UACC: A Chiplet-Based Auxiliary Cache with Near-Cache Operations for Cost-Effective Universal Scaling

Zichao Ling
Beijing University of Posts and Telecommunications
Beijing, China
lingzichao@bupt.edu.cn

Chen Bai
Fudan University
Shanghai, China
cbai@fudan.edu.cn

Jianwang Zhai*
Beijing University of Posts and Telecommunications
Beijing, China
zhaijw@bupt.edu.cn

Kang Zhao
Beijing University of Posts and Telecommunications
Beijing, China
zhaokang@bupt.edu.cn

Abstract

Static Random Access Memory (SRAM) scaling in advanced technology nodes is approaching stagnation, and its rising cost severely limits on-chip cache capacity in modern multicore processors. To alleviate this challenge, this work proposes the Universal Auxiliary Cache Chiplet (UACC), which leverages heterogeneous technology nodes and a partially decoupled cache hierarchy, providing additional high-speed memory capacity at a cost equivalent to an advanced node monolithic design. To overcome the performance degradation from die-to-die latency and bandwidth (the "Chiplet Tax"), we introduce a contextualization strategy that allocates UACC capacity only where it yields net performance benefit. Our congestion-aware 2D lookahead algorithm systematically evaluates real-time access characteristics, die-to-die contention, and workload-level utility to determine both the timing and magnitude of auxiliary capacity allocation. Experimental results show UACC improves performance by 8.92% on balanced workloads and up to 27.36% on cache-sensitive ones at same costs, without degrading baseline performance.

1 Introduction

The pervasive latency challenge in general-purpose computing is primarily attributed to the growing disparity between memory access times and compute speeds—a phenomenon known as the "memory wall." While transistor density continues to scale with successive process nodes, the area density of caches, crucial for latency hiding, has largely stagnated. Measurements [7, 8, 23] indicate that the SRAM bitcell area in 3nm technology nodes has scaled by less than 5% compared to the previous generation, with certain variants even exhibiting a density regression. As a result, the economic cost of expanding die area solely to increase cache capacity in advanced processes has become prohibitively high.

Chiplet, an advanced packaging approach that achieves heterogeneous process integration through modular assembly, offers a promising solution to this challenge. Based on our modeling and evaluation of prior work [11] (detailed in Section 2), we observe that under the same N3 process cost budget, heterogeneous integration of N5 or N7 nodes can provide an additional 86%-147% high-speed SRAM capacity and transforms the otherwise exponential cost growth curve into an approximately linear trend.

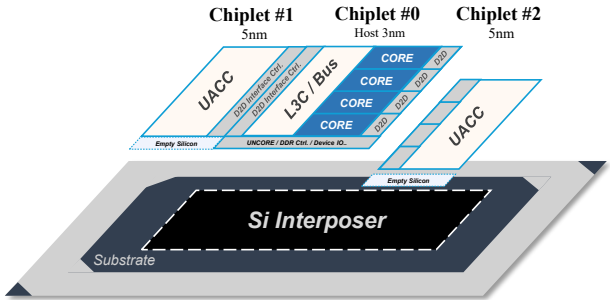


Figure 1: A 2.5D Chiplet System integrated with UACC.

However, translating this capacity advantage into performance gains is challenging. When SRAM is disaggregated into a separate Chiplet, as explored in several prior studies [12, 16, 19] through designs that partition the Last Level Cache (LLC) and cores into a memory die and a logic die, the additional data-transfer latency introduced by die-to-die (D2D) communication, together with non-uniform effects from VLSI physical layout, increases the critical path of the memory hierarchy and elevates the average memory access time (AMAT). This degradation of baseline performance means that even conservative workloads, which would otherwise operate comfortably within a monolithic cache capacity, can still suffer noticeable performance loss. This penalty, commonly referred to as the **Chiplet Tax** [10, 11, 17, 21], poses a fundamental challenge to the practical use of chiplet-based cache extension.

To mitigate this, we adopt a *contextualization* strategy and propose the **Universal Auxiliary Cache Chiplet (UACC)**[1]. For a multi-core host, the system is not strictly divided into a logic die and a cache die; instead, as illustrated in Figure 1, the logic portion retains a portion of the cache on chip to preserve baseline performance. Through coordinated hardware-software adaptive scheduling and control, *only* workloads that can benefit from the additional capacity, namely those with large cache demands, are allowed to offload to the UACC, while all other workloads remain on the original critical path to maintain low latency. This selective offloading mechanism defines the role of the UACC as *auxiliary*, since it supplements rather than replaces the on-chip cache and

*Corresponding author.

[1]The source code and experiment artifacts are publicly available in: https://anonymous.4open.science/r/gem5-UACC-7780.

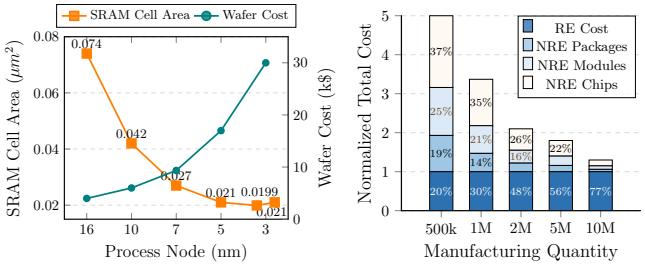


Figure 2: (a) SRAM cell area and wafer cost vs. process node; (b) RE/NRE cost breakdown vs. manufacturing quantity.

Table 1: Cost for N3 SRAM budgets and equivalent capacities.

