

BSC **Barcelona Supercomputing Center**
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

Using Heterogeneous Memory Systems

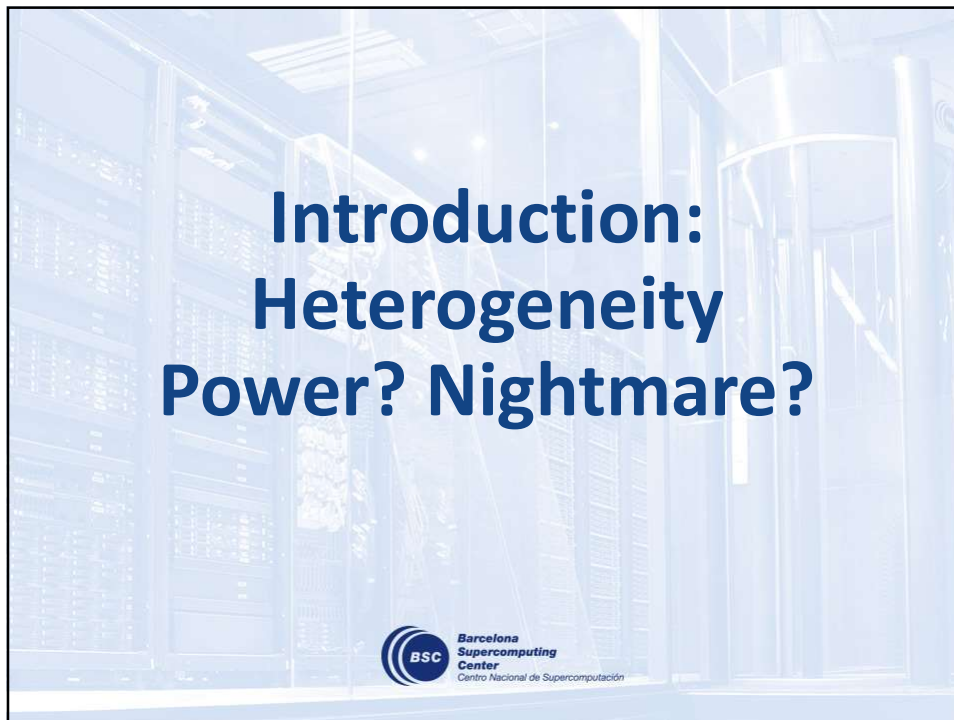
Antonio J. Peña

Barcelona

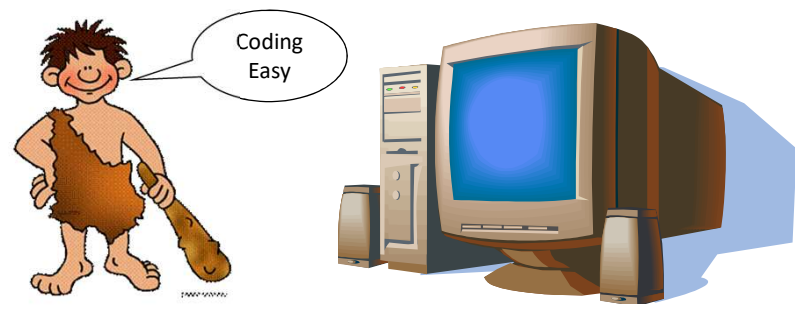
Feb. 22, 2022

Agenda

- Introduction: Heterogeneity. Power? Nightmare?
- Heterogeneous Memory Systems
 - Definition and Examples
 - Production Programming
 - Research Approaches
- Summary




Back in the Computers Stone Age...



Coding Easy

- Multi-what?
- What is a core?



BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación

4

By 1994...

Hmmm... me need more performance

What if...

Damn! Coding not so easy!

MPI!

Just wait for next processor!

BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación

5

With the end of Frequency Scaling...

Multicore

OpenMP!

DRAM MCDRAM NVRAM

NUMA

DRAM MCDRAM NVRAM

CUDA! OpenCL

NUMA

DRAM MCDRAM NVRAM

Nightmare?

BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación

6

Power? Nightmare?

- Pros

- Specialized HW
 - Awesome performance
 - More energy friendly

Exascale!

