On the Interaction of Memory, Application and Operating System on a Heterogeneous Memory System

Steffen Christgau

Supercomputing Department
Zuse Institute Berlin

NHR PerfLab Seminar – March 16, 2021

Agenda

Application Use Case and Heterogenous Memory Systems

Memory Mode – The Transparent Way…and the impact of the OS

Making use of a Heterogenous Memory System

Agenda

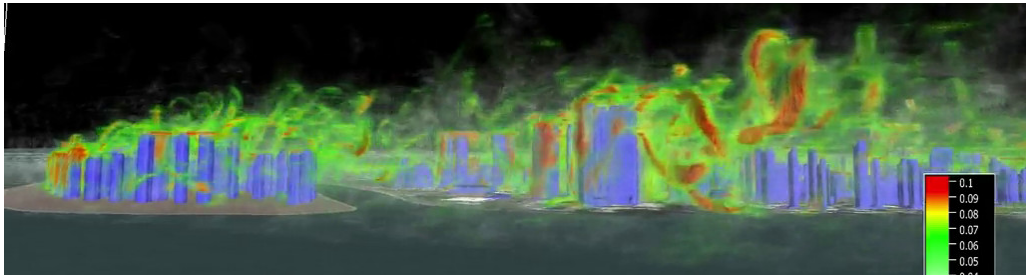
Application Use Case and Heterogenous Memory Systems

Memory Mode – The Transparent Way…and the impact of the OS

Making use of a Heterogenous Memory System

Background: PALM

- Parallelized Large-Eddy Simulation Model (PALM), University of Hannover
- well-established HLRN code, runs at ZIB in and for Berlin $\rightarrow$ city climate
- **Fortran** 2003 code, 200k+ LoC, MPI (+ OpenMP)
- 3D compute domain, **large memory footprint**: $2048 \times 2048 \times 2048 \rightarrow 1.1$ TB
- solving 3D Poisson equation with **multigrid solver** dominates runtime



Knoop, H.; Keck, M.; Raasch, S.: Urban Large-Eddy Simulation, A Parallelized Large-Eddy Simulation Model for Atmospheric and Oceanic Flows. https://doi.org/10.5446/14368

Emerging Memory Types

The era of homogenous main memory systems (single type of RAM) is over!

Heterogenous Memory Systems (HMS)

- DRAM
- High Bandwidth Memory (HBM)
- **Storage Class Memory (SCM)**: fast, large, and persistent RAM
- …

HMS already have been around:

- IBM Cell: scratch pad memory
- Intel Single-Chip Cloud Computer (SCC): Message Passing Buffer (SRAM)
- Intel Knights Landing (KNL): MCDRAM (HBM)
- FPGAs: BlockRAM + several memory interfaces

Storage Class Memory (SCM)

aliases: Non-Volatile Memory (NVM, NVRAM), Persistent Memory (PMEM)

Best of both worlds: *SCM blurs distinction between memory and storage.*[1]

technology independent attributes:

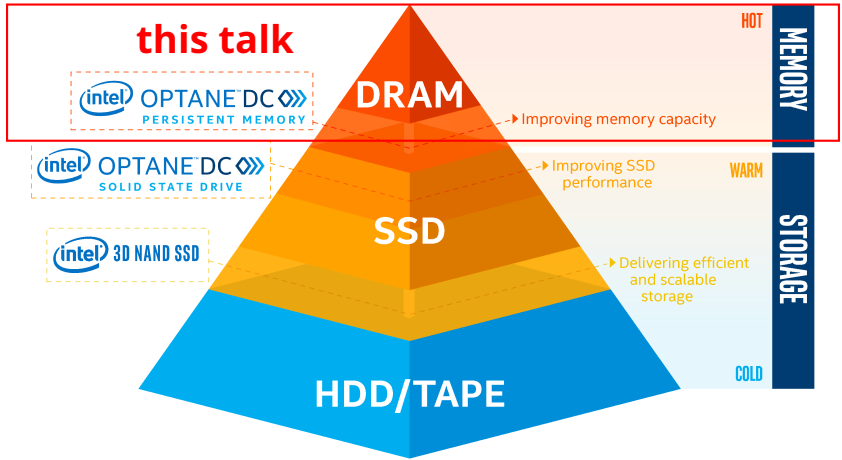
- latency/bandwidth close to DRAM → "minor" **performance** penalty (?!)
- **high capacity** → terabytes of memory, **cheaper** than DRAM w.r.t. $/Byte
- **persistence** → keep data over system reboots