N3 Budget	Area (mm^2)	Die Cost	RE Total	Eqv. N5 Cap.	Eqv. N7 Cap.	N5-N7 Gain
1 MB	0.316	0.266	0.315	2.25MB	3.25MB	2.25×–3.25×
2 MB	0.532	0.396	0.471	4MB	5.25MB	2.00×–2.63×
4 MB	0.964	0.643	0.768	7MB	9.25MB	1.75×–2.31×
8 MB	1.828	1.115	1.339	13MB	17MB	1.63×–2.13×
16 MB	3.556	2.031	2.449	26.5MB	32.5MB	1.66×–2.03×

is invoked only when beneficial. By functioning in this supportive capacity, the UACC enables scalable cache expansion while preserving baseline performance.

Realizing this potential requires an allocation mechanism that can accurately identify capacity-sensitive workloads and direct them to the auxiliary cache when appropriate. Some prior studies in cache partitioning have explored dynamic allocation of a monolithic shared LLC capacity: UCP [20] allocates cache based on utility and provides practical hardware support for sampling workload capacity demand, but it lacks awareness of non-uniform cache latency; CloudCache [15] enables distance-aware reallocation of private cache capacity in a 2D mesh topology, yet its Manhattan-distance partition cannot capture congestion-induced non-uniformity stemming from limited D2D bandwidth. To address these limitations, we propose a **congestion-aware 2D lookahead** algorithm that provides effective UACC scheduling and allocation strategies for both multicore systems and configurable D2D interconnect scales.

To summarize, our main contributions are as follows:

- We propose the UACC architecture that leverages Chiplet-based heterogeneous integration to achieve cost-effective on-chip cache capacity scaling.
- We propose a hardware-software cooperative mechanism, including a congestion-aware 2D lookahead scheduling algorithm, to efficiently enable elastic cache expansion.
- We analyze Chiplet-induced non-uniformity using simulator evaluation to identify the workload and system conditions under which UACC delivers the greatest benefit.

Experimental evaluations demonstrate that UACC boosts performance by 8.92% on balanced workloads and up to 27.36% on highly cache-sensitive workloads without hurting baseline performance or incurring congestion penalties, and at essentially no additional system cost.

2 Background and Motivation

From a manufacturing cost perspective, decomposing a monolithic SoC into smaller dies significantly improves yield and reduces the effective cost per function. For a monolithic die with area A and defect density D_0, the yield Y follows the defect-density model [11]:

$$Y \approx e^{-D_0 A}. \tag{1}$$

The recurring cost (RE) of the monolithic design can thus be expressed as:

$$C_{\text{RE}}^{\text{mono}} \propto \frac{A \cdot C_{\text{wafer}}}{Y}, \tag{2}$$

where C_{wafer} is the wafer cost per unit area. Since SRAM often occupies about 50% of the total multi-core processor area [19], integrating large SRAM arrays on advanced-node logic dies amplifies both wafer cost and yield loss. Partitioning the design into a logic die and a cache chiplet decreases each die's area, thereby improving yield and reducing the aggregate RE cost term $\sum_i \frac{A_i C_{\text{wafer},i}}{Y_i}$.

Further cost reduction is achieved through heterogeneous integration. The logic chiplet benefits from advanced nodes for performance, while the cache chiplet can be fabricated using a mature process node with lower wafer and mask costs. The total RE cost for the heterogeneous system is then

$$C_{\text{RE}}^{\text{chip}} = \frac{A_{\text{logic}} C_{\text{wafer}}^{p_{\text{logic}}}}{Y_{\text{logic}}^{p_{\text{logic}}}} + \frac{A_{\text{cache}} C_{\text{wafer}}^{p_{\text{cache}}}}{Y_{\text{cache}}^{p_{\text{cache}}}}, \tag{3}$$

where p_{logic} and p_{cache} denote the respective process nodes. Because SRAM scaling offers diminishing density improvement in advanced technologies, this heterogeneous partition minimizes RE cost without compromising functionality.

On the non-recurring engineering (NRE) side, the cache chiplet exhibits strong design reuse across product lines. If its design and verification cost $C_{\text{NRE}}^{\text{cache}}$ is amortized over R products, the average contribution per design becomes $C_{\text{NRE}}^{\text{cache}}/R$. The total system cost can thus be written as

$$C_{\text{chip}} = \sum_i \frac{A_i C_{\text{wafer},i}}{Y_i} + C_{\text{NRE}}^{\text{logic}} + \frac{C_{\text{NRE}}^{\text{cache}}}{R} < C_{\text{mono}}. \tag{4}$$

Figure 2(b) highlights this trend: as manufacturing volume increases, RE cost dominates while the amortized NRE cost fraction decreases substantially. This reinforces the economic advantage of reusable chiplets, especially for SRAM-heavy processors.

When the system-level bandwidth and latency constraints are satisfied, Equation (4) indicates that the chiplet partition is economically favorable whenever the yield and process-node savings outweigh the integration overhead. Therefore, isolating the cache as a reusable chiplet becomes a cost-effective architectural choice for multicore processors with high SRAM ratios. As shown in Table 1, a chiplet built in an N5 process offers on average 1.86× more SRAM capacity at equal silicon cost, and an N7 chiplet provides 2.47× more capacity. In practical designs, these gains can be conservatively approximated as 1.5× and 2× effective capacity scaling, respectively.

