Exploiting Dynamic Reuse Probability to Manage Shared Last-level Cache in CPU-GPU Heterogeneous Processors

Siddharth Rai
Mainak Chaudhuri

Department of CSE
IIT KANPUR
Indian Institute of Technology Kanpur, India

May 29, 2016
OUTLINE

SYSTEM ARCHITECTURE
SYSTEM AND WORKLOAD

MOTIVATION
SPEEDUP OVER BASELINE
LLC POTENTIAL MISS SAVINGS
POTENTIAL IMPROVEMENT AND BYPASS STUDY

PROPOSAL
Working-set Sampler
LLC Policies
Overhead

RESULTS
Comparison with Other Proposals

SUMMARY
Cache configurations:

- ROP has 2 level cache hierarchy
  2 KB and 32 KB/32 way
- Texture has 3 level cache hierarchy
  2 KB, 64 KB/16 way and 384 KB/48 way
- 16 KB vertex, 16 KB/16 way HiZ and 32 KB/8-way instruction
- GPGPU, 4 KB/4 way instruction, 32 KB/8 way data and 16 KB shared memory and 8 KB texture and constant

- Each out-of-order x86 core is clocked at 4 GHz
- GFX, 64 cores, 1 GHz, 4K thread, RR scheduling, 128 GTexels/s sampler, 64 GPixels/s ROP
- GPGPU, 6 cores, 1.5 GHz, 80 warps, two warp schedulers, alternate cycles
- 2 single channel MC, latency parameter 14-14-14, FR-FCFS
CPU executes memory sensitive SPEC 2006 applications
- GPU for graphics executes DX9 and OpenGL 2.0 games
- GPGPU executes CUDA applications
- Heterogeneous mix is formed by choosing 1, 2, 4 CPU applications for a GPU application
- Each CPU application commits at least 250 million instructions, GPU application till the end
**Speedup in Standalone Mode**

- Effect of mutual interference in one, two and four core configurations
  1. CPU in standalone mode improves over heterogeneous mode by 35%, 35%, and 26%
  2. GPU in standalone mode improves over heterogeneous mode by 21%, 32%, and 54%
LLC Miss Savings

- Baseline runs SRRIP with some GPU specific optimizations
- SHiP-hybrid runs SHiP-PC for CPU and SHiP-mem for GPU
- Bypass is OPT guided

**Observations:**
- SHiP-hybrid is the best
- Bypass has nominal effect

- OPT saves 35%, 34%, 30% misses for one, two, and four core CPU and one GPU configuration, respectively over the baseline
- SHiP-hybrid saves 7%, 8%, 11% misses for one, two, and four core CPU and one GPU configuration, respectively over the baseline
GPU Speedup with Ideal LLC and Effect of GPU Read Bypass

- GPU achieves on average 63% speedup over baseline with ideal LLC
- Depth and texture are most critical
- Bypassing have little effect on performances
- With increasing CPU core counts, returns due to bypassing diminish
LLC MANAGEMENT USING DYNAMIC REUSE PROBABILITY

- LLC policy can be divided into four component policies
- Read hit, write fill and write hit ages are assigned based on dynamic reuse probability measured using working set sampler (WSS)
**Working-set Sampler**

- Consist of a set-associative WSS cache, set of counters to estimate reuse probabilities and a WSS cache controller
- Each entry in WSS cache track blocks from one sampled page
- Each WSS cache block contains per-page valid and TAG bits to uniquely identify a valid page entry
**Working-set Sampler (Contd.)**

- All blocks from one sampled page are tracked in the corresponding WSS cache entry.
- For each tracked block there is a valid (V), write (W) and stream-id (SID) bits.
- W bit is set when block is written and is reset when block is read.
**WSS Cache Access and Update**

- Looked up on every LLC access in parallel
- WSS cache miss, replacement is usually turned off
  1. Random replacement for low occupancy (< 32 entries)
- WSS cache hit
  1. Valid bit and stream is set for an invalid entry
  2. Reuse event is recorded for a valid entry
WSS Cache Access and Update (Contd.)

- Reuse counter array with one counter for each stream
  1. WR counters: track write to read reuses
  2. RR counter: track read to read reuses
  3. WA counter: tracks write access count

- MAX REUSE register for maximum reuse seen

- WSS cache is invalidated and counter are halved every 512K LLC access
Read Fill and Read Hit Policy

- Read fill policy is borrowed from existing LLC insertion policies
  1. CPU blocks are filled using SHiP-PC policy
  2. GPU blocks are filled using DRRIP

- On hit blocks are promoted to RRPV 0 except for dynamic texture blocks
  1. Demoted to RRPV 3 if read reuse probability is below 1/64
  2. RRPV is set to 2 if it is between 1/64 and 1/2
  3. Otherwise, block is promoted to RRPV 0
**Write Fill Policy**

- **Write fill RRPV**
  1. Needed only for GPU as for CPU LLC is inclusive
  2. 0, if WR is above MAX REUSE/3 or \( \geq 1/8 \)
  3. Pin recommended if WR is above MAX REUSE/2 or \( \geq 1/8 \)
  4. 3, if 1, 2, 3 fail and WA is above 128K
  5. If 1, 2, 3 fail, fill with RRPV 2
**Write Hit Policy**

- Decides block’s RRPV on a write hit
- Chooses between two competing policies
  1. Congestion oblivious write hit policy
     1.1 Needed for both CPU and GPU blocks
     1.2 $0$, if $WR$ is above $MAX\ REUSE/2$ or $\geq 1/16$
     1.3 Pin recommended if $WR$ is above $MAX\ REUSE/2$ or $\geq 1/16$
     1.4 Otherwise, no change
  2. Congestion aware write hit policy
     2.1 2, if $WR$ is above $MAX\ REUSE/2$ or $\geq 1/16$ and current RRPV is 3
     2.2 Otherwise, no change
  3. Followed policy is decided based on a set duel
**Storage Overhead**

- WSS cache
- Counter array
- SHiP counter

- 86 bits × 2K entries
- 264 bytes
- 3 bits × 16K entries
- 3 bits × 256K LLC blocks

▶ Total storage overhead is 124 KB for 16MB LLC
Our proposal on averages improves CPU and GPU performances by 2%, 4%, 7% and 8%, 9%, 12%, for one, two, and four core CPU and one GPU configuration, respectively.
Our proposal on average saves 13%, 12%, and 13% LLC misses for one, two, and four core CPU and one GPU configuration, respectively.
- For all mixes, HeLM improve CPU by 6% for four core configurations while sacrificing about 2% of GPU performances
- For only CUDA mixes HeLM improves CPU and GPU performances
- DRP outperforms HeLM in one and two core configuration
- In one core configuration HeLM improves GPU by 9% whereas DRP improves by 7%
We have presented a novel LLC management policy for emerging heterogeneous CMPs. Our proposal estimates reuse probability of CPU and GPU streams. Central contribution of our proposal is a working set sampler cache. For a four core CPU and one GPU configuration our proposal improves CPU and GPU by 7% and 12% on average. It saves 13% LLC read misses on average.
Thank you!