Managing Heterogeneous Memory Infrastructures

ecoHMEM meets SHAMBLES

Jointly presented by:

Hatem El-Shazly

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Marios Asiminakis

FORTH-ICS
Computer Architecture & VLSI Systems

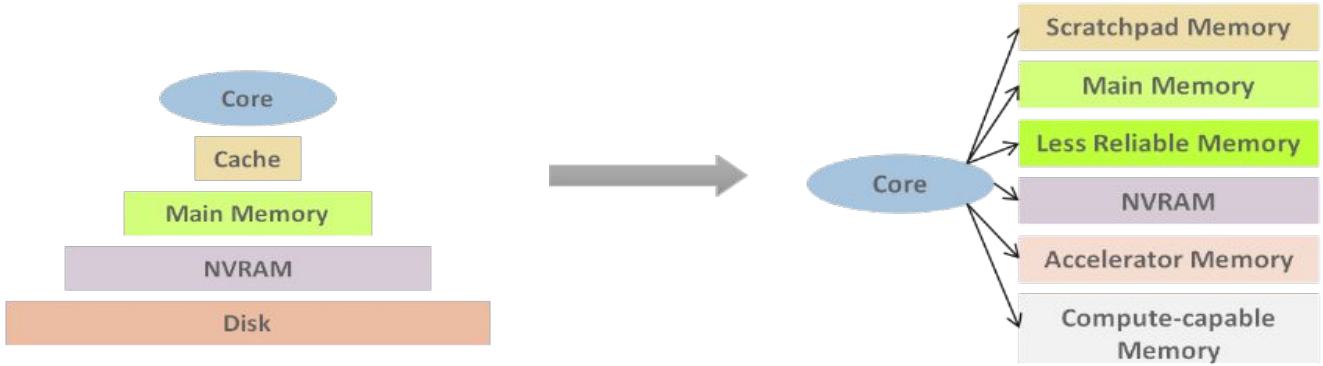
Nov. 25, 2022

Outline

- Overview & Motivation
- Objective: Memory Optimization Cycle
- Integration Components:
  - ecoHMEM (BSC Contribution)
  - SHAMBLES (FORTH Contribution)
- Towards an integrated solution
- Current status
- Conclusions

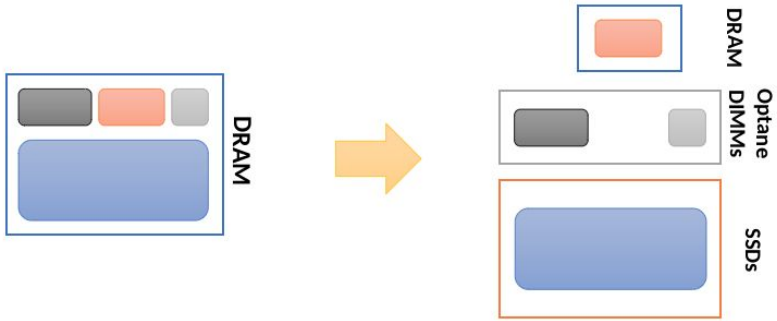
Overview & Motivation

- Applications produce increasing amounts of data
- HPC execution platforms with heterogeneous memory resources
  - First-class citizens
  - Move from hierarchical to explicitly managed



Overview & Motivation (Cont.)

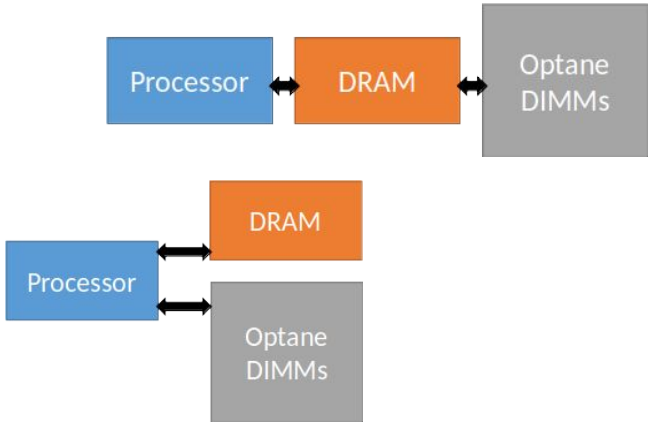
- Heterogeneous Memory Systems
  - KNL: R.I.P
  - Byte-addressable NVRAM / Persistent
    - **Intel® Optane™ (PMEM)**
  - CPUs + **HBM** (e.g., Sapphire Rapids)
  - **CXL Memory Pools**
  - Also GPUs



- Goals:
  - Maximize Performance
  - Minimize energy
  - …

Overview & Motivation (Cont.)

