TFusionVcache (TFVcache) Policy v0.1

# TFusionVcache (TFVcache) Policy v0.1

## 1. 개요 (Overview)

일반적으로 \*\*TPU / GPU\*\*에서 \*\*AI 대규모 연산\*\*을 수행할 때는, \*\*수십 MB~수백 MB급 KVCache\*\*가 필요합니다.

- 보통 \*\*32 MB, 64 MB, 128 MB\*\* 혹은 그 이상의 캐시 용량이 사용되며,

- 이는 \*\*모델 크기, 배치 사이즈, 추론 시퀀스 길이\*\* 등에 따라 결정됩니다.

이번 설명에서는 \*\*다중 캐시 정책을 병행\*\*하는 사례를 보여주기 위해, \*\*64 MB\*\* 물리 캐시를 기준으로 설정했습니다.

즉, 하나의 64 MB 물리 캐시를 논리적으로 \*\*KV-policy\*\*와 \*\*Tensor-policy\*\* 두 가지 영역으로 나누어, 상황에 따라 가변적으로 용량을 재조정하는 구조를 \*\*TFusionVcache\*\*라 부릅니다.

---

## 2. 왜 필요한가? (동기)

### 2-1. 기존 계층형 캐시의 한계

CPU의 전통적 캐시는 L1 → L2 → L3 식의 계층 구조입니다.

계층이 깊어질수록 히트율은 좋아지지만, \*\*latency가 누적\*\*되어 GPU/TPU 같은 대규모 병렬 연산에는 불리합니다.

### 2-2. AI 워크로드의 특성

- \*\*KV-cache\*\* (LLM 디코더 단계) → 작은 크기의 토큰, 순차적·재사용 많은 접근.

- \*\*Tensor 연산\*\* (행렬/합성곱) → 큰 블록, 스트리밍 위주, 1회성 접근 많음.

- 동일한 캐시 정책으로 처리하면, 한쪽은 히트율 낮고 다른 쪽은 낭비가 많아집니다.

👉 따라서 \*\*워크로드 특성별로 맞춤형 정책을 병행\*\*해야 전체 효율이 올라갑니다.

---

## 3. TFusionVcache 구조

- 물리: \*\*64 MB 단일 캐시 뱅크(16 banks × 4 MB)\*\*

- 논리: \*\*KV-policy 영역\*\*과 \*\*Tensor-policy 영역\*\*

- 초기값: 32 MB / 32 MB → 동적 조정(최소 16 MB, 최대 48 MB)

### Front-End Cache Multiplexor

- 요청이 들어오면 \*\*Classifier\*\*가 “KV인지, Tensor인지” 판별 → 각각의 policy 엔진으로 전달

- 히트율, 미스패널티, DRAM 큐 점유율 등 Telemetry로 파티션 비율을 실시간 조정

---

## 4. 정책별 세부 동작

### KV-policy

- 작은 라인 크기(64–128B)

- 대체 정책: \*\*SRRIP + pin bit\*\* (토큰 윈도는 무조건 유지)

- 프리패치: “다음 토큰(+1 step ahead)” 미리 가져오기

- \*\*비유\*\*: 시험공부할 때 자주 보는 포스트잇은 책상 위에 계속 붙여둔다

### Tensor-policy

- 큰 블록 크기(256–512B), 스트림 버퍼 활용

- 대체 정책: \*\*NMR/Bypass 우선\*\* (1회성 데이터는 캐시 안 함)

- 프리패치: 타일 2개 이상 미리 준비

- \*\*비유\*\*: 마트에서 대용량 생수를 카트에 미리 두세 박스 쌓아두는 것

---

## 5. 동적 파티셔닝

S\_k = w1\*Hk - w2\*MPk - w3\*BQk

S\_t = w1\*Ht - w2\*MPt - w3\*BQt

Δ = S\_k - S\_t

if Δ > θ and KV\_MB < 48: KV\_MB += 4; T\_MB -= 4

elif Δ < -θ and T\_MB < 48: T\_MB += 4; KV\_MB -= 4

- Δ가 양수 → KV 캐시 확대

- Δ가 음수 → Tensor 캐시 확대

- 4 MB 단위 step, 쿨다운 주기 있음

---

## 6. 도면 (Visualization)

### 6-1. 정책 관점 (Algorithm view)

![TFusionVcache Policy v0.1](cache\_algorithm\_v0\_2\_1\_tfv.svg)

### 6-2. 데이터 플로우 관점

![TFusionVcache Dataflow v0.1](cache\_dataflow\_v0\_2\_1\_tfv.svg)

### 6-3. 캐시 블록 내부

![TFusionVcache Block Internals](cache\_block\_algo\_v0\_2\_1\_tfv.svg)

### 6-4. 시스템 연계

![TFusionVcache in System](cache\_centric\_system\_v0\_2\_1\_tfv.svg)

---

## 7. TCO (Total Cost Optimization) 관점

- \*\*DRAM/HBM 전송\*\*을 줄여줌 → 전송 횟수 줄수록 전기 요금/TCO 절감

- \*\*클럭을 높이지 않고도 latency 개선\*\* → 발열/쿨링 비용 절약

- \*\*같은 칩 면적에서 더 많은 성능 효율\*\* → CAPEX 대비 ROI 향상

\*\*비유\*\*:

- “매번 편의점 가서 물 사오는 것(DRAM)”보다, \*\*집 냉장고에 2리터 물을 쟁여두는 것(Cache)\*\*이 훨씬 싸고 편리하다.

- TFusionVcache는 \*\*냉장고 안을 ‘음료/물’ 칸처럼 나눠두고\*\*, 필요에 따라 칸 크기를 바꾸는 스마트 냉장고와 같다.

---

## 8. 결론

\*\*TFusionVcache\*\*는

- 평면적(1-depth) 구조,

- 정책별 최적화,

- 동적 파티션,

- QoS 기반 운영

을 통해 기존 계층형 캐시보다 \*\*낮은 latency, 더 높은 효율, 낮은 TCO\*\*를 달성할 수 있는 신개념 캐시입니다.

👉 학부생도 이 원리를 이해하면, GPU/TPU뿐 아니라 \*\*네트워크 라우터, SSD 컨트롤러, IoT 엣지 디바이스\*\* 등 다양한 시스템에도 응용할 수 있습니다.