- Concerns

- Programming models?
- Programmability
- Coding productivity
- Code maintainability
- Portability
- Performance
- Performance Portability

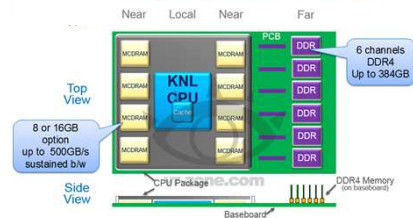


Heterogeneous Memory Systems

Motivation

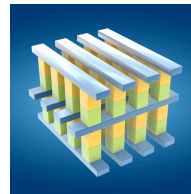
- Heterogeneity in computing explored:
 - Heterogeneous processing ✓
 - Heterogeneous memory ...
- Different memory technologies within computers already a reality
 - Scratchpad
 - Embedded processors
 - GPUs
 - High Bandwidth Memory
 - Intel KNL
 - GPUs
 - (Byte-addressable) NVRAM
 - HP's "The Machine"
 - Intel 3D XPoint
- We expect more memory heterogeneity

Knights Landing Integrated On-Package



Integrated on-package MCDRAM brings memory nearer to CPU for higher memory bandwidth and

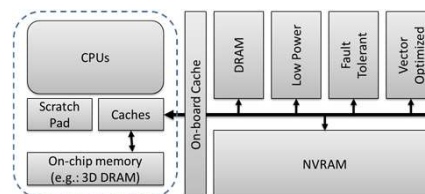
Intel KNL memory architecture



Intel 3D XPoint Technology

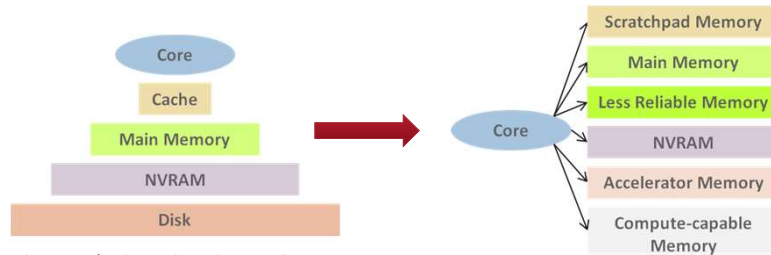
Motivation

- Different features:
 - Size, resilience, access patterns, energy, persistency...
- Examples:
 - Scratchpad:
 - Cachelike speeds, small sizes
 - Vector-specialized (e.g.: GDDR)
 - High bandwidth if cont. accesses
 - Low-power memory
 - Increased energy/speed ratio
 - ECC-enabled memory
 - Fault tolerance; speed & size ↑
 - I/O class (e.g.: NVRAM)
 - Large; reduced speeds & energy
 - Faster reading than writing



Motivation

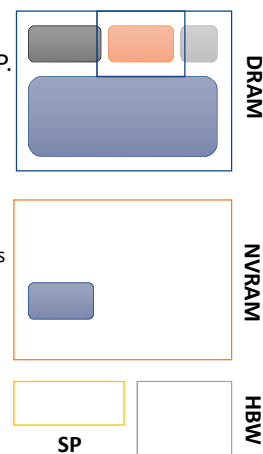
- To efficiently exploit heterogeneous memory:
 - Bring them as first-class citizens
 - Move from hierarchical to explicitly managed



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

Heterogeneous Memory Systems

- Heterogeneous memory systems
 - KNL: DRAM + MCDRAM (\uparrow BW, \uparrow Lat.) \rightarrow R.I.P.
 - Byte-addressable NVRAM (persistent)
 - HP's The Machine \rightarrow R.I.P.
 - Intel's 3D XPoint
 - Optane SSD (storage, NVMe)
 - Optane Persistent Memory (DIMM)
 - Current generation: Apache Barlow Pass
 - Also GPUs (and more with UVM)
- Our Goal
 - Assess optimal data distribution
 - Maximize performance
 - Minimize energy
 - ...