- Intel Optane Persistent Memory (DIMMs)
- Memory Mode
  - DRAM as cache for Optane DIMMs
  - No Applications modification
- App Direct Mode
  - DRAM and Optane DIMMs are addressable
  - Software managed

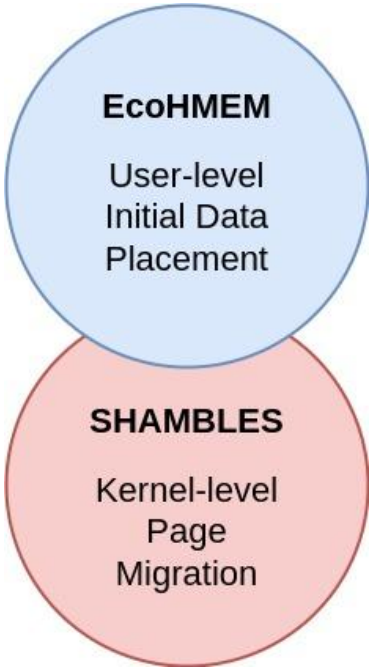


- Application's data distribution?
  - OS? Heuristics? On-the-fly monitoring? Hardware-assisted? Historic data? User hints?
  - Need ecosystem to assist users/developers: Profilers, libraries, runtime systems

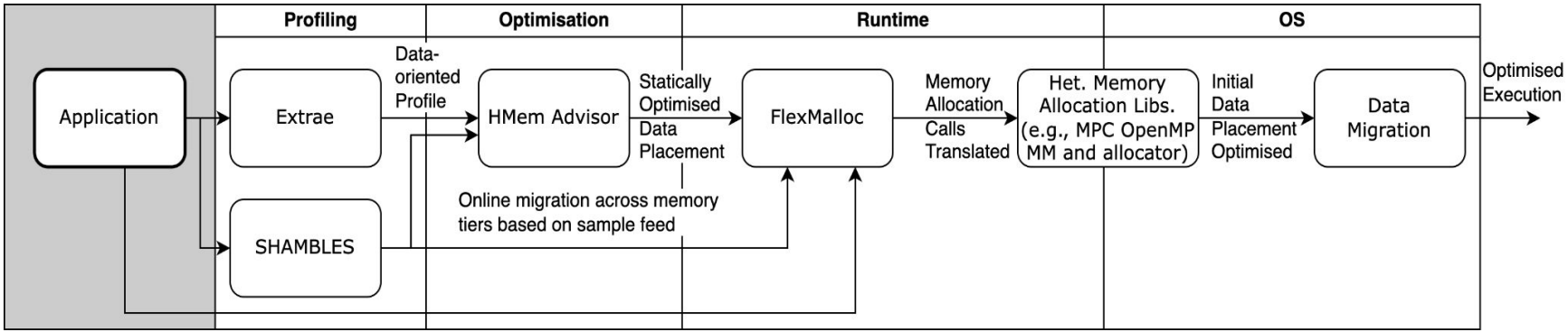
Overview & Motivation (Cont.)

- ecoHMEM (BSC contribution): User-level data placement
  - User-level profiles + Placement algorithms + allocator to honour data distributions

- SHAMBLES (FORTH contribution): Kernel-level data migration
  - Use page faults + custom allocation library to migrate pages



Objective: Memory Optimization Cycle

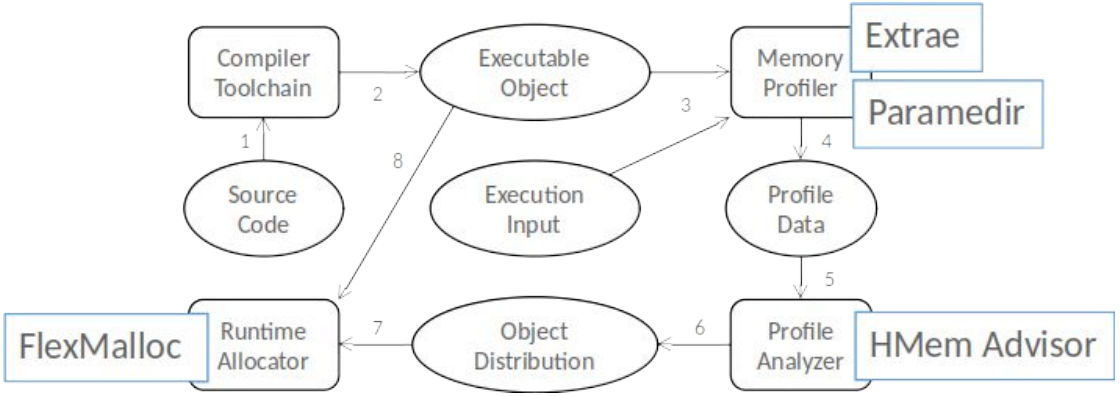


[diagram created by Simon Pickartz (WP3)]

Integration Components (1-2)

