰計算資源。

SECTION

10.3 預取的效益條件

對預取 p：

> G_p
> =
> P_(use)L_(saved)
> -
> C_(bandwidth)
> -
> C_(cache-pollution)
> -
> C_(energy).

只有 G_p>0 且安全政策允許時才執行。O-Chip 不應把所有可能資料都提前搬入快取。

SECTION

10.4 快取控制的合理邊界

O-Chip 可以：

- 透過 OS／resctrl 分配 LLC 區域或記憶體頻寬；
- 標記資料生命週期；
- 建議 bypass、streaming 或 keep-hot；
- 在裝置一致性快取中預置資料；
- 使用 scratchpad 保留控制資料。

  O-Chip 不應：

- 無一致性地覆寫私有 L1；
- 假定 L2 可以任意縮小而沒有命中率代價；
- 以軟體模型取代所有硬體預取；
- 對安全敏感資料跨租戶共享快取狀態。

SECTION

11.1 系統流程

> 應用意圖／任務圖
> →
> O-Chip 計畫
> →
>
> DEO 運作包絡;
> SynCore 執行模式;
> DPCPS 功率承諾;
> DryCore 熱承諾;
> SDMCA 資源拓撲
>
> →
> CPU／GPU／NPU／DPU 執行
> →
> 遙測與證據回饋.

SECTION

11.2 與 DEO

O-Chip 不直接設定任意電壓和頻率，而是提出：

```
requested_envelope = E2 / E3 / ...
duration
latency_goal
```

confidence
fallback
DEO 驗證平台、功率、熱與可靠度條件後接受、修改或拒絕。

SECTION

11.3 與 SynCore

O-Chip 決定何時適合：

- 神核模式；
- 流動模式；
- 混合模式；
- 普通多核心模式。

  SynCore 負責如何在硬體內實作該模式。O-Chip 不能只憑語義標籤宣稱多核心已物理融合。

SECTION

11.4 與 DPCPS

DPCPS 回報：

- 可用功率；
- 區域電流餘裕；
- 調節器健康度；
- 預期 droop 風險；
- 可承諾時間。

  O-Chip 依此安排任務與資料，而不是假定供電會自動追隨所有計畫。

SECTION

11.5 與 DryCore

DryCore 回報：

- 熱餘裕；
- 熱路徑狀態；
- 冷卻流量／風量；
- 露點安全；
- 熱虹吸或泵健康度；
- 模組間熱耦合。

O-Chip 可把高功率任務移到熱餘裕較大的模組，或降低品質／延後任務。

SECTION

11.6 與 SDMCA

SDMCA 提供：

- 模組與互連拓撲；
- NUMA／CXL／網路距離；
- 電源與冷卻故障域；
- 可維修狀態；
- 模組健康度。

O-Chip 的任務圖映射必須尊重真實物理拓撲，不能把所有資源視為等距。

SECTION

11.7 與拓撲計算引擎

後續拓撲計算引擎若被保留，可把 O-Chip 視為其工程化的計畫與控制層之一；但 TCE 的高維幾何、本體論與複雜度主張，不能反向作為 O-Chip 已可行的證據。O-Chip 必須先在現有硬體與任務圖上獨立驗證。

SECTION

12.1 威脅模型

O-Chip 增加新的控制面，因此必須假設：

- 惡意應用提交虛假期限或資源需求；
- 租戶嘗試透過遙測推斷他人工作；
- 模型被污染或對抗輸入欺騙；
- O-Card DMA 越權；
- 計畫造成拒絕服務或熱熱點；
- helper thread／runahead 擴大推測執行攻擊面；
- 韌體或策略更新遭竄改；
- 計畫與實際硬體狀態失同步。

SECTION

12.2 最小權限

每個 O-Bundle 使用 capability handle，限制：

- 可存取記憶體；
- 可使用裝置；
- 最大功率與時間；
- 可發出的提示；
- 可調整的排程域；
- 可讀取的遙測；
- 可採用的回退動作。

SECTION

12.3 遙測隱私

細粒度效能計數器與硬體回饋可能洩漏工作負載特徵。系統應採用：

- 租戶隔離；
- 最小化與聚合；
- 延遲或量化敏感遙測；
- 權限審核；
- 保留期限；
- 禁止默認建立個人行為模型。

  O-Chip 的預測應以任務與系統狀態為主，不以廣泛監控使用者生活習慣為前提。

SECTION

12.4 正確性與回退

所有提示均不得改變程式語義。對可能改變執行順序或資料位置的計畫，必須維持：

- 記憶體一致性；
- 同步語義；
- 例外與中斷可見性；
- 精度與品質承諾；
- 事務與冪等；
- 檢查點和重試。

SECTION

12.5 失效安全

O-Chip 失效時：

- 停止提交新計畫；
- 撤銷未承諾提示；
- 完成或取消可重試工作；
- 回到 OS 預設排程、硬體 HWP／CPPC 與裝置原生控制；
- 保存 trace 與錯誤證據。