3 UACC Architecture

As depicted in Figure 3, the UACC functions as an on-demand auxiliary layer that is activated through hardware sampling, dynamic

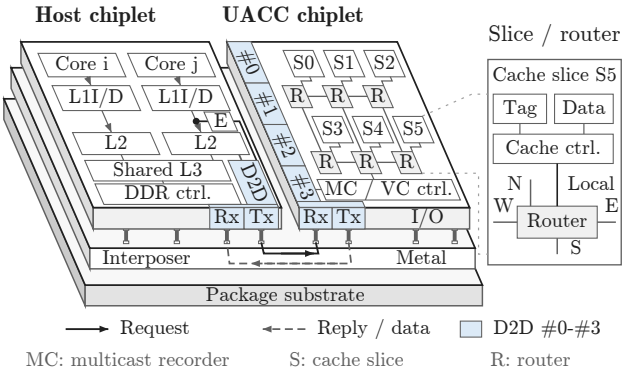


Figure 3: UACC architecture including a 2D-mesh cache chiplet to extend the host's private L2.

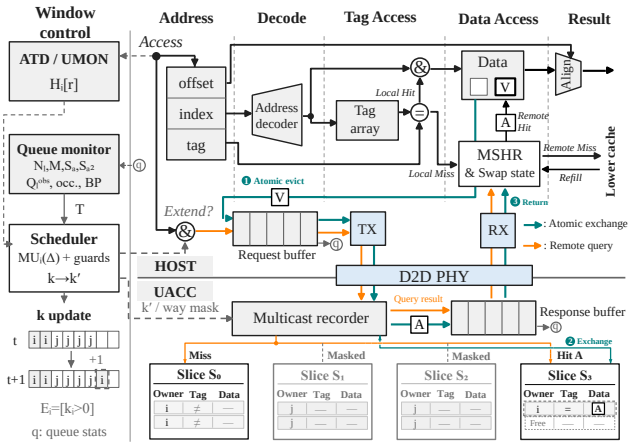


Figure 4: Pipelined way extension with parallel UACC tag lookups and an atomic exchange flow.

scheduling, and adaptive allocation mechanisms. This design preserves the latency-critical cache hierarchy of the baseline processor while enabling elastic capacity scaling through chiplet integration.

Notably, due to D2D bandwidth limits and transmission latency, cache ways expanded in UACC exhibit Non-Uniform Cache Access (NUCA) between local and remote chiplet. In fact, Chiplet-based cache hierarchies follow two main directions: Uniform Cache Access (UCA) versus NUCA, corresponding to the two UACC integration approaches in Figure 1: extending the shared cache layer (L3/LLC, #1, UCA) or the private cache layer (L2, #2, NUCA). The UCA-style static capacity expansion is more practical from an engineering standpoint, such as 3D-stack, requiring minimal microarchitectural modification and enabling a more deterministic physical layout, thus widely adopted. NUCA may appear as a naive alternative. However, as Chiplet integration enables rapid cache capacity growth, compounded long-path latency and unpruned traffic quickly saturate D2D bandwidth, causing inter-die congestion and pushing the cache system toward its hierarchical latency bound. Under such scaling, NUCA is no longer optional but necessary. Therefore, this work focuses on the private-layer extension (#2), a NUCA-based approach, to relax this bound.

Pipelined Integration. As illustrated in Figure 4, the extended pipeline enables parallel HOST and UACC tag lookups, with Miss Status Handling Register support for out-of-order transaction; upon an remote hit and local miss, a **swap operation** atomically exchanges the accessed line with a local victim (e.g., LRU), thereby promoting frequently accessed data into the local cache and maintaining its contents in a near-optimal LRU order.

Multicast Handling. To minimize host-side modifications and suppress unnecessary cross-die traffic, the UACC implements a **multicast recorder** that performs parallel tag probes across its internal slices. When the host issues a remote lookup, the recorder aggregates slice responses locally within the UACC, ensuring that only the final hit/miss outcome is returned over the D2D link. This design maintains the existing on-host organization, preventing a microarchitectural redesign and additional D2D communication.

Coherence Compatibility. Because the UACC functions solely as a private-cache way extension rather than an independent coherence participant, the system's coherence protocol remains unchanged. Remote tag access follows the same pipelined lookup structure described earlier, simply adding a remote way check without introducing new coherence states or messages. As a result, coherence behavior stays fully transparent while the UACC provides elastic capacity augmentation.

4 Elastic Expansion Scheduling

4.1 Cache Sensitivity

To determine when elastic cache expansion is beneficial, we first characterize the capacity sensitivity of representative SPEC CPU2017 workloads. For each workload, we compute

$$\text{Speedup} = \frac{\text{IPC}(C_i)}{\text{IPC}(256\ \text{KB})}, \tag{5}$$

where C_i denotes an enlarged private-cache configuration. All metrics include only demand misses and exclude prefetch-generated accesses. As shown in Figure 5, capacity-sensitive workloads achieve speedups well above 1.0 because their working sets exceed the baseline cache capacity. Capacity-insensitive workloads remain close to 1.0 because their working sets already fit, or because execution is compute- or stream-bound. This separation provides an online allocator with a direct measure of whether additional cache ways can improve performance.

The same figure also shows that capacity sensitivity and memory-access intensity are distinct properties. A workload may benefit from additional ways yet inject enough remote traffic to congest a D2D interface. At light load, remote packets experience mostly propagation, routing, and serialization latency. At higher load, packets wait behind occupied transmit ports and buffered packets, causing queueing delay to rise sharply as utilization approaches one. Moreover, cache misses are not generally Poisson: MSHR releases, writeback bursts, and phase changes can produce highly variable inter-arrival times. UACC therefore models each independently contended request or response direction as a queueing domain and measures its arrival and service-time moments. This avoids both the Poisson assumption of the original M/G/1 formulation and the incorrect aggregation of traffic that does not share a server.