The ecoHMEM Framework

ecoHMEM



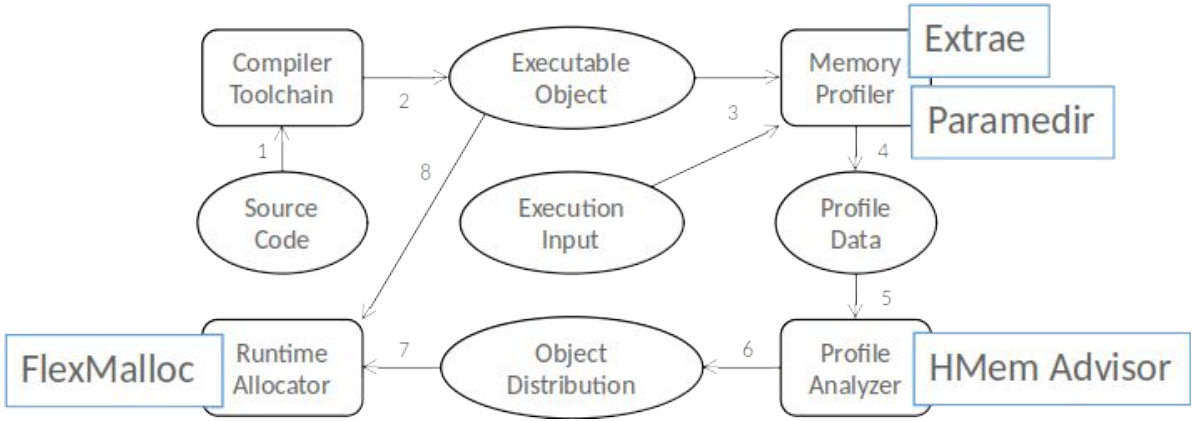
IEEE Cluster'22

ecoHMEM: Improving Object Placement Methodology for Hybrid Memory Systems in HPC

Marc Jordà, Siddharth Rai, Eduard Ayguadé, Jesús Labarta and Antonio J. Peña

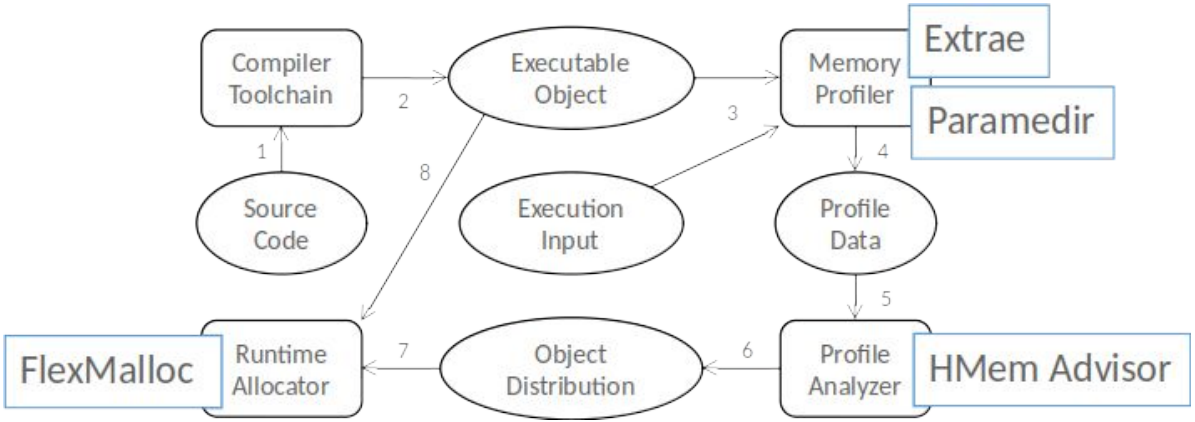
ecoHMEM (Cont.)

- Offline Profiling (Extrae & Paramedir)
  - Collects allocations address, size, and allocation point callstack (our object ID)
  - Sampling of precise profiling events (PEBS) for LLC load and L1 store misses, including accessed address



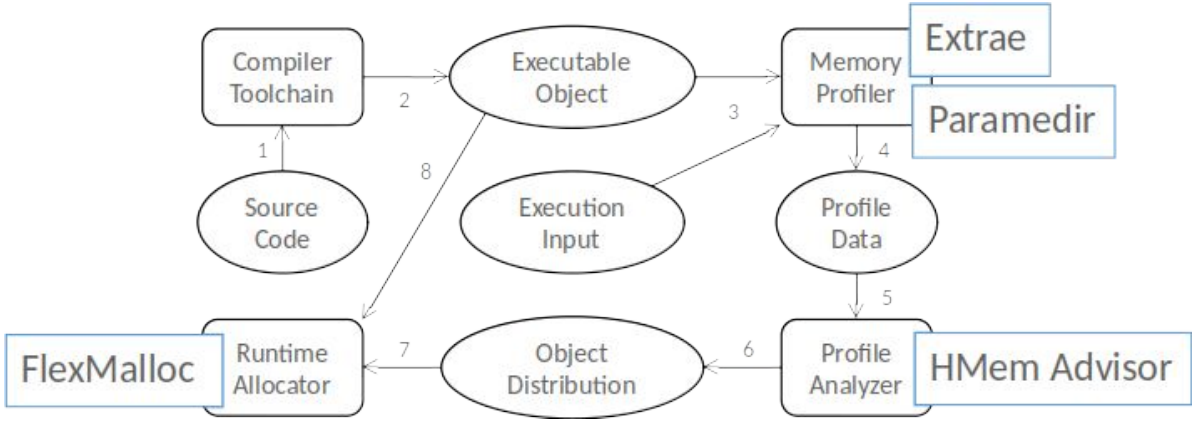
ecoHMEM (Cont.)

- Offline Profiling (Extrae & Paramedir)
- Analysis (HMem Advisor)
  - Computes per-object cost using heuristics based on profiling data to assign each object to a memory tier
  - Greedy relaxation of the 0/1 multiple knapsack problem + optional fine-tuning heuristics



ecoHMEM (Cont.)

