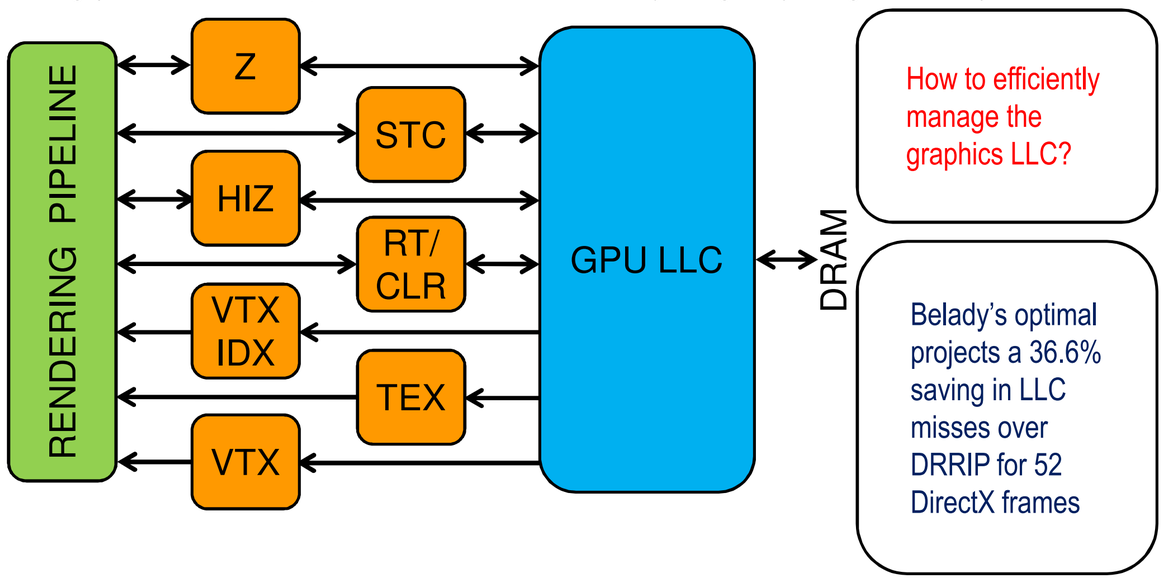
Efficient Management of Last-level Caches in Graphics Processors for 3D Scene Rendering Workloads

Jayesh Gaur (Intel), Raghuram Srinivasan (Ohio State University),
Sreenivas Subramoney (Intel), Mainak Chaudhuri (Indian Institute of Technology, Kanpur)

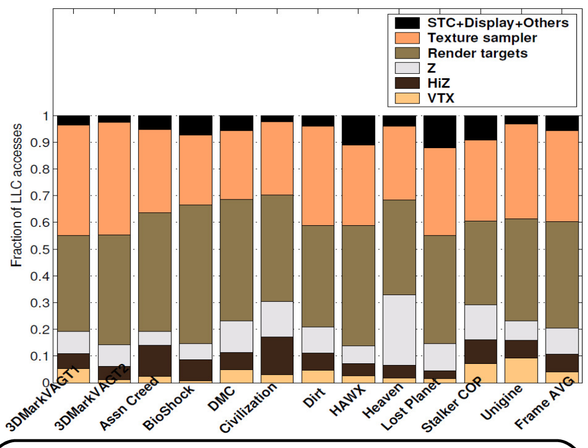
Last-level Cache (LLC) Interface in GPUs

➢3D scene rendering is implemented in the form of a pipleline in GPUs
➢Each pipeline stage accesses different types of data e.g., vertices, vertex indices, depth (Z), hierarchical depth (HiZ), stencil, render targets (or pixel colors), and texture for sampling
➢GPUs include small render caches for each such data stream
➢Recent commercial GPUs have incorporated reasonably large LLCs that can be shared by all these streams enabling far-flung intra-stream reuses and fast inter-stream producer-consumer sharing (Nvidia's Fermi, Kepler; AMD's GCN; Intel's Sandy Bridge, Ivy Bridge, Haswell)