  Sched_ext 的動態載入與錯誤時恢復預設排程，提供了軟體 MVP 可借鑑的失效模式。[3]

SECTION

13.1 MVP-A：O-Soft

平台

- Linux，啟用 sched_ext；
- perf_event_open／eBPF 遙測；
- cgroup v2 與 cpuset；
- resctrl 或 Arm MPAM 類資源控制；
- intel_pstate／amd-pstate 或通用 CPUFreq；
- 可選 GPU／NPU／DPU；
- 外部整機功率與溫度量測。

軟體組件

```
libochip-intent
      ↓
ochip-runtime
      ↓
phase detector / planner / certificate manager
```

```
    ↓
sched_ext + cgroup + resctrl + cpufreq + device adapters
    ↓
trace / metrics / replay
```

O-SDK 最小 API

```
struct ochip_intent {
  uint64_t deadline_ns;
  uint32_t quality_level;
  uint32_t flags;
  uint64_t energy_budget_uj;
  uint64_t data_bytes;
  uint32_t preferred_device_mask;
  uint32_t fallback_policy;
};
int ochip_begin_region(const struct ochip_intent *intent);
int ochip_submit_dependency(uint64_t parent, uint64_t child);
int ochip_end_region(void);
```

API 只是示意；實作應提供版本、權限和錯誤處理。

SECTION

13.2 測試工作負載

至少包含：

- 幀式互動管線：模擬遊戲或即時視覺，每幀有 CPU 邏輯、GPU、I/O 與期限；
- 編譯工作負載：前端、最佳化、連結與 I/O 階段明確；
- AI 推論服務：前處理、CPU、GPU／NPU、後處理與動態批次；
- 圖與 pointer-chasing：驗證預取與 helper thread 是否真正有效；
- 網路／儲存服務：驗證 O-Card 適用的粗粒度卸載；
- 不可預測混合負載：測試 O-Chip 是否知道何時回退。

SECTION

13.3 公平基線

比較：

- B0：平台預設排程與功率管理；
- B1：手工靜態 pinning／NUMA／裝置配置；
- B2：profile-guided 最佳化；
- B3：只有遙測的反應式 O-Soft；
- B4：明確應用提示 O-Soft；
- B5：提示＋預測＋跨層證書完整 O-Soft。

SECTION

13.4 指標

- 平均、p95、p99 和最大延遲；
- 吞吐量與期限違反率；
- 任務遷移與裝置切換次數；
- CPU、GPU、記憶體、I/O 與整機能耗；
- 資料搬移位元組；
- 快取失誤、記憶體頻寬與佇列等待；
- O-Chip 自身 CPU 時間、記憶體與推斷延遲；
- 預測校準與域外拒絕率；
- 回退次數、恢復時間與錯誤；
- 功率、溫度與熱節流。

SECTION

13.5 MVP-B：O-Card

在 FPGA／DPU 上實作：

- 命令佇列；
- DMA buffer 預置；
- 遙測壓縮；
- I/O 管線；
- 規則引擎；
- 安全 mailbox；
- 硬體時間戳。

比較 O-Card 與純軟體在不同任務粒度下的 break-even point。令工作粒度為 W，通訊與控制成本為 C_(card)，只有：

> W_(saved)
> >
> C_(card)

時才值得卸載。

SECTION

13.6 MVP-C：O-Die 模擬與 FPGA 原型

使用 RISC-V SoC、FPGA 多 tile 或 cycle-accurate 模擬器建立：

- CPU tile；
- 記憶體 tile；
- O-Die 控制 tile；
- command／telemetry NoC；
- 可選 DAE／helper engine；