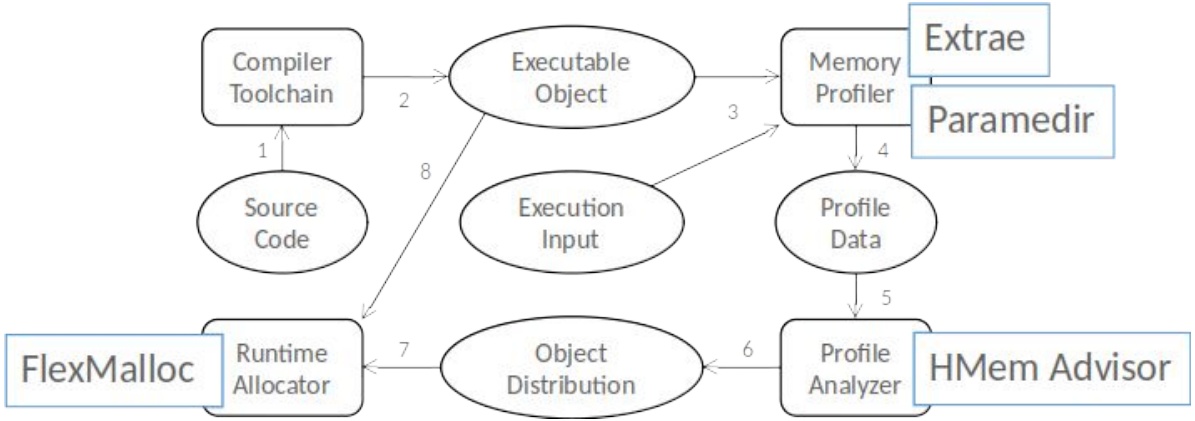
- Offline Profiling (Extrae & Paramedir)
- Analysis (HMem Advisor)
- Production execution with optimized automatic data placement (user-level interposition)
  - Original binary with FlexMalloc library, which interposes memory allocation functions to redirect each allocation to the corresponding memory tier



ecoHMEM (Cont.)

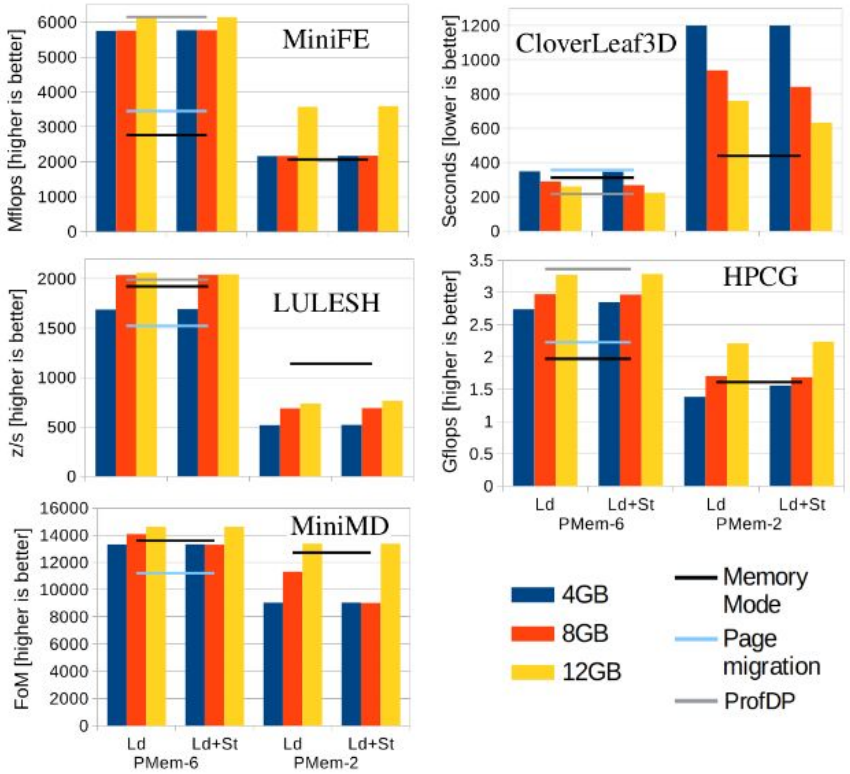
- Offline Profiling (Extrae & Paramedir)
- Analysis (HMem Advisor)
- Production execution with optimized automatic data placement (user-level interposition)

Fully automatic!
No need for
application mods



Results

- More detailed analysis in the ecoHMEM paper
- ecoHMEM + Optane App direct vs Optane Memory Mode (baseline)
- With 12GB, in all PMem-6 and most Pmem-2 cases, our framework performs better than MM and KPM
  - Reasons: MM cache misses, memory bound
  - MiniFE: ~2X speedup
  - HPCG: ~1.6x speedup
  - CloverLeaf3D: 10% slowdown with 4GB, gradual improvement with increasing DRAM sizes
  - LULESH: 7% improvement
  - MiniMD: 8% improvement
- Performance in pair (+/- 5%) with ProfDP - ecohmem provides much easier and productive execution workflow