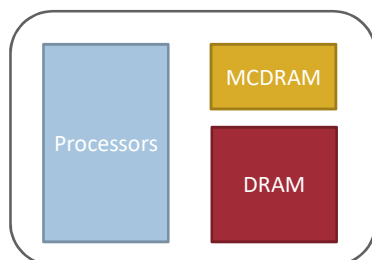
Heterogeneous Memory Systems

- Deep Memory != Heterogeneous Memory
 - Rationale: “deep” implies hierarchy (i.e., cache-like)
 - (own view – many authors don’t follow this distinction)
 - In some cases deep memory may work well (high locality)
- Heterogeneous Memory Methodologies
 - Page level
 - Leverages OS’s view
 - Can monitor hot vs. cold pages, # of allocations, total size, global status
 - Easy migrations
 - Object granularity (object: variable, static array, heap buffer, etc.)
 - Leverage object semantics
 - Usually same access pattern across entire object
 - User-friendly – user may hint / control
 - Combined?

e.g., KNL, Optane

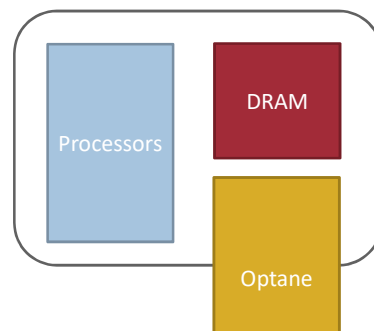
KNL

- Two levels of memory
 - Main memory (DRAM)
 - Regular DRAM latency/bandwidth
 - MCDRAM
 - High bandwidth memory: 16 GB
 - ~10%+ latency than main memory
 - Default allocation: DRAM (*slower*)



Optane

- Two levels of memory
 - Main memory (DRAM)
 - Regular DRAM latency/bandwidth
 - Intel Optane DC Persistent Memory
 - Very high capacity + persistency
 - Higher latency, but better than SSDs
 - Default allocation: DRAM (*faster*)



Memory Modes: Deep Memory

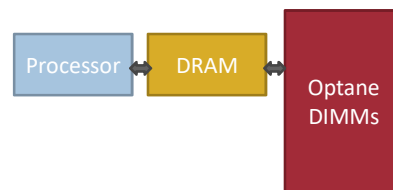
KNL: Cache Mode

- MCDRAM cache for DRAM
- Only DRAM address space
- Done in hardware (applications don't need modified)
- Misses more expensive (MCDRAM and DRAM access)



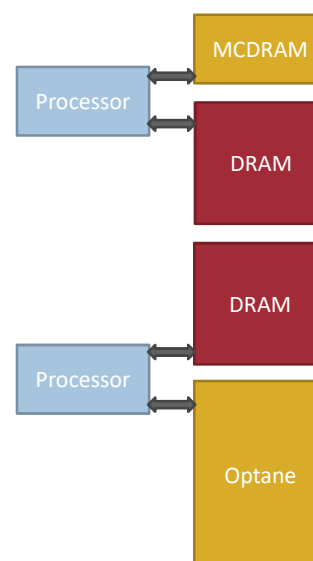
Optane: Memory Mode

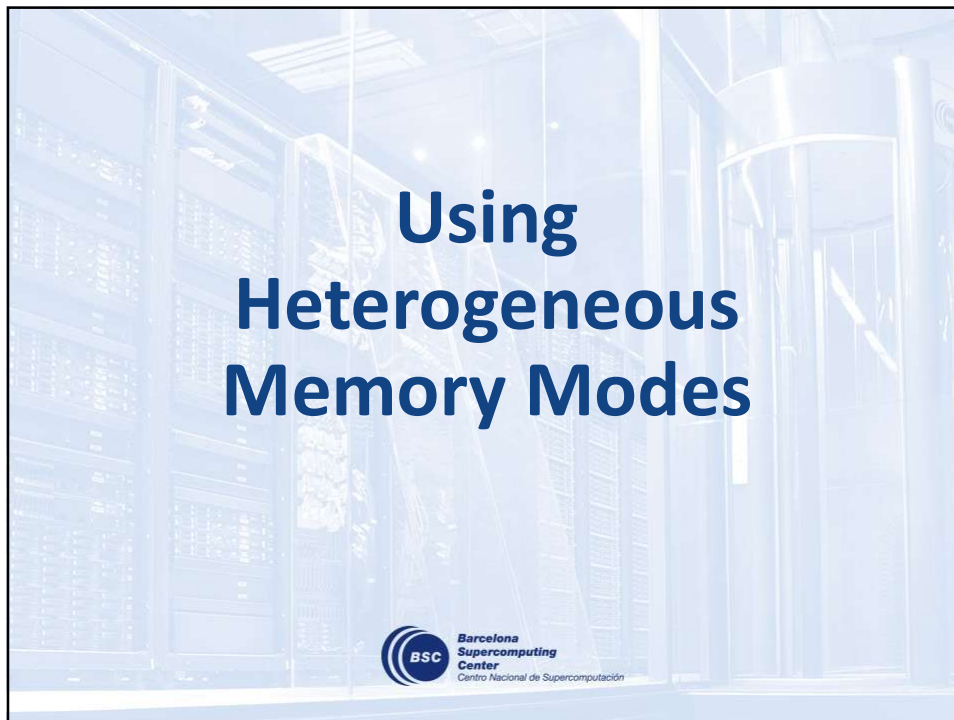
- DRAM as cache for Optane DIMMs
- Only Optane DIMMs address space
- Done in hardware (applications don't need to be modified)
- Misses more expensive (DRAM and Optane access)