These two workload properties determine whether UACC should extend a private cache. Treating the local cache and its UACC ways as one logical cache, we write

$$\text{AMAT} = \text{HitTime} + \text{MissRate} \times \text{MissCost}. \tag{6}$$

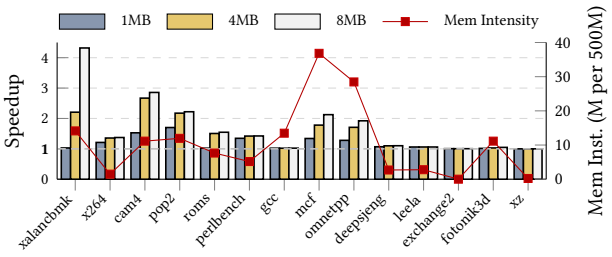


Figure 5: Cache capacity sensitivity and memory intensity of SPEC CPU2017 workloads in the gem5 evaluation.

Enabling UACC changes AMAT by

$$\begin{aligned}\Delta\text{AMAT} &= \text{AMAT}_{\text{Enable}} - \text{AMAT}_{\text{Disable}} \\ &= \Delta\text{HitTime} + \Delta\text{MissRate} \times \text{MissCost},\end{aligned} \tag{7}$$

where $\Delta\text{HitTime} > 0$ captures the remote traversal and contention penalty, whereas $\Delta\text{MissRate} < 0$ captures the capacity benefit. UACC is useful only when the avoided lower-level accesses outweigh the additional remote access cost, i.e., when $\Delta\text{AMAT} < 0$.

4.2 Congestion-Aware Scheduling

Each core i has a fixed local-cache depth Pos_{base}. The allocation vector $\mathbf{k} = (k_1, \ldots, k_N)$ specifies the number of remote ways assigned to each core. UACC profiles capacity utility using an auxiliary tag directory (ATD) and utility monitor, following UCP [20]. For each L1 miss, the ATD records the LRU recency position r and increments $H_i[r]$. Over a profiling window of T cycles, the residual miss fraction with k remote ways is

$$\text{miss}_k(i) = \frac{N_i^{L1\text{miss}} - \sum_{r=0}^{\text{Pos}_{\text{base}}+k-1} H_i[r]}{N_i^{L1\text{miss}}}, \tag{8}$$

and the remote-hit probability is

$$p_i(k) = \frac{\sum_{r=\text{Pos}_{\text{base}}}^{\text{Pos}_{\text{base}}+k-1} H_i[r]}{N_i^{L1\text{miss}}}. \tag{9}$$

Thus, a lookahead candidate $k_i \to k_i + \Delta$ predicts

$$\Delta H_i(\Delta) = \sum_{r=\text{Pos}_{\text{base}}+k_i}^{\text{Pos}_{\text{base}}+k_i+\Delta-1} H_i[r] \tag{10}$$

additional remote hits. If each hit avoids $L_{\text{lower},i}$ cycles of lower-level access latency, its capacity gain is

$$G_i(\Delta) = \Delta H_i(\Delta) L_{\text{lower},i}, \tag{11}$$

in stall cycles per profiling window.

G/G/1 Congestion Model. UACC models each independently contended request or response direction as a queueing domain ℓ. Packet latency is $L_{\text{remote}} = L_{\text{fixed}} + S + W_q$, where S is link occupancy time and W_q is queue wait; propagation RTT is not service time. For the N_ℓ packets observed in a T-cycle window, we define

$$\begin{aligned}\lambda_\ell &= \frac{N_\ell}{T}, & \rho_\ell &= \lambda_\ell E[S_\ell], \\ C_{A,\ell}^2 &= \frac{\text{Var}(A_\ell)}{E[A_\ell]^2}, & C_{S,\ell}^2 &= \frac{E[S_\ell^2] - E[S_\ell]^2}{E[S_\ell]^2}.\end{aligned} \tag{12}$$

Kingman's G/G/1 moment approximation [24] estimates the mean queue wait as

$$W_{q,\ell}^{\text{GG1}} \approx \frac{\rho_\ell}{1-\rho_\ell} \frac{C_{A,\ell}^2 + C_{S,\ell}^2}{2} E[S_\ell]. \tag{13}$$

This form captures first-order arrival and service variability. At $C_A^2 = 1$, it reduces to the M/G/1 Pollaczek–Khinchine mean. UACC uses it only for candidate ranking and saturation avoidance.

Window-Count Formulation. Evaluating Equation (13) per candidate requires explicit rates, moments, and multiple divisions. The implementation instead uses integer window counts for packet classes j (requests, hit/miss responses, and writebacks) with configured service times $S_{\ell,j}$. The ATD predicts $n_{\ell,j}(\mathbf{k})$; enabling the first remote way adds all lookup requests, while later ways mainly change the response mix.

For arrival burstiness, the monitor retains M valid inter-arrival samples, $S_A = \sum_n A_n$, and $S_{A2} = \sum_n A_n^2$, and computes

$$C_A^2 = \max\left(0, \frac{M S_{A2}}{S_A^2} - 1\right), \qquad C_{A,\text{eff}}^2 = \max(1, \overline{C_A^2}). \tag{14}$$

The overline is a shift-based EWMA. Undersampled windows retain the previous estimate, and clamping at one prevents underprediction.