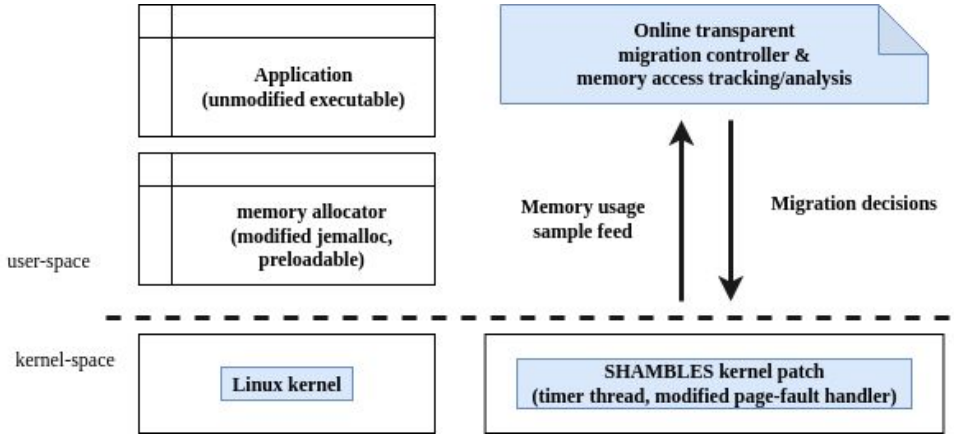


Integration Components (2-2)

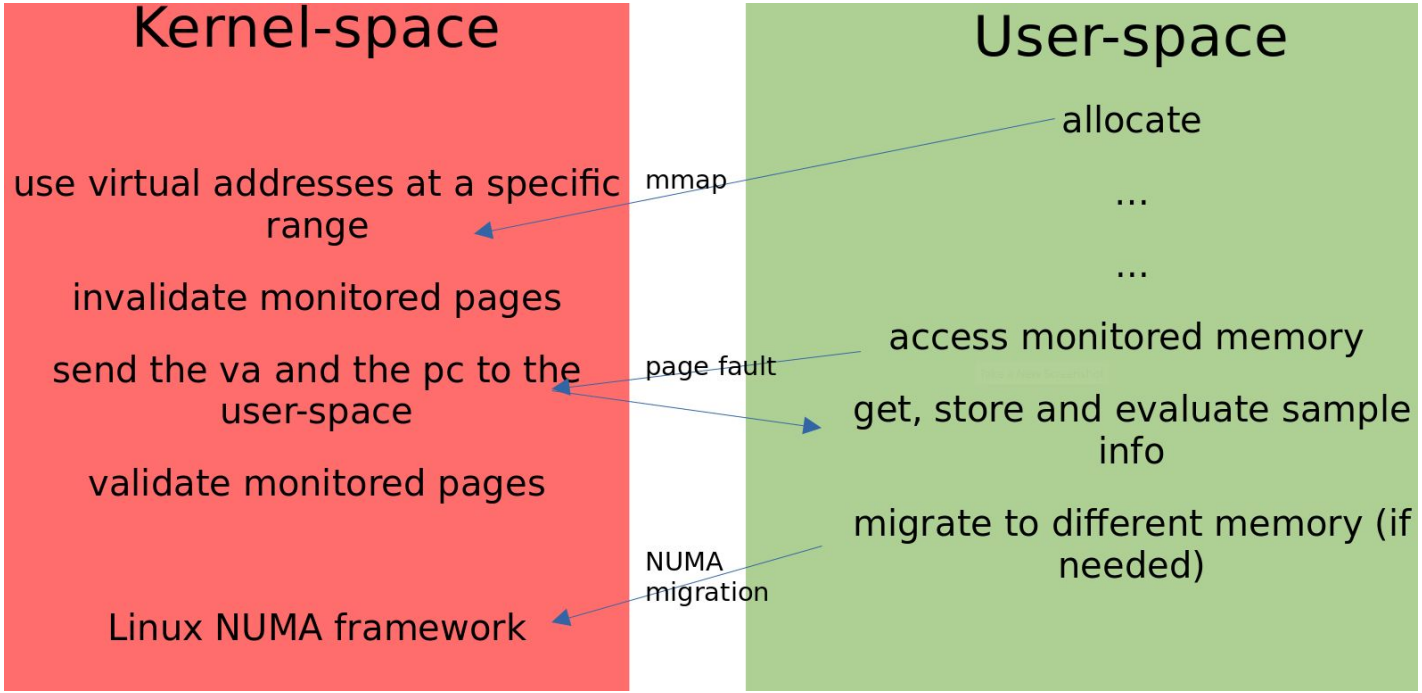
SHAMBLES

SHAMBLES overview

- Kernel space part
  - Custom mmap() flag
  - Timer invalidates part of the page table
  - Page fault handling
  - Debugfs interface
- User space part
  - Custom allocator, based on jemalloc
  - Allocator plugin, implementing policies
- No need for dedicated hardware