Memory Modes: Heter. Memory

- KNL: Flat Mode / Optane: App Direct
- MCDRAM/Optane and DRAM are both available
- More overall memory available
- Software managed:
 - Software needs to handle itself allocation placement
 - Regular loads/stores afterwards





Using Heterogeneous Memory Modes

- Two memory spaces are available
 - By default DRAM is used
- Using MCDRAM/Optane without changing an application
 - Requires kernel support (for Optane, 5.1)
 - Set bulk memory policy
 - Preferred or enforced memory for application
 - MCDRAM/Optane exposed as **NUMA nodes**
 - Use *numactl* program
 - `numactl -H` [List of available NUMA nodes]
 - `numactl --membind 1 my_app` [Fails if it exhausts memory]
 - `numactl --preferred 1 my_app` [Falls back if it exhausts memory]
- Automatic page movement in Optane since kernel 5.5
 - Intel's HMEM DAX driver

memkind et al.

- Open source software that can manage allocations in different memory subsystems
 - <https://github.com/memkind>
- Allows defining/using memory “kinds”
- **hbwmalloc**
 - Implements memory model for KNL using memkind
 - Predefines kind options for easy KNL usage
- **autohbw**
 - “Automatic” HBM allocator

HBWMalloc

```
#include <hbwmalloc.h>
double *a = (double *) malloc(sizeof(double)*2048);
double *b = (double *) hbw_malloc(sizeof(double)*2048);
```

- Need to link to memkind library
- Automatic variables/arrays are in main memory
 - May need to restructure data to deal with this
- Also *hbw_calloc* / *hbw_free*
- If not enough memory available it will fallback (*hbw_set_policy*)
 - HBW_POLICY_BIND: Fail
 - HBW_POLICY_PREFERRED (default): Fallback to main memory
 - HBW_POLICY_INTERLEAVE: Interleave pages between MCDRAM / DDR

autohbw

- Automatic allocator, part of *memkind*
- Intercepts heap allocates and performs them on MCDRAM
- Can be tuned for different size limits
- Link to the library and set environment variables:
 - export AUTO_HBW_SIZE=1K:5K
 - export AUTO_HBW_SIZE=10K
- Needs to be linked before system libraries, e.g. LD_PRELOAD

memkind

- Malloc from the NUMA nodes of the specified “kind”
 - void *memkind_malloc(memkind_t kind, size_t size)
 - where *kind* = MEMKIND_REGULAR, MEMKIND_HBW, MEMKIND_DAX_KMEM, MEMKIND_DEFAULT...

```
#include <memkind.h>
```

```
double *a = (double *) memkind_malloc(MEMKIND_REGULAR,  
                                       sizeof(double)*2048);
```

```
double *b = (double *) memkind_malloc(MEMKIND_HBW,  
                                       sizeof(double)*2048);
```

- Also *memkind_calloc*, *memkind_realloc*, and *memkind_free*

Het. Memory Placement Mechanisms

- numactl – too coarse grained
 - Place everything in NUMA node's memory N while it fits, FCFS
- memkind allocator and derivatives (Intel)
 - User specifies (preferred) memory per allocation
 - What about statically-allocated objects (i.e., stack)?
- Kernel page movement (Intel's HMEM DAX kernel driver)
 - Reactive and coarse-grained
- OpenMP/MPI working on it