- (**byte-addressable** → no need for block-wise access)

[1] Phillip Mills: Storage Class Memory - the Future of Solid State Storage, SNIA, 2009

New Level in Memory Hierarchy

SCM product by Intel: *Optane – Data Center Persistent Memory (DCPM)*



Source: https://newsroom.intel.com/editorials/re-architecting-data-center-memory-storage-hierarchy/

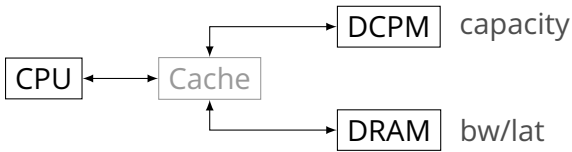
Using DCPM

two (three) operating modes – fixed after system boot

1. **Memory Mode**: transparently increase RAM capacity, DRAM as cache for DPCM

(* = "volatile")

2. **AppDirect Mode**: control location of memory (choice between DRAM or DCPM)



3. (Hybrid Mode: Not considered here)

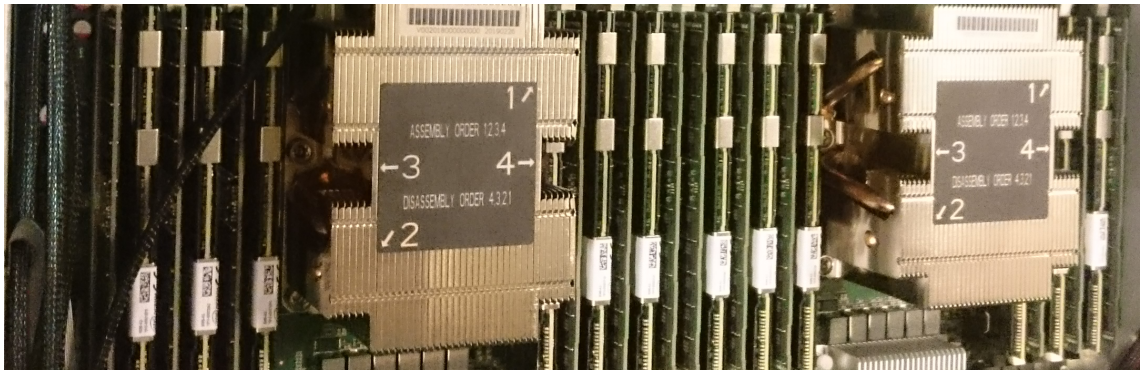
DCPM Testbed for PALM case study

single node Inspur System:

- dual Cascade Lake SP (8260**L**): 2×24 cores
- memory:
  - 384 GB DRAM
  - 3 TB DCPM **Apache Pass**

software:

- CentOS 7.8, Linux kernel 3.10
- GCC/gfortran 9.1.0
- Open MPI 4.0.3



Agenda

Application Use Case and Heterogenous Memory Systems

Memory Mode – The Transparent Way…and the impact of the OS

Making use of a Heterogenous Memory System

Memory Mode Details

- **transparent** for OS and application: large capacity available without efforts
- DRAM acts as directly-mapped cache

- hit latency = DRAM latency
- miss latency = DRAM + DPCM latency

- internal DCPM buffer size = 256 Byte, 64 Byte (CL) returned to DRAM/CPU → new **locality** level

- performance impact on PALM?