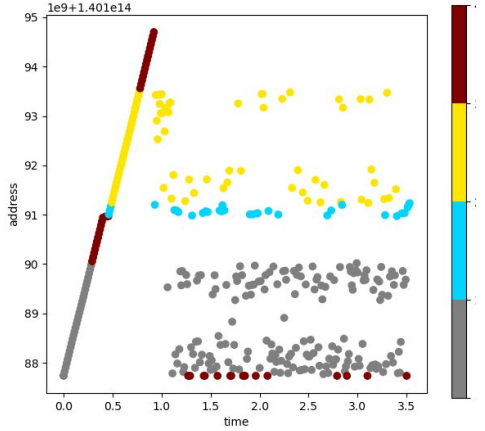


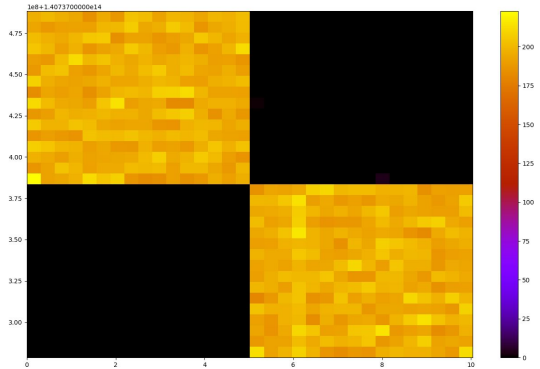
SHAMBLES Execution Flow



SHAMBLES-Based Profiling

- When we use the profiling plugin, we can then use the output to generate heatmaps and scatterplots.
- This profile can also be used to create files compatible with the HMEM advisor, developed at BSC. This is crucial for our integration effort.



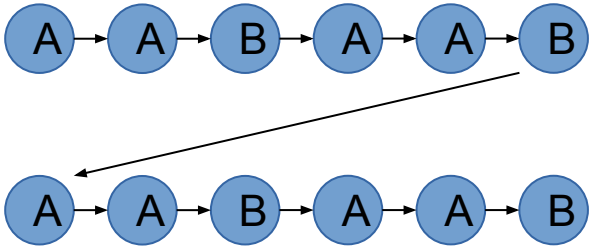


Migration Policies

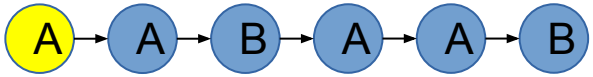
- Currently, two policies: LRU and Window.
- Both policies work on entire allocation (malloc) or parts of an allocation (chunks).
- We assume two types of memory, one of which is faster than the other.
  - The fast memory is limited and the limit is parameterisable.
  - Explicitly managed (rather than typical hierarchical arrangement)
- The **LRU policy** keeps the chunks sorted by the most recent sample.
  - The most recently sampled chunks are moved to the fast memory while the least recently use chunks go to the slow memory.
- The **Window policy** keeps track of a window of the previous samples.
  - The chunks with the most samples in the window go to the fast memory.

LRU and Window Example

- Assuming two memory areas A and B
- We access A twice then B once and so on
- We are constrained to have one of the two areas each memory type
- We are going to compare behavior of the two policies



LRU and Window Example (Cont.)



LRU

Current State: 

Hit

LRU list: 

Hits: Warming up

Misses: Warming up

Migrations: Warming up

WINDOW

Current State: 

Hit

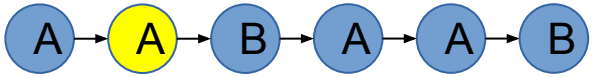
Samples in window: 

Hits: Warming up

Misses: Warming up

Migrations: Warming up

LRU and Window Example (Cont.)



LRU

Current State: 

Hit

LRU list: 

Hits: Warming up

Misses: Warming up

Migrations: Warming up

WINDOW

Current State: 

Hit

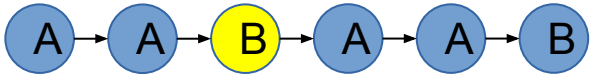
Samples in window: 

Hits: Warming up

Misses: Warming up

Migrations: Warming up

LRU and Window Example (Cont.)



LRU

Current State: 

Miss and migration

LRU list: 

Hits: Warming up

Misses: Warming up

Migrations: Warming up

WINDOW

Current State: 

Miss

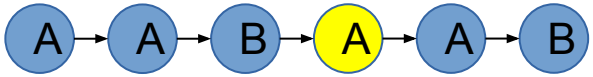
Samples in window: 

Hits: Warming up

Misses: Warming up

Migrations: Warming up

LRU and Window Example (Cont.)



LRU

Current State: 

Miss and migration

LRU list: 

Hits: 0

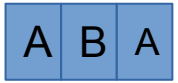
Misses: 1

Migrations: 1

WINDOW

Current State: 

Hit

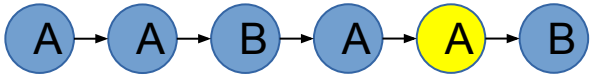
Samples in window: 

Hits: 1

Misses: 0

Migrations: 0

LRU and Window Example (Cont.)



LRU

Current State: 

Hit

LRU list: 

Hits: 1

Misses: 1

Migrations: 1

WINDOW

Current State: 

Hit

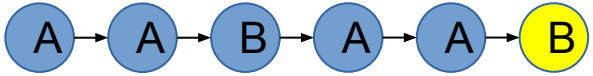
Samples in window: 

Hits: 2

Misses: 0

Migrations: 0

LRU and Window Example (Cont.)