Assisting Users on Het. Memory Mode

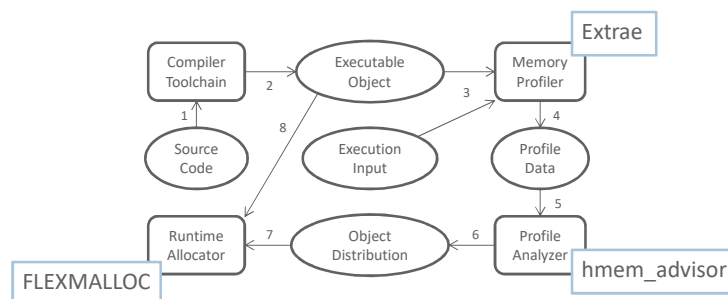
Assisting Users: Background

- Code-oriented profiler: Cost attributed to code locations
- Data-oriented profiler: Cost attributed to memory objects

for (i = 0; i < N; i++) {	5 %	a ← 5%
a[i] = b[i];	20 %	b ← 20%
c[i] = d[f(i)];	75 %	c ← 5%
}	-	d ← 70%

Methodology

- Object-differentiated data-oriented profiling + distribution algorithm (analysis):
 1. Profile to determine per-object last-level cache misses / avg. access time
 2. Assess the optimal distribution of the different objects among the memory subsystems
 - Minimize processor stall cycles



Evolved version of:

Data-Oriented Profiling (I)

EVOP: Instrumentation + Cache Simulation

Valgrind

- Generic instrumentation framework
- Ecosystem: Set of tools
 - Memcheck is just default
- Virtual machine – JIT
 - Typically overhead around 4x-5x
- Rich API to tools
 - Notify requested capabilities
 - Get debug information
 - Get information about thread status
- Intercept memory management calls
- Client request mechanism
 - Start / stop instrumentation from application's code

Callgrind

- Valgrind tool
- “Call-graph generating cache and branch prediction profiler”
- Purpose: profiling tool
 - **By source line of code**
- **Cache simulation:**
 - Cache misses
 - Cache hierarchy modeled after the host's one by default
 - Branch predictor
 - Hardware prefetcher
- Kcachegrind integration: visualization

Data-Oriented Profiling (I)

EVOP: Instrumentation + Cache Simulation

- To enable the **differentiation of memory** objects:
 - Locate the memory object comprising a given memory address
 - Store its associated access data
- Added support to be used from tools
- During the execution of a profiled application:
 - Every data access causing a last-level cache miss is checked against matching object
- To tackle profiling overhead for production-sized runs, limit to a region of interest
 - Modifying the code to be profiled
- **Output:**
 - **Per-object LL cache misses during the profiled portion of interest**

```
simulate(max_iter): /* max_iter = 2 */
CALLGRIND_START_INSTRUMENTATION
for i in 1..max_iter:
/* Simulation original code */
if i = 1:
CALLGRIND_ZERO_STATS
CALLGRIND_STOP_INSTRUMENTATION
```

A. J. Peña and P. Balaji. “A framework for tracking memory accesses in scientific applications”, in P2S2 2014

Data-Oriented Profiling (II)

Extræ: Sampling HW Counters

- Extræ is BSC's monitoring package
- Parallel programming models
 - MPI, OpenMP, pthreads, OmpSs, CUDA, OpenCL, OpenSHMEM, GASPI
- Support for in-production binaries
 - LD_PRELOAD preferred among others (DynInst, re-linking, ...)
- Sampling capabilities
 - HWC- (and alarm-) based
- Link to source code
 - Callstack at MPI, pthread, OmpSs, CUDA, OpenCL, OpenSHMEM, malloc()-related routines
 - Need debugging information to translate addresses into source-code references
- Intercept I/O calls

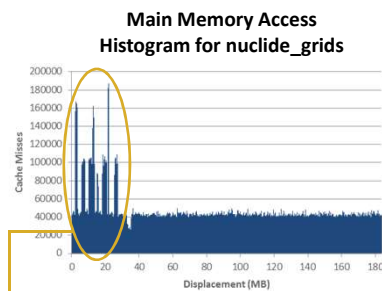
- Latest additions / related to this framework
 - Monitor malloc()-related calls (begin/end plus parameters/return)
 - Capture PEBS memory records (memory referenced plus access cost)