For candidate $\mathbf{k}$, the engineering datapath accumulates

$$\begin{aligned}R_{0,\ell}(\mathbf{k}) &= \sum_j n_{\ell,j}(\mathbf{k}), \\ R_{1,\ell}(\mathbf{k}) &= \sum_j n_{\ell,j}(\mathbf{k}) S_{\ell,j}, \\ R_{2,\ell}(\mathbf{k}) &= \sum_j n_{\ell,j}(\mathbf{k}) S_{\ell,j}^2.\end{aligned} \tag{15}$$

Here R_0 is the packet count, R_1 the predicted busy cycles, and R_2 the service-time second-moment sum. Substituting $\lambda = R_0/T$, $E[S] = R_1/R_0$, and $E[S^2] = R_2/R_0$ into Equation (13), then multiplying the mean wait by R_0, gives the total queueing cost directly:

$$Q_\ell^{\text{model}}(\mathbf{k}) = \frac{R_{1,\ell}^2(\mathbf{k})(C_{A,\text{eff},\ell}^2 - 1) + R_{0,\ell}(\mathbf{k}) R_{2,\ell}(\mathbf{k})}{2\left(T - R_{1,\ell}(\mathbf{k})\right)}. \tag{16}$$

The result is queue-stall cycles per window and requires only count accumulation, products, and one reciprocal/division; λ, $E[S]$, C_S^2, and W_q are never explicitly constructed. The division-free utilization guard rejects $R_{1,\ell} \geq \rho_{\max} T$.

To cover correlation, finite buffers, arbitration, and backpressure omitted by the moment model, UACC uses completed-window measurements. Let $Q_\ell^{\text{obs}} = \sum_n W_{q,n}^{\text{obs}}$ and define

$$\begin{aligned}\beta_\ell^{\text{raw}} &= \text{clip}\left(\frac{Q_\ell^{\text{obs}}}{\max(Q_\ell^{\text{model,current}}, \epsilon)}, 1, \beta_{\max}\right), \\ \widehat{Q}_\ell &= \overline{\beta}_\ell Q_\ell^{\text{model}}.\end{aligned} \tag{17}$$

The feedback factor is EWMA-smoothed and computed before the next allocation, preserving causality. Persistent occupancy, buffer-full, or backpressure events serve as additional rejection guards.

Marginal Utility and Allocation. The non-queueing communication cost of allocation $\mathbf{k}$ is

$$F(\mathbf{k}) = \sum_{\ell,j} n_{\ell,j}(\mathbf{k}) L^0_{\ell,j}. \tag{18}$$

For candidate $\mathbf{k}'$, the marginal utility is

$$\mathrm{MU}_i(\Delta) = G_i(\Delta) - \left[F(\mathbf{k}') - F(\mathbf{k})\right] - \sum_{\ell} \left[\widehat{Q}_\ell(\mathbf{k}') - \widehat{Q}_\ell(\mathbf{k})\right]. \tag{19}$$

All terms are stall cycles per window. The total queue-cost difference includes delay imposed on existing traffic. Multi-way candidates are ranked by $\mathrm{Score}_i(\Delta) = \mathrm{MU}_i(\Delta)/\Delta$, but the controller commits at most one way per window to avoid unstable allocation changes.

Only affected domains are recomputed. A candidate must have positive utility, fit the way budget, and pass all congestion guards. Under persistent congestion, the controller revokes the lowest-gain way and waits for a congestion-free recovery interval before expanding.

4.3 Controller Implementation

The controller decouples packet collection from scheduling: each queueing domain maintains arrival moments, packet-class counts, wait, occupancy, and backpressure with saturating snapshot/reset updates, and the allocator consumes these counters off the cache and D2D critical path. At each boundary, every ATD candidate is reduced to R_0, R_1, and R_2 using Equation (15); utilization guards precede queue-cost evaluation in Equation (16), followed by the fixed-cost, gain, marginal-utility, and score calculations in Equations (11), (18) and (19).

For area efficiency, one shared direct-cost pipeline serializes domain and candidate evaluation; single-port LUTRAM stores per-candidate queue costs, and a shared arithmetic tail handles marginal utility, score scaling, and selection. The C_A^2 and measured-wait β updates Equations (14) and (17) share one rolled restoring divider, while EWMA and $\{1, 2, 4\}$ lookahead scaling use add/shift datapaths. The area-oriented W100K point uses UQ4.4 for C_A^2, β, and costs, Q1.8 utilization, 17-bit R_0/window counters, 22-bit R_1, 27-bit R_2, 35-bit per-domain cost, 38-bit total cost, 35-bit signed score, and a 51-bit integer divider datapath. Saturating arithmetic plus utilization, occupancy, pressure, and overflow guards reject unsafe candidates before fixed-point overflow can affect allocation.

5 Evaluation

In this section, we evaluate UACC with two goals: (a) to show that chiplet-based capacity extension can improve performance under a cost-equivalent cache budget, and (b) to demonstrate that our congestion-aware allocation algorithm is effective under diverse workload pressures and D2D interconnect tiers. We summarize the benchmark suite, simulation settings, and baseline designs below.

5.1 Experiment Setup

Benchmarks. We construct a multi-program workload suite from the memory access trace of SPEC CPU2017 [5]. We classify workloads by cache-capacity sensitivity into three groups: **high** { `xlanCBMK`, `cam4`, `pop2` }, **medium** { `x264`, `roms`, `perlbench` }, and **low** { `all others` }. Then we form three representative workload classes: Low-Pressure (LP) with 0% high-, 30% medium-, and 70% low-sensitivity programs; Balanced-Pressure (BP) with 33% high-, 33% medium-, and 34% low-sensitivity programs; and High-Pressure (HP) with 90% high-, 9% medium-, and 1% low-sensitivity programs. Each class is evaluated by randomly assigning a workload to each core (with replacement) and repeating this process 5 times to report the average.