LRU

Current State: 

Miss and migration

LRU list: 

Hits: 1

Misses: 2

Migrations: 2

WINDOW

Current State: 

Miss

Samples in window: 

Hits: 2

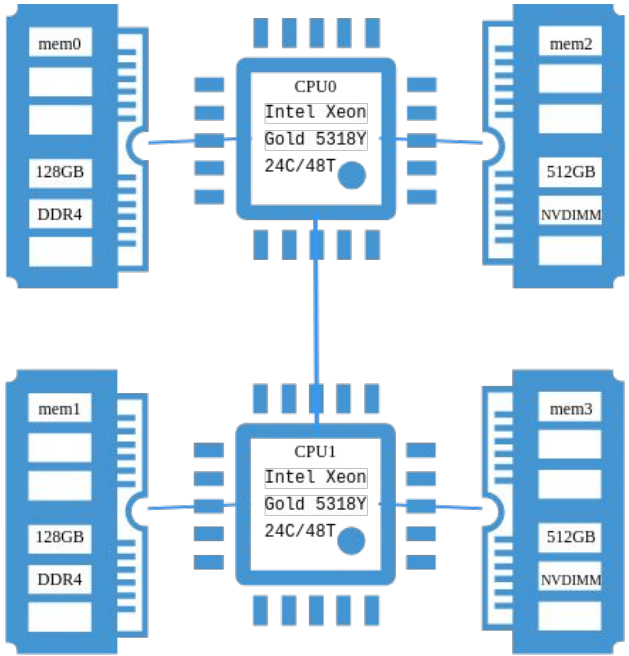
Misses: 1

Migrations: 0

Migration Evaluation Testbed

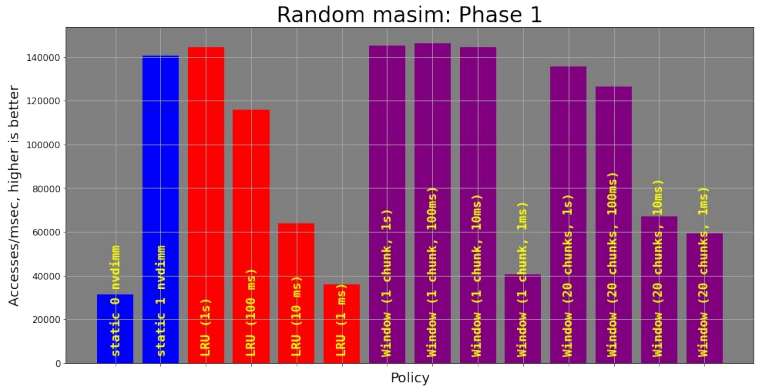
- 2 sockets Intel Xeon Gold 5318Y 24 Cores/48 Threads
- 256GB DDR
- 1024GB NVM

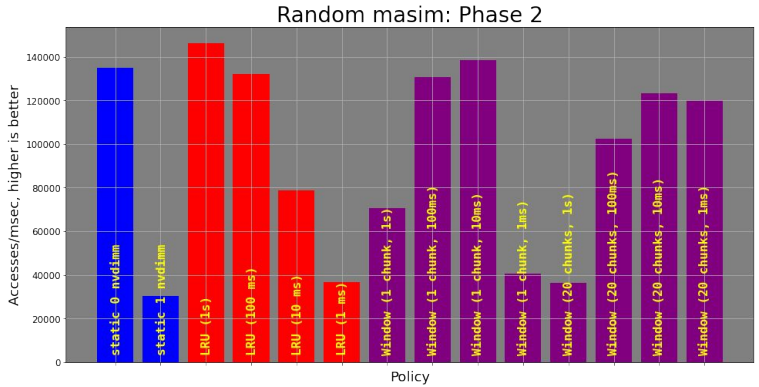
	Latency (nsec)	Bandwidth (MB/sec)
LOCAL DDR (cpu0 to mem0)	72.2	81169.8
REMOTE DDR (cpu0 to mem1)	131.6	53946.5
LOCAL NVM (cpu0 to mem2)	183.2	26566.4
REMOTE NVM (cpu0 to mem3)	183.2	2646.9



Migration Evaluation MASIM

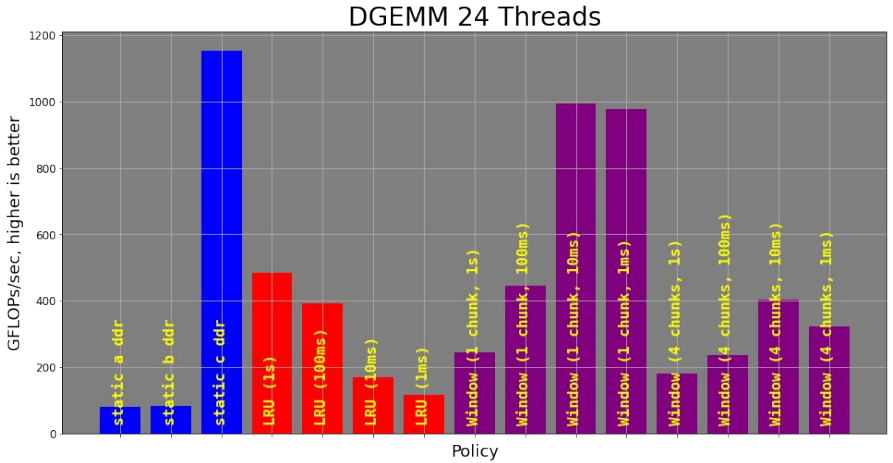
- In the masim example, we access two memory allocations in two phases.
- In each phase 99% of the accesses are in the respective area.
- Half of the memory is in the NVM and half in the DDR.
- With migrations we can perform well in both phases.