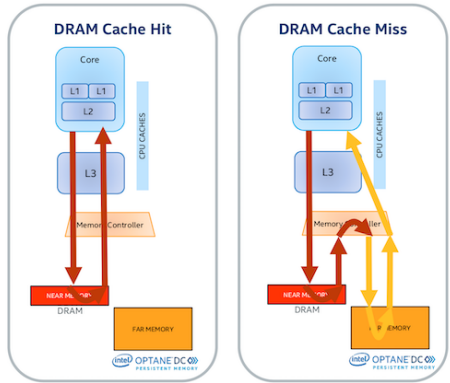
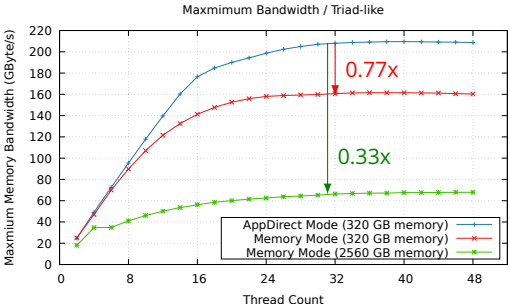
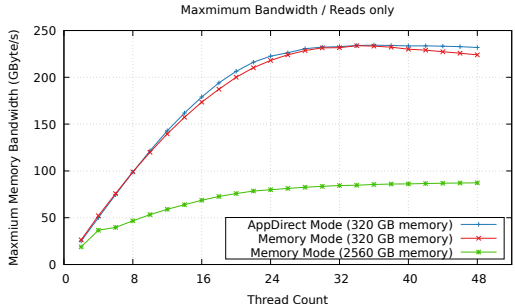


image source: Intel

Performance Assessment with MLC

- widely used Intel Memory Latency Checker
- use **triad access pattern** → MLC's closest match to PALM's mutigrid solver
- scale number of threads, distribute evenly on sockets
- measure max. bandwidth for different buffer sizes in DRAM and DCPM





Impact on PALM

- compare execution times of DRAM-only vs. Memory Mode configuration for different problem sizes → slowdown = $t_{\text{Memory Mode}} / t_{\text{DRAM mode}}$
- Multigrid Solver is the most affected by switch to Memory Mode

Class	Domain Size	Mem. Footprint	Slowdown total	Slowdown MGS
S	$32 \times 32 \times 32$	0.6 GB	**1.00**	**1.10**
W	$128 \times 128 \times 128$	1.2 GB	**1.19**	**1.13**
B	$256 \times 256 \times 256$	3.6 GB	**1.03**	**1.11**
C	$512 \times 512 \times 512$	21.2 GB	**1.12**	**1.12**
D	$1024 \times 1024 \times 1024$	152.8 GB	**1.08**	**1.07**
e	$2048 \times 2048 \times 1024$	593.3 GB	*	*
E	$2048 \times 2048 \times 2048$	1184.6 GB	*	*

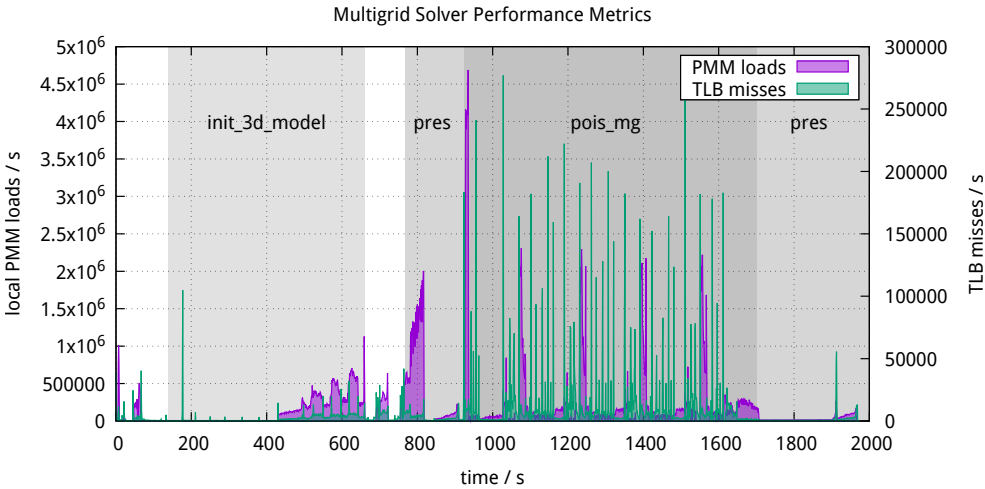
* = not runnable in DRAM mode

Memory Mode Impact on PALM

- less impacted by Memory Mode than MLC may suggested
- PALM is generally **cache-friendly** written, so is the MGS
- **higher read/write ratio** than MLC triad, like 14 reads : 1 write → example

```
!-- loops over i, j, and k
p_mg(k,j,i) = 1.0_wp / f1_mg_b(k,l) * (                &
              rho_air_mg(k,l) * ddx2_mg(l) *           &
            ( p_mg(k,j,i+1) + p_mg(k,j,i-1) )          &
            + rho_air_mg(k,l) * ddy2_mg(l) *           &
            ( p_mg(k,j+1,i) + p_mg(k,j-1,i) )          &
            + f2_mg_b(k,l) * p_mg(kp1,j,i)             &
            + f3_mg_b(k,l) * p_mg(km1,j,i)             &
            - f_mg(k,j,i)                     )
```