Tradeoff

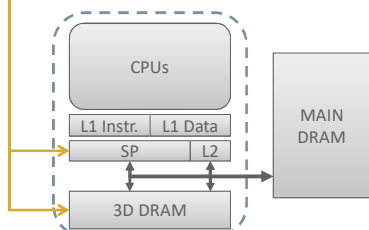
- EVOP uses instrumentation and runs on an emulator
 - Detailed but suffers high overhead
 - Simulated cache

- Extræ uses sampling & hardware counters
 - Runs on real target platform
 - Low overhead but lossy

Partitioning Approach



- **Main memory access histograms**
- energy_grid[.xs] uniform distribution
- nuclide_grids
 - >23% of LL misses within first 15%
 - Up to 80% higher than other regions
 - **Highest main memory access density**
 - Physical distribution of materials in reactor's core → diverse probabilities



```
size_t elems[2] = { ELEMS1, ELEMS - ELEMS1 };
int * split_buffer[2];
split_buffer[0] = (int *) malloc(elems[0] * sizeof(int));
split_buffer[1] = (int *) malloc(elems[1] * sizeof(int));

// Example of code needing branching
int get_elem(int **split_buffer, size_t *elems, size_t nth) {
  if(nth < elems[0]) return split_buffer[0][nth];
  else return split_buffer[1][nth-elems[0]];
}

// Example of code not needing branching
int sum(int **split_buffer, size_t *elems) {
  int sum=0;
  for(int b=0; b<2; b++)
    for(int i=0; i<elems[b]; i++)
      sum += split_buffer[b][i];
  return sum;
}
```



A. J. Peña and P. Balaji, "A data-oriented profiler to assist in data partitioning and distribution for heterogeneous memory in HPC", Parallel Computing, vol. 51, 2016 ³¹

Data-Oriented Profiling...

- Instrumenting-based solutions (slow)
 - EVOP, MemSpy, SLO, MACPO, Intel Advisor
- HW-based solutions (lossy)
 - Extrae, Sun ONE Studio, HPCToolkit, Intel Vtune Amplifier, MemAxes
- Those give data access information to developers
 - Anything more automated possible?



32

hmem_advisor (BSC)

- **Multiple knapsack** problem:
 - Knapsacks: memory subsystems
 - Knapsack capacity: memory size
 - Items: memory objects
 - Item weight: size
 - Item value: number of LL load cache misses
 - CPU stall cycles
- Not exactly a textbook problem:
 - The different knapsacks modify the value of their items:
 - Multiply cache misses by a different factor: **average latency**

Maximize the value of the content of a (set of) knapsack(s) given a set of items of different values and sizes

hmem_advisor (BSC)

- **Greedy approach solving separate 0/1 knapsack problems:**
 - Target memories in ascending order of average access cycles
 - Prioritize the placement of the most “valuable” objects in the faster memories
 - In practice not divergent from the optimal global solution
 - Removes computational complexity (0/1 knapsack is *weakly NP-hard*)
 - 4 KB page granularity

Obj #1
5k Cache Misses
Size: 6

Obj #2
10k Cache Misses
Size: 5

Obj #3
1k Cache Misses
Size: 2

```
memories.sort(key=latency, order=ascending)
for memory in memories:
    packed = knapsack(objects, memory.size)
    objects = objects - packed
```

Memory #1
Avg. Latency: 1
Size: 10

Memory #2
Avg. Latency: 10
Size: 100

Total Cost:
 $5k \times 10 + (10k + 1k) \times 1$

Methodology

Assumptions and Current Known Limitations

- Average latency estimations for the different memory subsystems
- No memory migrations nor reuse of freed space
 - (other than by the same memory object)