Table 2: Evaluated D2D Interfaces for UACC Integration

Interface	RTT (ns)	Bandwidth (GB/s)	Buffer Dep. (flits)	Norm. Area (mm²/Tb/s)	Cost ($)
AIB	10–15	10–120	8–16	0.6–0.8	**Low**
BoW	8–12	50–250	16–32	≈0.8	**Low–Med**
EMIB*	6–10	100–300	24–32	≈0.8	**Medium**
UCIe	4–8	128–512	24–64	≈0.31	**High**

Configuration. We implement UACC in gem5 [4] with four 4-GHz O3 cores (352-entry ROB; 6-wide fetch/decode/dispatch, 4-wide issue, and 5-wide retire) and 64-B cache lines. Each core has 32-KiB/8-way L1I, 48-KiB/12-way L1D, and private 256-KiB/16-way L2 caches with 4/5/10-cycle hit latencies. Private-cache misses traverse per-core selectors and serialized D2D request and response links before reaching the shared remote cache; the implementation records packet enqueue time, service time, queue occupancy, and backpressure. For each D2D tier, fixed propagation and routing latency are modeled separately from packet serialization and queueing delay [13, 22], while request and response directions retain independent queue statistics.

Starting from an N3 monolithic LLC budget of 4 MB, 8 MB, or 16 MB, we initialize the UACC chiplet with zero capacity and progressively shift cache ways from the host die to the chiplet. In each step, we subtract 256 KB (4 slices × 64 KB) from the on-HOST LLC and add an equivalent-cost capacity to on-UACC. Cost equivalence is maintained using per-MB SRAM area and per-mm² process-technology (N5 or N7) cost models [11], ensuring that all compared configurations correspond to identical N3-normalized budgets. For the remaining PPA evaluations, we adopt the design-space exploration framework presented in [18].

We further sweep four D2D interface tiers by varying link bandwidth and round-trip latency while keeping the routing topology and buffer depth fixed, as summarized in Table 2. Here, EMIB* denotes a modeled D2D link over an embedded silicon bridge, with the listed parameters serving as modeling assumptions. To create additional traffic sources and intensify congestion pressure, we also evaluate larger multicore configurations with 16, 32, and 64 cores together with 2 MB budget per core.

Baselines. We compare UACC against four cache allocation strategies: (1) Monolithic-Static employs a monolithic N3 LLC without way extension, representing the cost-equivalent baseline. (2) UCP [20]-Greedy implements utility-based capacity allocation using marginal miss reduction, but without distance-awareness, and all capacity is exposed through a unified address space, isolating the benefit of dynamic capacity management. (3) CloudCache [15] adopts dynamic global partitioning combined with distance-aware

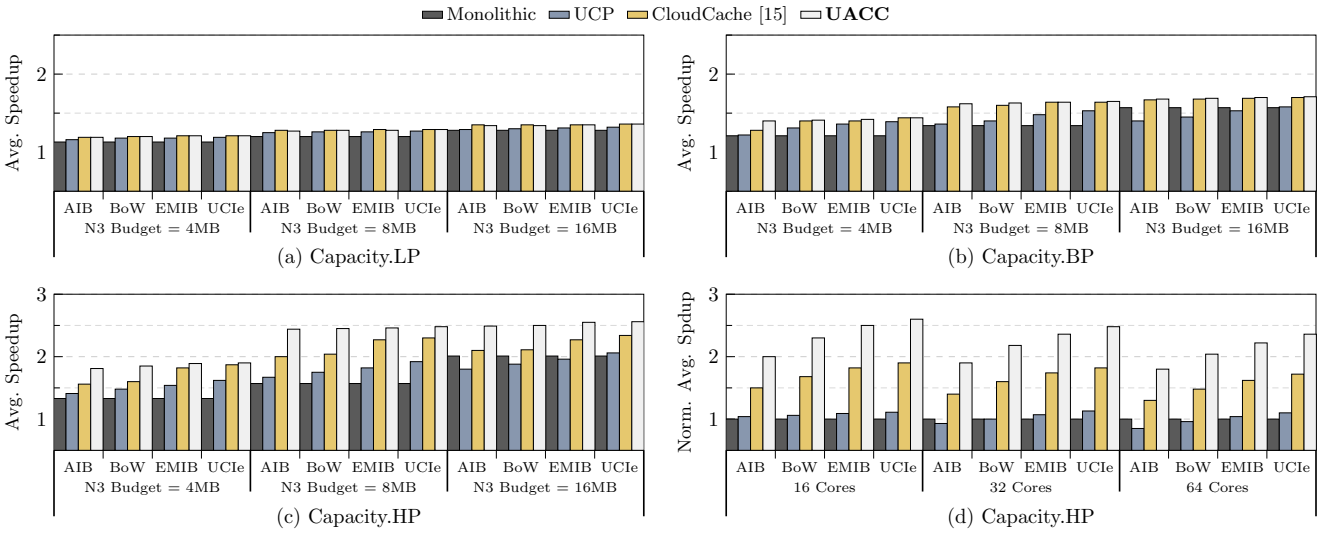


Figure 6: Overall performance comparison between UACC and baseline cache designs under different cache sizes.

allocation, but remains agnostic to network congestion. (4) UACC (our design) combines distance-aware allocation with the window-count G/G/1 queue cost and measured congestion feedback described in Equations (16) and (17).

5.2 Results and Analysis

Figure 6 compares the four schemes across capacity budgets, D2D tiers, cache-pressure mixes, and core counts. UACC is competitive with the distance-aware baseline when traffic is light and separates from it as remote contention grows.

Capacity pressure. Under *Capacity.LP*, UACC remains within 1–2% of CloudCache and introduces no meaningful regression. Its benefit becomes clear for capacity-sensitive workloads: at 8 MB under *Capacity.HP*, UACC reaches up to 2.46× speedup, compared with 1.82× for UCP and 2.27× for CloudCache. Gains taper beyond 8 MB as more working sets fit, confirming that remote capacity should be expanded only while its marginal utility remains high.

