StimulusCache: Boosting Performance of Chip Multiprocessors with Excess Cache

Hyunjin Lee, Sangyeun Cho, and Bruce R. Childers

Presented by Xinran Yu
Growing number of cores in a chip multiprocessor (CMP)

- **Problem:**
  - Smaller device size & Greater integration -> chip yield degrades significantly.

- **Solution:**
  - Chip vendors adopt a design strategy: “core disabling”
    - e.g. AMD sells tri-core chips, which is a byproduct of a quad-core chip.

But...
Another Problem:

- Logic part has a much lower yield than the on-chip memory -> waste much memory capacity.

Solution (This paper):

- StimulusCache, which decouples the L2 caches of faulty compute cores and employs them to assist applications on other working cores.
This paper contributes:

- A new yield model for processor components
- Architectural support for StimulusCache
- Strategies to utilize excess caches
- An evaluation of StimulusCache
Decoupled yield model

- Given multiple functional blocks in a chip and their individual yields ($Y_{blocki}$), the chip yield

$$Y_{Die} = \prod Y_{blocki} \quad (1 < i < N)$$

$$Y_{blocki} = Y_{logic_i} \times Y_{memory_i}$$
StimulusCache

**Basic idea**
- Exploit "excess cache" (EC) in a failed core
- Core disabling (yield ↑) + larger cache capacity (performance ↑)
- EC is a shared resource among multiple cores

**Simple HW architecture extension**
- Cache controller has knowledge about EC utilization
- L2 cache are *chain linked* using vector tables

**Modest OS support**
- OS manages the hardware data structures in cache controllers to set up EC utilization policies
Hardware design support

- **Conventional core:** disable the whole core including its private L2 cache.

- **StimulusCache:** aggressively pushes the isolation point beyond the L2 cache controller.
Flexible accessibility:

**ECAV:** parallel searching (c) → faster

**NECP:** serial searching (d) → less network traffic

**SCV:** Coherence management 
→ When L1 data invalidation occurs, the SCV identifies the cores that need to receive an invalidation message. For functional cores, SCV entries are empty because their local L2 caches are not shared.
Software support

The system software would assign an EC to a core in a way that meets the application needs by properly setting the values of ECAV, SCV, and NECP.

**e.g.**

OS may enforce an EC utilization policy before programming cache controllers.

OS may program cache controllers.

OS may take into account workload characteristics before programming.
Example 1

Core 6 is allocated 2 EC from core 1 (higher priority) and core 2
Example 2 (Coherence)

- If exclusive L2 data is migrated to an EC, an L1 data invalidation is not needed. (a)

- Two cores have the same data which is shared by cache-to-cache transfer. If one core should evict this shared data, the data is not migrated to the excess cache. (b)

- If exclusive data in L2 is migrated to the EC, no L1 invalidation. (c)

- If the data in the EC is migrated to P2's L2, P1's L1 should be invalidated. (d)
Excess Cache Utilization Policies

Static private:
only one core can use a particular EC as assigned by the system.

Static sharing:
use the available EC as a shared resource for multiple cores.

Dynamic sharing:
cache capacity demands from cores are continuously monitored
and EC are allocated dynamically to maximize their utility.
Experiment Set Up

Intel ATOM-like cores with 16-stage pipeline @ 2GHz

Memory hierarchy
L1: 32KB I/D, 1 cycle
L2: 512KB, 10 cycles
Main memory: 300 cycles, contention modeled

On-chip network
Crossbar for 8-core CMP (4 cores + 4 ECs)
2D mesh for 32-core CMP (16 cores + 16 ECs)
Contention modeled

Benchmark suit
SPEC CPU2006, SPLASH-2, and SPECjbb2005
Performance

(a) Performance improvement of single-threaded applications. (b) Misses per 1,000 instructions. (c) Miss reduction.

(a) Performance improvement with the static private scheme. (b) Performance improvement of individual programs.
(a) Performance improvement with the static sharing scheme. Workloads are grouped into: Group 1: “Large gain,” Group 2: “Limited gain due to heavy applications,” Group 3: “Large gain due to astar,” and Group 4: “Small gain.” (b) Performance improvement of individual programs with four excess caches. astar consistently shows a high gain of 135%.

(a) Performance improvement with the dynamic sharing scheme. (b) Additional performance gain with the dynamic sharing scheme compared with the static sharing scheme with four excess caches. Grouping follows Figure 7(a).
Conclusions

Processing logic yield vs. L2 cache yield
A large number of excess L2 caches

StimulusCache
Core disabling (yield $\uparrow$) + larger cache capacity (performance $\uparrow$
Simple HW architecture extension + modest OS support

Performance improved


Questions?
Thank you!