CoMerge
Toward Efficient Data Placement in Shared Heterogeneous Memory Systems

Thaleia Dimitra Doudali
Ada Gavrilovska
Motivation
Performance slowdown in heterogeneous memory systems.

Application data objects ➡ DRAM cost ↑ Non Volatile Memory ➡

Heterogenous Memory Subsystem

higher access latency ⇒ performance slowdown from ‘all-data-in-DRAM’

How to reduce the slowdown?
Existing Solutions
Data tiering that maximizes DRAM accesses.

Think about which objects get allocated in DRAM.

- More memory requests with lower latency

**Application**

**Data objects**

**DRAM**

**Non Volatile Memory**

**Heterogenous Memory Subsystem**

Existing Solutions
1. X-Mem - Dulloor et al.
2. Dataplacer - Shen et al.
3. Valgrind extension - Peña, Balaji.
Problem Statement
Limited Utility of Existing Solutions in Shared Systems.

Do the partitioning techniques using existing solutions:
- Reduce the slowdown across all collocated applications?
- Maximize DRAM utilization?

⇒ NO
Our Contributions
What do we need to do differently?

1. Sorting objects within one application: *co-benefit* metric captures:
   a. Exact contribution of a data object to overall application runtime.
   b. Overall application sensitivity to execution over Non-Volatile Memory.

2. Distributing DRAM across applications: *CoMerge* memory sharing technique.
   a. Mitigates slowdown across all collocated applications.
   b. Maximizes the DRAM usage.
Observations
What are we going to see next?

1. Not all applications are slowed down in the same degree when accessing Non Volatile Memory.
2. Not all data objects of an application help reduce the performance slowdown, when placed in DRAM.

**Polybench Benchmarks**
- 30 simple algebraic kernels.
- Single-threaded.

**CORAL Suite of mini-apps**
- 3 HPC representative kernels.
- Multi-threaded. OpenMP.

Hardware Testbed

Emulate Non Volatile Memory for various combinations of *reduced bandwidth* and *increased latency*.

e.g. B 0.5 : L 2
0.5 times less bandwidth : 2 times more latency
Overall Application Sensitivity

Do all applications get slowed down in the same way when accessing Non Volatile Memory?

Performance slowdown across Polybench/C, normalized to ‘all-data-in-DRAM’ execution.

Applications show different levels of sensitivity to execution over slower memory components.
Data Object Sensitivity
Do all data objects help minimize the slowdown, when allocated in DRAM?

Observations
1. For non or low sensitive apps, doesn’t matter which object is in DRAM.
2. Different data objects can contribute equally to the application runtime.
3. There can be objects whose allocation in DRAM is the only way to mitigate slowdown.
Co-Benefit Metric
Can we capture the previous observations?

<table>
<thead>
<tr>
<th>Run Time</th>
<th>Objects in DRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>All</td>
</tr>
<tr>
<td>t(O)</td>
<td>object O</td>
</tr>
<tr>
<td>S</td>
<td>None</td>
</tr>
</tbody>
</table>

How much does a specific object help reduce the slowdown? How can we make sure that objects of higher sensitivity kernels are getting prioritized?

e.g. $B(O) = 0.9$ $\Rightarrow$ $coB(O) = 0.9 \times \text{low sensitivity} = 0.9$
$coB(O) = 0.9 \times \text{high sensitivity} = 3.9$
DRAM Distribution
What are the goals of an efficient technique?

1. Minimize overall runtime slowdown across all applications.

2. Maximize the utilization of DRAM.
Sharing DRAM
Sorting objects using co-benefit metric.

---

**Normalized Execution Time**

Fair Merge

CoMerge

DRAM
Summary
More detailed analysis in the paper

Partitioning & existing solutions

Equal Split
- xsbench
- clomp
- stream

Proportional Split
- xsbench
- clomp
- stream

7x slowdown
6x

Fair Merge
- xsbench
- clomp
- stream

CoMerge
- xsbench
- clomp
- stream

2.7x slowdown
2.6x

Co-Benefit metric allows CoMerge to achieve:
- Lower runtime across all collocated applications.
- Higher DRAM utilization.

Sharing & co-benefit metric