Exploiting Private Local Memories to Reduce the Opportunity Cost of Accelerator Integration

Emilio G. Cota
Paolo Mantovani
Luca P. Carloni

Columbia University
What accelerators, exactly?
The diagram illustrates the Generality vs. Efficiency trade-off in computing. The vertical axis represents the specialization level, while the horizontal axis indicates energy efficiency. Various computing paradigms such as CPUs, Many-cores, Multi-cores/Asymmetric, GPUs/DSPs, FPGAs, and ASICs are depicted. The end of Dennard scaling explains the surge of interest in specialization. This work is highlighted as a point of interest in the diagram.
Problem: Accelerators' Opportunity Cost

An accelerator is only of *utility* if it applies to the system's workload.

Consequence: Integrating accelerators in general-purpose architectures is rarely cost-effective.
Private Local Memories (PLMs)

Example: Sort Accelerator to sort FP vectors

Stage 1
Parallel Bubble Sort
64-port PLM

Stage 2
Merge Sort
64-port PLM

Input
2-port PLM

Output
2-port PLM

Tailored, many-ported Private Local Memories (PLMs) are key to exploit all parallelism in the algorithm.
Key Observations:

1. Accelerators are mostly memory

   "An average of 69% of accelerator area is consumed by memory"

   Lyons et al., "The Accelerator Store", TACO'12

   Accelerator examples: AES, JPEG encoder, FFT, USB, CAN, TFT controller, UMTS decoder.

2. Average Accelerator Memory Utilization is low

   Not all accelerators on a chip are likely to run at the same time
Related Work

Proposal [1]: The Accelerator Store

Shared memory pool that accelerators allocate from

Proposals [2,3]: Memory for Cache & Accelerators

Substrate to host cache blocks or accelerator buffers

Limitation: storage is external to accelerators

- High-bandwidth PLMs cannot tolerate additional latency
- Complicate accelerator designs
  - Hide PLM latency with pipelining, or reduce performance

Our proposal: ROCA

Observation #3: accelerator PLMs provide a de facto NUCA substrate

Goal:
To extend the LLC with PLMs when otherwise not in use

✓ Applies to all accelerator PLMs, not only low-bandwidth ones
✓ Minimal modifications to accelerators
Last-Level Cache Capacity Over Time

*Source: cpudb + Intel ARK*

- 90 nm
- 65 nm
- 45 nm
- 32 nm
- 22 nm
- 14 nm
- Year Average

Year Average
ROCA
ROCA. How to...

1. handle intermittent accelerator availability,

2. accommodate accelerators of different sizes,

3. transparently coalesce accelerator PLMs,

..with minimum overhead and complexity?
1. core0's L1 misses on a read from 0xf00, mapped to the L2's *logical bank1*
2. L2 bank1's tag array tracks block 0xf00 at acc2; sends request to acc2
3. acc2 returns the block to bank1
4. bank1 sends the block to core0

- Additional latency for hits to blocks stored in accelerators
- Return via the host bank guarantees the **host bank** is the only coherence synchronization point
  - No changes to coherence protocol needed
ROCA Host Bank

- **Enlarged tag array** for accelerator blocks
  - Ensures **modifications** to accelerators are **simple**

- Leverages Selective Cache Ways [*] to **adapt to** accelerators' intermittent availability
  - Dirty blocks are flushed to DRAM upon accelerator reclamation

4-way example: 2 local, 2 remote ways

### Logical Bank Way Allocation

Increasing associativity helps minimize waste due to uneven memory sizing across accelerators (Ex. 2 & 3)

- Power-of-two number of sets not required (Ex. 4), but
  - complicates set assignment logic [*]
  - requires full-length tags: modulo is not bit selection anymore

[*] André Seznec, "Bank-interleaved cache or memory indexing does not require Euclidean division", IWDDD’15
**Coalescing PLMs**

- **PLM manager** exports same-size dual-ported SRAM banks as multi-ported memories using MUXes
- ROCA requires an additional NoC-flit-wide port, e.g. 128b
Coalescing PLMs

SRAMs are accessed in parallel to match the NoC flit bandwidth

- Bank offsets can be computed cheaply with a LUT + simple logic
- Discarding small banks and SRAM bits a useful option
ROCA: Area Overhead

- Host bank's **enlarged tag array**
  - 5-10% of the area of the data it tags (2b+tag per block)

- Tag storage for **standalone directory** if it wasn't there already
  - Inclusive LLC would require prohibitive numbers of recalls
  - Typical overhead: **2.5%** of LLC area when LLC = 8x priv

- **Additional logic:** way selection, PLM coalescing logic
  - **Negligible** compared to tag-related storage
Assuming no accelerator activity,

- **6M ROCA** can realize **70%/68%** of the performance/energy efficiency benefits of a **same-area 8M S-NUCA**
  - retaining accelerators' potential orders-of-magnitude gains

Configurations:

- 2M S-NUCA baseline
- 8MB S-NUCA (not pictured)
- same-area 6M ROCA, assuming accelerators are 66% memory (below the typical 69%)
Also in the paper

- **Sensitivity studies** sweeping accelerator activity over
  - **space** (which accelerators are reclaimed)
  - **time** (how frequently they are reclaimed)

- **Key result**: *Accelerators with idle windows >10ms are prime candidates for ROCA*
  - perf/eff. within 10/20% of that with 0% activity
Accelerators can be highly-specialized, fixed-function units and still be of general-purpose utility.
backup slides
Why Accelerators?

- Every generation provides less efficient transistors, i.e. power density is growing
- Single-threaded perf. improvements slowing down
- Parallelization gains bounded by Amdahl's Law

*a.k.a. "the end of the multi-core era"

Esmaeilzadeh et al., "Dark Silicon and the End of Multicore Scaling", ISCA'11
Why Accelerators?

- If performance increases are to be sustained, we need **efficiency gains** well beyond what microarchitectural changes can provide.

- **Accelerators** achieve this via **specialization**
Per-core LLC Capacity Over Time

Source: cpudb + Intel ARK

Year Average

90 nm - 32 nm - Year Average
65 nm - 22 nm
45 nm - 14 nm
## Simulated Systems

<table>
<thead>
<tr>
<th>cores</th>
<th>16 cores, i386 ISA, in-order IPC=1 except on memory accesses, 1GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 caches</td>
<td>Split I/D 32KB, 4-way set-associative, 1-cycle latency, LRU</td>
</tr>
</tbody>
</table>
| L2 caches | 8-cycle latency, LRU  
S-NUCA: 16ways, 8 banks  
ROCA: 12 ways |
| Coherence | MESI protocol, 64-byte blocks, standalone directory cache |
| DRAM | 1 controller, 200-cycle latency, 3.5GB physical |
| NoC | 5x5 or 7x7 mesh, 128b flits, 2-cycle router traversal, 1-cycle links, XY router |
| OS | Linux v2.6.34 |
Flushed delay

- 330 64-byte blocks (i.e. largest amount in our tests)
- sufficiently buffer DRAM controller
- 128b NoC flits

\sim 10560 \text{ cycles}, \text{i.e.} 10.5\text{us at }1\text{GHz}