Interconnect contention. Better D2D tiers improve all schemes, whereas congestion awareness determines whether the added bandwidth is used effectively. At 8 MB under *Capacity.HP*, UACC sustains 2.44–2.48× across the four tiers, while CloudCache reaches 2.00–2.30× and UCP 1.67–1.92×. The core-scaling experiment exposes the same effect: at 64 cores with AIB, UACC retains 1.80× normalized speedup, compared with 1.30× for CloudCache and 0.85× for UCP. Thus, distance awareness is sufficient at low pressure, but explicit queue-cost control is required to avoid harmful remote allocations once traffic becomes material.

Queue-model ablation. Figure 7 isolates the allocator on a synthetic 4-core HP trace with identical arrivals, capacity utilities, and candidate allocations for all policies. During windows 8–31, unmodeled service contention reduces the actual FIFO bandwidth to 40% of the predictor's nominal value; the measured-delay oracle is an offline upper bound. At $\rho_{\text{ref}} = 0.426$, G/G/1 raises utility/oracle from 0.352 to 0.642 and reduces backpressure from 288.2 to 208.1 events/1K packets. Feedback further reaches 0.875 and 96.5, respectively, by correcting persistent model underestimation from

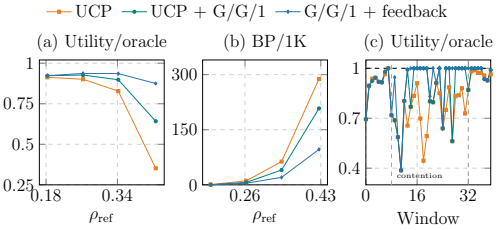


Figure 7: Queue-model ablation comparing UCP, G/G/1 queue costing, and feedback: (a) utility/oracle agreement versus reference utilization, (b) backpressure events per 1K packets, and (c) utility/oracle over time under hidden contention.

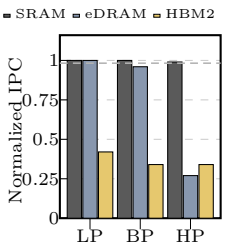
Table 3: Range-width profiles and projected controller PPA.

Profile	$T_{\max}$	R_1/R_2 bits	Agree.	Sat.	Area (10^{-3} mm^2)	Fmax (MHz)	Pwr. (mW)
Lossy16	100K	16 / 20	74.17%	81.1K	8.46	456.8	19.7
W100K	**100K**	**22 / 27**	**100%**	**0**	**9.34**	**456.8**	**16.2**
W1M	1M	25 / 30	100%	0	10.09	456.8	17.4
Legacy	1M	56 / 80	100%	0	22.60	416.7	159.1

completed windows. Thus, the online feedback improves allocation under model mismatch, whereas both queue-aware policies converge with UCP when the FIFO is fully saturated.

5.3 Hardware Overhead Evaluation

Evaluation Flow. We implement the controller as a fixed-point HLS design and use Vitis HLS to generate Verilog [2]. Each point first passes C/RTL co-simulation; the generated RTL is then synthesized with Yosys [25] and evaluated in a common ASAP7 flow [1, 9] at 0.77 V and 5 ns, including placement, CTS, and global routing. All width-sweep points retain the same candidate set, packet classes, UQ4.4 formats for C_A^2, β, and costs, Q1.8 utilization, and physical constraints; only integer counter ranges are swept.



Storage models at 8 MiB UACC

Metric	SRAM	eDRAM	HBM2
Lat. / ns	1.931	1.693	72.624
Cycle / ns	1.295	2.772	-
BW / GB/s	49.42	23.09	178.91
R E/64B / nJ	1.747	1.048	1.997*
W E/64B / nJ	1.799	13.408	-
Leakage / W	0.268	0.107	-
Power@10 / W	0.542	0.464	0.312*
8 MiB util.	100%	100%	0.195%

Figure 8: Left: pressure-performance behavior across storage media. Right: storage-model parameters. Power@10 is evaluated at 10 GB/s; * denotes the modeled 4 GiB HBM2 stack.

Range-Width Results. We sweep only the integer dynamic range: the deliberately unsafe Lossy16 point, the 100K-cycle minimum-range design (W100K), a 1M-cycle design (W1M), and the legacy-wide prototype. The test uses 10,000 deterministic stress windows with on/off and Poisson traffic. Table 3 jointly reports decision agreement, saturation, and projected-controller PPA.

W100K, W1M, and Legacy make identical decisions and incur no saturation. W100K reduces projected controller area by 58.65% relative to Legacy while retaining 70.1% of the 100K-cycle profiling window after its worst-case HLS computation. Lossy16 saves only 9.46% controller area because the collector is unchanged, but agrees with W100K in just 74.17% of windows. We therefore select W100K as the minimum safe range. Power uses default switching activity and is reported only as a comparative estimate [6].

5.4 Memory Technology Comparison

We compare SRAM with eDRAM [3] and HBM [14] to determine whether UACC should use a mainstream chiplet-memory alternative rather than SRAM, as summarized in Figure 8. eDRAM is attractive at low to moderate bandwidth because its denser cells and lower leakage provide a favorable capacity-power tradeoff while preserving performance close to SRAM; under sustained pressure, however, its longer cycle time and limited effective interface bandwidth increase backpressure and erase this advantage. The modeled 4 GiB HBM2 stack is unsuitable for the 8 MiB UACC target: its capacity far exceeds the required extension, while its latency, cost, and I/O requirements outweigh the benefit of surplus bandwidth. SRAM therefore remains the practical default for predictable low-latency operation at cache-scale capacity, with eDRAM useful only as an energy-oriented option for low-to-medium-bandwidth deployments.