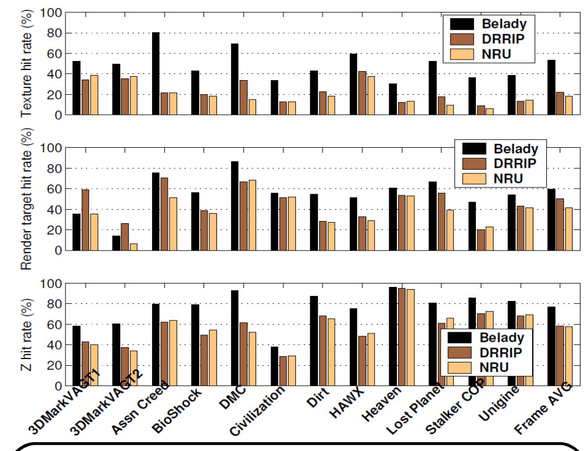


3D Graphics Streams and Reuses from the LLC

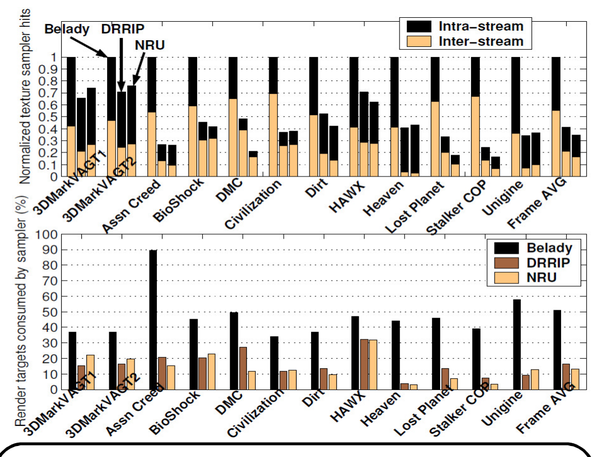
➢Characterization done on 52 frames selected from eight DirectX games and four DirectX benchmark applications spanning two versions of DirectX and three different frame resolutions
➢LLC is 8 MB 16-way non-inclusive/non-exclusive relative to the graphics render caches (no back-invalidation on LLC eviction)



More than 80% of LLC accesses come from Z, render target, and texture samplers



Large LLC hit rate gap between optimal policy and DRRIP or NRU for Z, RT, TEX

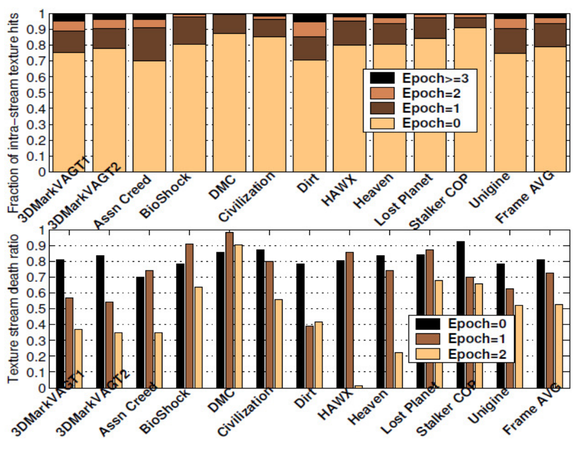


Large volume of potential LLC reuses from RT production to TEX consumption

With an optimal LLC policy, 55% of LLC hits enjoyed by the texture samplers come from render targets produced by color ROP writes; DRRIP and NRU fall significantly short

With an optimal LLC policy, 51% of all produced render target blocks are consumed by the texture samplers, while for DRRIP and NRU, this is only 16% and 13%, respectively

➢Let the life of an LLC block be divided into epochs demarcated by the LLC hits the block enjoys
➢Define *death ratio* and *reuse probability* of epoch E_k as $(|E_k|-|E_{k+1}|)/|E_k|$ and $|E_{k+1}|/|E_k|$, for $k \geq 0$
➢Examine intra-stream reuse behavior of texture sampler and Z streams in terms of epochs
➢A good LLC policy should victimize blocks from the epochs with low reuse probabilities



With an optimal LLC policy, most sampler hits come from the E_0 texture partition and almost all hits are covered by the E_0, E_1, E_2 partitions

While the blocks in the E_2 texture partition are mostly live, the reuse probabilities of the E_0 and E_1 texture epochs need to be learned dynamically

Stream-aware Graphics LLC Management

➢Our proposal modulates the two-bit RRPV of an LLC block on insertion and hits based on the reuse probability (RPROB) of different graphics streams; always replaces the block with RRPV three
➢The reuse probabilities are learned from a few sample sets that always execute the SRRIP policy
➢We describe our proposal as three increasingly better LLC management policies

Policy#1: Graphics Stream-aware Probabilistic Z and Texture caching (GSPZTC)

➢Z fill: RRPV ← (RPROB(Z) < 1/(t+1)) ? 3 : 2 or equivalently, RRPV ← (t.HIT(Z) < FILL(Z)) ? 3 : 2
➢The HIT and FILL counters are updated by the sample sets
➢The threshold t is chosen to be a power of two (specifically, eight) so that no multiplication is needed
➢TEX fill: RRPV ← (t.HIT(TEX) < FILL(TEX)) ? 3 : 0
➢RT fill: RRPV ← 0 to offer the highest protection to the render targets anticipating render to texture
➢Other fills: RRPV ← 2 as in SRRIP
➢All hits: RRPV ← 0 as in SRRIP

Policy#2: GSPZTC + Texture Sampler Epochs (TSE)