- DEO 與 SynCore 模式模型。

先驗證延遲與協議，再考慮實體 chiplet。

SECTION

14.1 證據分級

等級	定義
E0	概念、形式模型與可否證設計
E1	軟體或模擬原型，可重現基線
E2	FPGA／DPU／硬體在迴路原型
E3	封裝內或專用矽原型，完整量測
E4	多工作負載、多平台、第三方重現
E5	產品化、長期可靠度、安全與資格化

本文目前為 E0。

SECTION

14.2 消融實驗

完整 O-Chip 必須分別移除：

- 應用提示；
- 預測器；
- 資料位置模型；
- DEO 整合；
- 熱／功率證書；
- 不確定性門控；

- 回退管理；

以辨識收益來源。若只開啟靜態 pinning 就得到相同結果，則不應把收益歸因於 O-Chip AI。

SECTION

14.3 可否證命題

H1：提示價值

對具有明確階段與期限的工作負載，顯式意圖可比僅依硬體計數器的黑箱預測更穩定地降低期限違反率。

否證條件：提示版本在多次測試中不優於反應式基線，或 API 成本抵銷收益。

H2：時間尺度分層

細粒度核心決策保留在核心內，粗粒度編排交給 O-Chip，將比把所有決策集中到軟體或外接卡更有效。

否證條件：O-Chip 的控制延遲在目標工作負載中普遍超過收益窗口。

H3：O-Card 粒度邊界

O-Card 對 I/O、網路、儲存與粗粒度命令圖可能有效，但對一般 CPU 分支與微指令無法產生正收益。

此命題可直接透過工作單元大小掃描驗證。

H4：不確定性門控

具有信心校準與域外回退的控制器，應比無條件預測控制器具有較低的 p99 回歸和安全違規率。

H5：跨層協同

在相同硬體上，O-Chip＋DEO＋資料位置＋熱／功率證書應比彼此獨立的局部控制器降低策略衝突與狀態震盪。

否證條件：完整系統沒有優於最佳單一控制器，或協同開銷過高。

SECTION

14.4 失敗也必須公開

最低公開資料包括：

- 原始 trace；
- 平台與韌體版本；
- 工作負載輸入；
- 基線設定；
- 計畫與證書；
- 模型版本；
- 回退和失敗記錄；
- 隨機種子；
- 統計方法；
- 能耗與溫度量測方法。

只公布最佳案例不足以證明架構。

SECTION

15.1 較適合的場景

- 異質 P-core／E-core 系統；
- CPU＋GPU＋NPU 推論管線；
- 幀圖、任務圖與 DAG 執行期；
- 編譯、渲染、科學工作流；
- 網路、儲存與 DPU 卸載；

- 多模組 SDMCA 系統；
- 需要功率、熱與期限共同控制的邊緣裝置；
- 具有顯式工作階段的資料中心服務。

SECTION

15.2 不適合或需高度限制的場景

- 單一、極短、不可預測的 CPU 分支；
- 無法取得應用語義且任務粒度極小；
- 通訊成本高於工作本身；
- 硬即時安全系統但尚未完成資格化；
- 密碼學常數時間與禁止推測執行的敏感程式；
- 資料無法跨裝置或安全域搬移；
- 平台不提供可靠遙測和控制接口；
- 需要完全透明且零整合成本的舊程式。

SECTION

15.3 主要限制

- 資訊不完整：O-Chip 無法從硬體計數器恢復完整程式意圖。
- 介面碎片化：不同 CPU、GPU、NPU 與 OS 的控制語義不一致。
- 控制開銷：跨層計畫、通訊與遷移可能抵銷收益。
- 模型轉移：工作負載和硬體更新會使預測失效。
- 安全面增加：更多遙測、DMA 和控制權需要更強隔離。

- 產業依賴：O-Die／O-Core 需要 CPU、封裝、OS 與 EDA 生態合作。
- 非普遍加速：某些工作負載已由成熟核心和 OS 接近最佳。

SECTION

階段 A：O-Soft E1

- 定義 O-Intent 與 O-Bundle schema；
- 建立 Linux sched_ext 原型；
- 接入 perf、cgroup、resctrl 與 CPUFreq；
- 完成五類工作負載與公平基線；
- 公開 trace、程式碼與失敗結果。

SECTION

階段 B：O-Card E2

- FPGA／DPU 遙測與命令佇列；
- 驗證任務粒度 break-even；
- 加入 IOMMU、安全 mailbox 與故障注入；
- 與 O-Soft 共同運作。