6 Conclusion

In this work, we present UACC, a chiplet-based auxiliary cache that provides cost-effective capacity scaling while preserving the latency-critical private-cache path. Its congestion-aware lookahead allocator combines per-core utility with D2D queue-cost estimation and feedback, extending remote capacity only when capacity gains outweigh interconnect delay. Across cache budgets, D2D tiers, and workload mixes, UACC outperforms monolithic, utility- or distance-based baselines, especially under high cache pressure and contention.

References

[1] Tutu Ajayi, Vidya A. Chhabria, Mateus Fogaça, et al. 2019. Toward an open-source digital flow. In *Proc. DAC*. 1–4.

[2] AMD. 2024. Vitis High-Level Synthesis User Guide. AMD Documentation. https://docs.amd.com/r/en-US/ug1399-vitis-hls

[3] Darren Anand, Kevin W. Gorman, Mark D. Jacunski, et al. 2011. Embedded DRAM in 45-nm technology and beyond. *IEEE Design & Test of Computers* 28, 1 (2011), 14–21.

[4] Nathan Binkert, Bradford Beckmann, Gabriel Black, et al. 2011. The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (2011), 1–7.

[5] James Bucek, Klaus-Dieter Lange, and Jóakim von Kistowski. 2018. SPEC CPU2017: Next-generation compute benchmark. In *Proc. ICPE Companion*. 41–42.

[6] R. Chadha and J. Bhasker. 2012. *An ASIC Low Power Primer: Analysis, Techniques and Specification.* Springer.

[7] Jonathan Chang, Yen-Huei Chen, Gary Chan, et al. 2020. 5nm 135Mb SRAM in EUV and high-mobility-channel FinFET technology with metal coupling and charge-sharing write-assist circuitry schemes for high-density and low-VMIN applications. In *Proc. ISSCC*. 238–240.

[8] Jonathan Chang, Yen-Huei Chen, Wei-Min Chan, et al. 2017. 7nm 256Mb SRAM in high-k metal-gate FinFET technology with write-assist circuitry for low-VMIN applications. In *Proc. ISSCC*. 206–207.

[9] Lawrence T. Clark, Vinay Vashishtha, Lucian Shifren, et al. 2016. ASAP7: A 7-nm FinFET predictive process design kit. *Microelectronics Journal* 53 (2016), 105–115.

[10] Pete Ehrett, Todd Austin, and Valeria Bertacco. 2021. Chopin: Composing cost-effective custom chips with algorithmic chiplets. In *Proc. ICCD*. 395–399.

[11] Yinxiao Feng and Kaisheng Ma. 2022. Chiplet actuary: A quantitative cost model and multi-chiplet architecture exploration. In *Proc. DAC*. 121–126.

[12] Pouya Fotouhi, Sebastian Werner, Jason Lowe-Power, et al. 2019. Enabling scalable chiplet-based uniform memory architectures with silicon photonics. In *Proc. MEMSYS*. 222–334.

[13] Md Arafat Kabir and Yarui Peng. 2020. Chiplet-package co-design for 2.5D systems using standard ASIC CAD tools. In *Proc. ASP-DAC*. 351–356.

[14] Chulwoo Kim, Hyun-Woo Lee, and Junyoung Song. 2014. *High-Bandwidth Memory Interface.* Springer.

[15] Hyunjin Lee, Sangyeun Cho, and Bruce R. Childers. 2011. CloudCache: Expanding and shrinking private caches. In *Proc. HPCA*. 219–230.

[16] Fuping Li, Ying Wang, Yujie Wang, et al. 2024. Chipletizer: Repartitioning SoCs for cost-effective chiplet integration. In *Proc. ASP-DAC*. 58–64.

[17] Tao Li, Jie Hou, Jinli Yan, et al. 2020. Chiplet heterogeneous integration technology—status and challenges. *Electronics* 9, 4 (2020), 670.

[18] Zichao Ling, Lin Li, Yi Huang, et al. 2025. AuxiliarySRAM: Exploring elastic on-chip memory in 2.5D chiplet systems design. In *Proc. GLSVLSI*. Association for Computing Machinery, New York, NY, USA, 879–885. https://doi.org/10.1145/3716368.3735238

[19] Samuel Naffziger, Noah Beck, Thomas Burd, et al. 2021. Pioneering chiplet technology and design for the AMD EPYC™ and Ryzen™ processor families: Industrial product. In *Proc. ISCA*. 57–70.

[20] Moinuddin K. Qureshi and Yale N. Patt. 2006. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Proc. MICRO*. 423–432.

[21] Dylan Stow, Itir Akgun, and Yuan Xie. 2019. Investigation of cost-optimal network-on-chip for passive and active interposer systems. In *Proc. SLIP*. 1–8.

[22] Haofeng Sun, Bobi Shi, Thong Nguyen, et al. 2024. Modeling and analysis of heterogeneously integrated chiplet-to-chiplet communication link in 2.5D advanced packaging. In *Proc. ECTC*. 1078–1084.

[23] Xiaofei Wang, Yusung Kim, Gwanghyeon Baek, et al. 2025. 0.021μm^2 high-density SRAM in Intel-18A-RibbonFET technology with PowerVia-backside power delivery. In *Proc. ISSCC*. 494–496.

[24] John A. White, J. W. Schmidt, and George K. Bennett. 2012. *Analysis of Queueing Systems.* Elsevier.

[25] Clifford Wolf. 2012. Yosys Open SYnthesis Suite. Open-source synthesis tool. https://yosyshq.net/yosys/