Sample Output

dmem_advisor, a memory object distribution tool for heterogeneous memory systems
Copyright (C) 2015, Argonne National Laboratory
Author: Antonio J. Pena <apenya@anl.gov>

```
-- SP - 8388608 bytes --
nitors [HPCCG.cpp:72] - 1 loads - 8 bytes
normr [main.cpp:170] - 1 loads - 8 bytes
t2 [HPCCG.cpp:82] - 1 loads - 8 bytes
t3 [HPCCG.cpp:82] - 1 loads - 8 bytes
rtrans [HPCCG.cpp:94] - 1 loads - 8 bytes
8 - 1 loads - 96 bytes
--
6 objects; 136 bytes (0.00162124633789%);
1080 saved

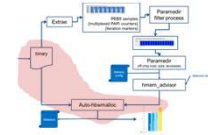
-- DRAM - 34359738368 bytes --
--
0 objects; 0 bytes (0.0%); 0 saved
```

```
-- 3D - 8589934592 bytes --
52 - 12566534 loads - 134217728 bytes
48 - 6291459 loads - 134217728 bytes
56 - 4194306 loads - 134217728 bytes
12 - 1048577 loads - 67108864 bytes
16 - 2097153 loads - 134217728 bytes
20 - 2097153 loads - 134217728 bytes
28 - 2097153 loads - 134217728 bytes
44 - 28090945 loads - 1811939328 bytes
40 - 56181888 loads - 3623878656 bytes
--
9 objects; 6308233216 bytes (73.4375%);
7453235920 saved
```

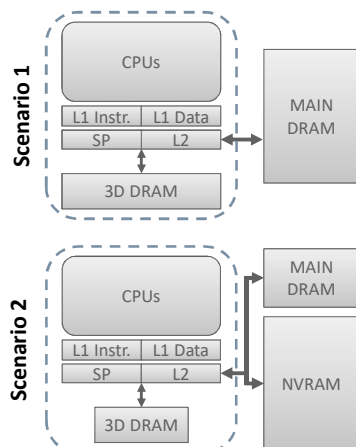
```
-- WHEREVER --
nitors [main.cpp:169] - 4 bytes
ruse [mytimer.cpp:104] - 144 bytes
--
2 objects; 148 bytes
```

FLEXMALLOC (by H. Servat, Intel)

- Runtime component
- For each malloc/realloc/free call check if it is to be substituted
- Use in-production binaries (the same as profiling)
 - Recompiled binaries may not work
 - Different debugging information, compiler parameters, ...
 - Inlining may reduce effectiveness providing false positive locations in deeper call-stacks but still work
- Only applies to dynamic allocations
 - Static allocations need to be handled manually (source code change)



Some Results Theoretical Systems (Estimated)



Overall Performance Improvement w.r.t. worst-case distribution

Hardware	Test Case	
	MiniMD	HPCCG
Scenario 1	0.0%	3.7%
Scenario 2	1,158.0%	0.1%

Some Results Real KNL

- Better or very close to baseline
- Caveats:
 - Dynamic allocation (Lulesh)
 - Will require runtime movement
 - Lack of some HW counters
 - Stack frame allocation not managed by memkind
 - We can do some assembly to place these in different mems.

Speedup of Framework w.r.t. other approaches

Code	numactl -p 1 (MCDRAM*)	Cache Mode
miniFE	1.15x	1.27x
HPCG	1.49x	1.25x
Lulesh	1.22x	0.89x
BT	1.00x	1.00x
CGPOP	0.83x	0.85x
SNAP	0.90x	0.91x
MAXW-DGTD	1.04x	0.98x
GTC-P	1.34x	1.06x

MCDRAM*: allocate as much as it fits in HBM, FCFS

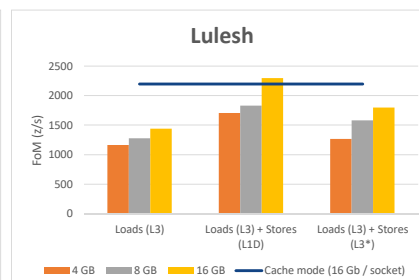
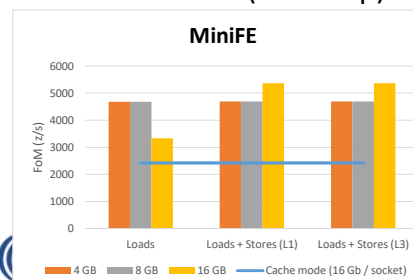
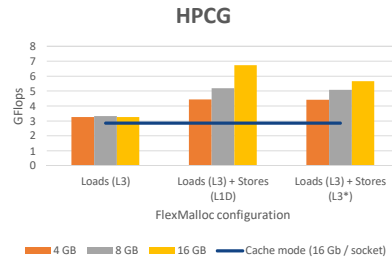
H. Servat, A. J. Peña, G. Lloret, E. Mercadal, H. C. Hoppe, and J. Labarta. "Automating the application data placement in hybrid memory systems", in IEEE Cluster 2017