SECTION

階段 C：O-Die 模擬與封裝原型 E2–E3

- 建立 RISC-V／FPGA 多 tile；
- 測試 UCIe／CXL 類協議；
- 與 DEO、DPCPS、DryCore 和 SynCore 控制模型整合；

- 量測延遲、功耗、熱與回退。

SECTION

階段 D：特定 O-Core 共設計 E3+

- 只在已證明有效的工作負載加入 helper engine、DAE 或分支預解析；
- 進行安全分析與形式驗證；
- 不以移除通用核心控制邏輯作為預設目標。

O-Chip 原始概念抓住了一個真實問題：當運算系統包含多種核心、裝置、記憶體、供電與冷卻資源時，只依靠每一層在最後一刻局部反應，會失去應用意圖與全局協同。將規劃、選擇與執行區分，是值得保留的架構方向。

但工程上的解耦不是把「思考」整體搬出 CPU。核心內分支預測、亂序執行與快取控制之所以存在，是因為它們必須在週期級窗口內回應；外接 PCIe 卡無法透明攔截並重新排列一般 CPU 微指令，也不能任意覆寫私有 L1。真正可行的 O-Chip 必須尊重時間尺度、協議、權限和資料一致性。

因此，v2.0 將 O-Chip 定義為 Orchestration Chiplet：它接受應用與編譯器的意圖，融合硬體遙測，建立任務圖與資源圖，產生受約束計畫，再透過 OS、裝置佇列、DEO、SynCore、DPCPS、DryCore 與 SDMCA 執行。它可以使用規則、最佳化或機器學習，但模型沒有凌駕安全與回退的權力。

O-Chip 的成功標準不是具有「超靈」稱號，而是能否在公平基線下證明：

> 節省的等待、搬移、震盪與錯配
>
> 觀察、推斷、通訊、切換與回退成本.

若不能，O-Chip 應縮減為軟體執行期、DPU 控制面或特定硬體提示模組；若能，則它可能成為異質運算時代連接應用意圖與物理資源的共同編排層。

[1] Intel, “Intel Performance Hybrid Architecture & Software Optimization,” including Intel Thread Director runtime feedback and OS scheduling guidance.

https://cdrdv2-public.intel.com/685861/211115_Hybrid_WP_1_Introduction_v1.2.pdf

[2] Intel, “What Is Intel Thread Director?”

https://www.intel.com/content/www/us/en/support/articles/000097053/processors/intel-core-processors.html

[3] Linux Kernel Documentation, “Extensible Scheduler Class (sched_ext).”

https://docs.kernel.org/scheduler/sched-ext.html

[4] NVIDIA DOCA Documentation, “BlueField Hardware Architecture.”

https://docs.nvidia.com/doca/archive/3-2-2/Hardware-Architecture/index.html

[5] J. E. Smith, “Decoupled Access/Execute Computer Architectures,” ACM Transactions on Computer Systems, 1984.

https://dl.acm.org/doi/10.1145/1067649.801719

[6] O. Mutlu et al., “Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-Order Processors,” HPCA, 2003; and later efficiency studies.

https://dl.acm.org/doi/10.5555/822080.822823

[7] N. V. G. P. A. Jog et al., “Hardware/Software Cooperative Helper Threading,” 2016.

https://arxiv.org/abs/1602.01348

[8] UCIe Consortium, “UCIe Specifications,” including 3D packaging, manageability, test and debug support.

https://www.uciexpress.org/specifications

[9] Compute Express Link Consortium, “About CXL.”

https://computeexpresslink.org/about-cxl/

[10] Compute Express Link Consortium, “Introducing the CXL 3.X Specification,” coherent memory sharing and fabric management.

https://computeexpresslink.org/wp-content/uploads/2025/02/CXL_Q1-2025-Webinar-Presentation_FINAL.pdf

[11] M. Goudarzi et al., “By-Software Branch Prediction in Loops,” 2023.

https://arxiv.org/abs/2305.08317

[12] Linux Kernel Documentation, “User Interface for Resource Control Feature (resctrl).”

https://docs.kernel.org/filesystems/resctrl.html

[13] Linux Kernel Documentation, “Control Group v2.”

https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v2.html

[14] C. Shen et al., “SPECRUN: The Danger of Speculative Runahead Execution,” 2024, illustrating security risks introduced by speculative helper mechanisms.

https://dl.acm.org/doi/10.1145/3649329.3655932