Migration Evaluation DGEMM

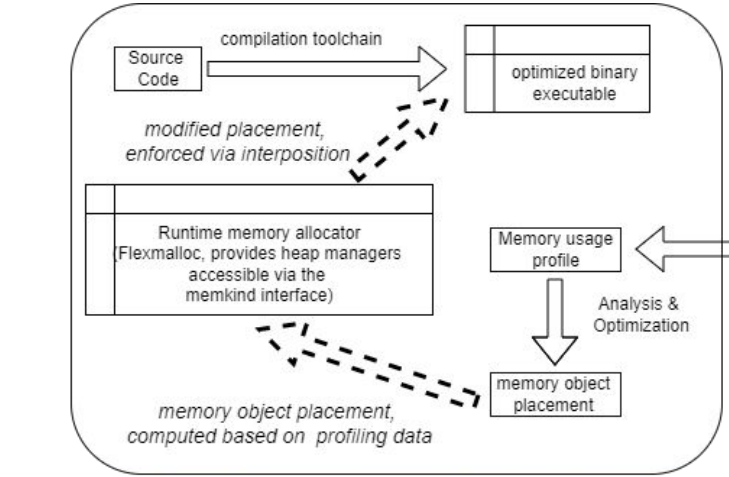
- For dgemm, we are using Intel MKL. 1/3 of the data reside in DDR memory
- Specific static allocations work better
- Some migration policies can approach the optimal static placement performance
- The LRU policy tends to have worse performance than the window based policy in both examples



Towards An Integrated Solution

Proposed Integration [1/2]

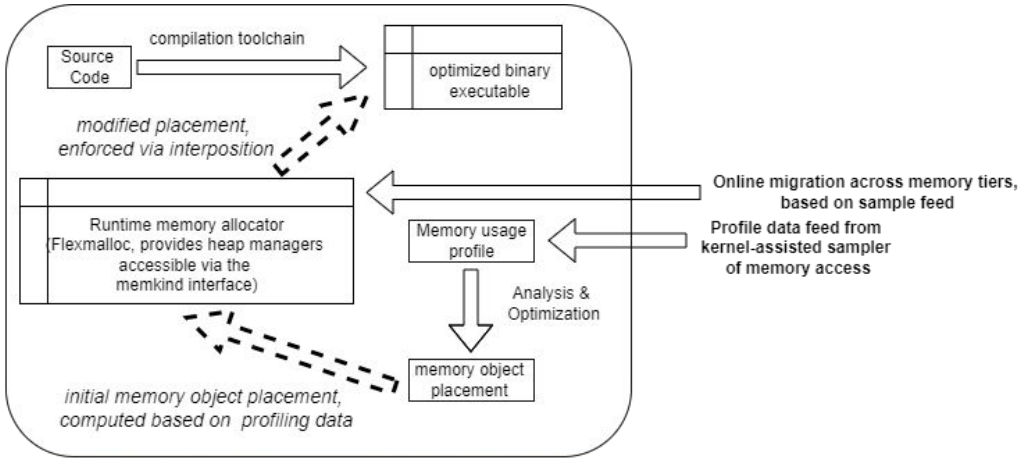
- Use a profile generated by SHAMBLES, as an alternative to the profile generated by Extrae.
- Need to post-process the data, and generate .csv and .json files compatible with the ecoHMEM Advisor.
- Migrations are beyond this phase of the integration



EcoHMEM + SHAMBLES (phase-I): optimization of memory object placement, based on profile provided by online sampler.

Proposed Integration [2/2]

- Integrate the SHAMBLES migration code with FlexMalloc.
- Use the EcoHMEM advisor for the initial memory placement.
- Afterwards, trigger migrations based on SHAMBLES policies.



EcoHMEM + SHAMBLES (phase-II): optimization of memory object placement, based on profile provided by online sampler, combined with online cross-tier migration.

Current status

- EcoHMEM
  - Enabled correct interposition for Python applications
  - On-going: Source-to-Source compiler as alternative to FlexMalloc
  - On-going: Adapting profiler to ARM architectures
- SHAMBLES
  - Heatmaps and scatterplots of profiles of various microbenchmarks
  - LRU and Window migration policies implemented
  - Performance evaluation of migration with Masim, Stream and DGEMM
- Integration
  - ecoHMEM running on FORTH infrastructure
  - SHAMBLES profiling generates .csv files compatible with ecoHMEM
  - On-going: Generation of optional .json files for ecoHMEM

Concluding Remarks

- Static, profile-based placement transparently and cheaply optimizes applications on heterogeneous infrastructures
- Applications with different memory access patterns during the execution can benefit from migrations
- When using migrations, the selection of policy is critical
- Profile-based initial placement and migrations are both needed for best performance

Source Code

- ecoHMEM
  - https://www.bsc.es/ca/research-and-development/software-and-apps/software-list/ecohmem-software-ecosystem-heterogeneous
- SHAMBLES
  - https://github.com/CARV-ICS-FORTH/shambles

Thank You
For
Your Attention

Contacts:

Marios Asiminakis: marios4@ics.forth.gr
Hatem El-Shazly: hatem.elshazly@bsc.es