Optane System Description

- Hardware
 - 2S – Intel Xeon Platinum 8260L CPU @ 2.30GHz (pre-qual), HT disabled
 - Only single socket executions
 - 2x 16 GB of DRAM (16 GB x socket)
 - 2 of 6 DIMM channels populated per socket – 1/3 of full BW
 - 12x 512 GB Optane™ DC Persistent Memory (3 TB x socket)
- Software stack
 - Fedora 27 (kernel 4.18.8-100.fc27.x86_64) – 2018ww40 BKC
 - Intel Compiler Suite 2019u3
 - Memkind checked out October'18

Some Results

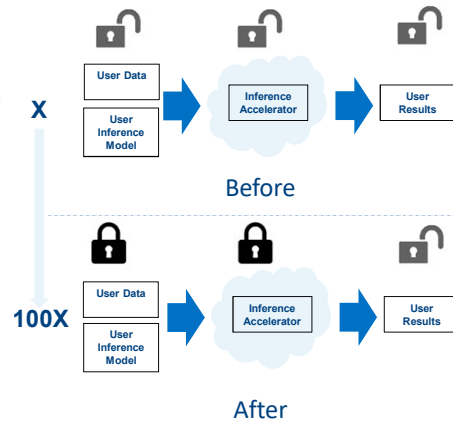
- Many cases beat cache mode in fair comparison
 - E.g., MiniFE: $\sim 100\%$ improvement w.r.t. cache mode
 - Even $\frac{1}{4}$ RAM w.r.t cache mode
- In other cases we are within negligible performance
- And few other cases require runtime actions (next step)



Highlight Use Case on Deep Memory

Homomorphic Encryption (HE)

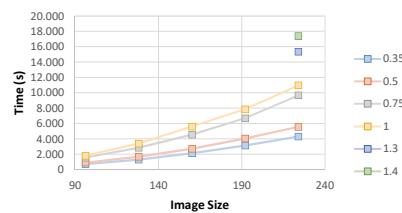
- HE allows inference on encrypted data using encrypted models.
- Allows user to submit Inference jobs externally without disclosing proprietary models & data
- Intel open sourced HE-Transformer, a library that enables HE, to accelerate research progress
- Why is HE relevant to HPC?
 - >100x memory requirements
 - >100x compute time



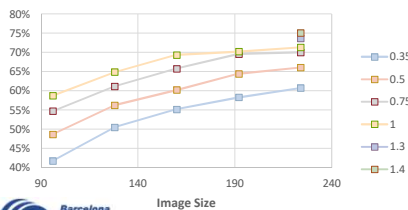
Unprecedented Accomplishment

- We have run the largest (plaintext) model to date using HE input data
- Biggest to date was MobileNetV2 at expansion factor of 0.35
- Limited by DRAM sizes
- We run MobileNetV2 at max. expansion factor (x3 previous size)
- Also ResNet-50, 1st non-toy DNN
- Thanks to Intel Optane DIMMs
- Memory mode yields high efficiency

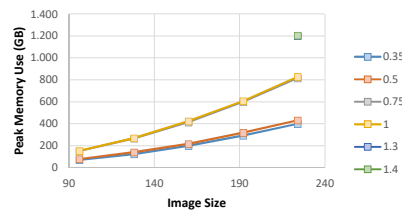
Execution Time



Top 1 Accuracy



Memory Use





Summary

- Heterogeneity is here for good and to stay
- Not only heterogeneous processing elements
 - Also memory and others
- Heterogeneous memory management APIs in production
 - Little help on deciding where to place data
- Research efforts on automatic/guided data distribution
- Some ongoing work ideas:
 - Runtime monitoring (migrations, reuse, get rid of previous profiling)
 - Seamless integration (no need for user intervention)
 - Improve profiling metrics
 - Integrate with other programming models (e.g., OpenMP)