➢Instead of learning the overall reuse probability of sampler accesses, maintain the reuse probabilities of the E_0 and E_1 texture partitions
➢When a texture block becomes a member of the E0 or E1 partition, its RRPV is updated according to the reuse probability of the new partition E_k: RRPV ← (t.HIT(TEX, E_k) < FILL(TEX, E_k)) ? 3 : 0
➢Two state bits per LLC block keep track of which partition (E_0, E_1, or $E_{\geq 2}$) a texture block belongs to; the fourth state is used to identify a newly produced render target block

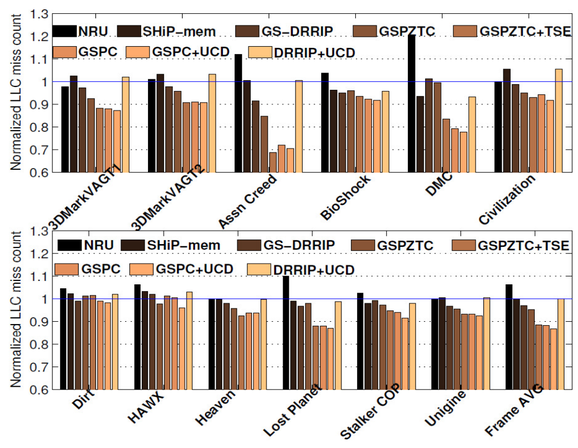
Policy#3: Graphics Stream-aware Probabilistic Caching (GSPC)

➢This policy further decides the render targets' insertion RRPV based on the render to texture reuse probability; in GSPZTC and GSPZTC+TSE, all render target blocks are filled with RRPV zero
➢Let the sample sets update two counters: PROD and CONS based on the number of blocks written to by the color ROPs (render target creation) and the number of such blocks consumed by the texture samplers from the LLC
➢RT fill: if PROD > 16.CONS then RRPV ← 3; else if 16.CONS ≥ PROD > 8.CONS then RRPV ← 2; else RRPV ← 0
➢Total hardware storage overhead is less than 0.5% of all LLC data array bits

Simulation Results

➢GPU clocked at 1.6 GHz, with 768 thread contexts and twelve texture samplers delivering peak shader throughput of nearly 2.5 TFLOPS and peak texture fill rate of 76.8 GTexels/second
➢Non-inclusive/non-exclusive LLC clocked at 4 GHz, 16-way set-associative, 8 MB or 16 MB in size
➢Eight-way banked DDR3-1600 DRAM with 15-15-15 latency

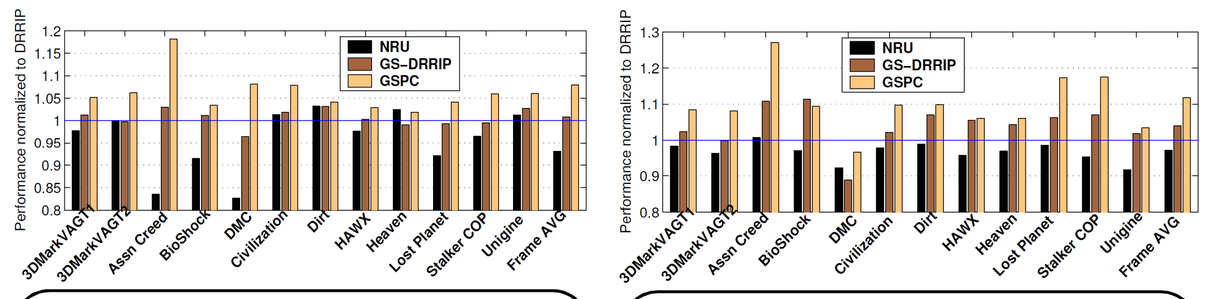
Normalized LLC Miss Count (8 MB LLC)



Relative to two-bit DRRIP, the proposed GSPC policy saves 11.8% LLC misses, on average in an 8 MB 16-way LLC

GSPC with uncached displayable color (GSPC+UCD) is the best policy and saves up to 29.6% and on average 13.1% LLC misses compared to two-bit DRRIP

Performance on 8 MB and 16 MB LLC



On an 8 MB LLC, the GSPC policy improves frame rate by up to 18.2% and on average 8% compared to DRRIP

On a 16 MB LLC, the GSPC policy improves frame rate by up to 27% and on average 11.8% compared to DRRIP

Sensitivity to Thread Count and DRAM Speed

With an 8 MB LLC and 512 thread contexts, the GSPC policy improves performance by 5.9% on average compared to DRRIP (down from 8% with 768 thread contexts)

With an 8 MB LLC and DDR3-1867 10-10-10 DRAM, the GSPC policy improves performance by 7.1% compared to DRRIP (down from 8% with DDR3-1600 15-15-15 DRAM)