A View on Performance Metrics



PMM loads = `MEM_LOAD_RETIRED.LOCAL_PMM`

TLB misses = TLB load + TLB store misses

Addressing TLB Misses: Using Huge Pages in Fortran

- explore impact of page size on performance
- how to (force) use huge page usage for Fortran code? $\rightarrow$ no language support

- our approach: hook into memory management with **custom allocator**[2]
  - catch all large allocations $\geq$ 2 MB
  - use pool of preallocated huge pages to fulfill allocation request
  - simple but application-suited allocation scheme: first fit
  - **no code changes**

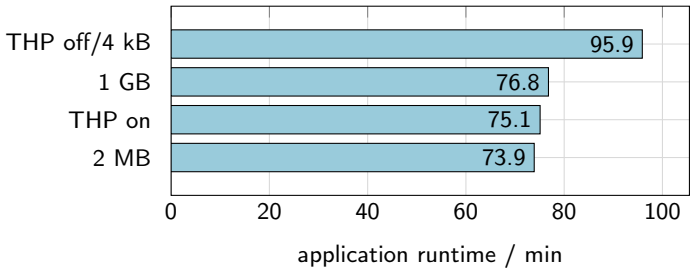
- 2 MB Linux **Transparent Huge Pages** (THP)
  - OS *may* return huge pages to application without additional efforts (if configured)
  - also **no code changes**, but **no guarantee** to get huge page

[2] https://github.com/christgau/hppmalloc

What Huge Page size is the best for PALM

run PALM in Memory Mode with different huge page sizes

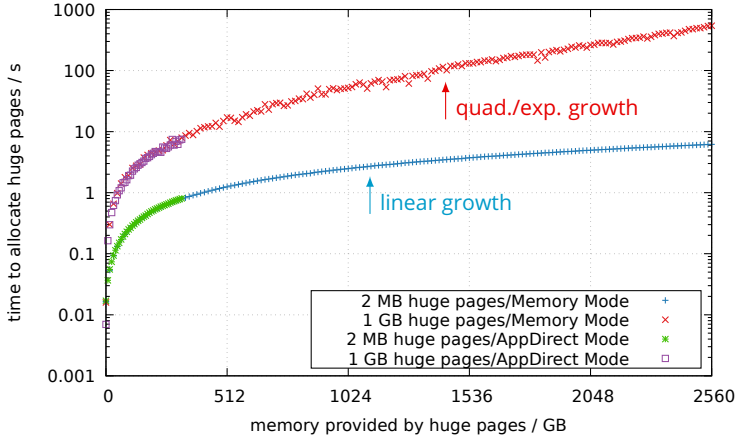
- problem class E $\rightarrow$ 2048 $\times$ 2048 $\times$ 2048 domain size, 1.2 TB memory footprint
- measure application runtime (excluding preallocation of huge page pool)



	page faults	TLB misses
THP off/4 kB	8748 M	70767 M
1 GB	38 M	257 M
THP on	61 M	512 M
2 MB	26 M	150 M

Costs of Huge Page Reservations

- measure execution time of `hugeadm --pool-pages-min=pagesize:memsize`
- fast growing time for 1 GB pages → not managed by Linux' buddy allocator



Consequences for Memory Mode

- 4 kB pages cost runtime → **use huge pages** either directly or via THP
- Why not **have 2 MB page size by default** from OS/processor?
  - 4 kB pages at Intel are from 80386/1985; systems had MBs of memory.
  - flatter page table hierarchy, if 4 kB pages dropped (have up to 5 levels now!)
- 1 GB pages have higher TLB misses and page faults than 2 MB pages
  - might be a consequence of low TLB entry number (16 for CLX)
  - have **more TLB entries for 1 GB pages** → Ice Lake (1024 shared)
  - 1 GB pages may need better support from OS (allocation time)
- better support in software stack needed?

Agenda

Application Use Case and Heterogenous Memory Systems

Memory Mode – The Transparent Way…and the impact of the OS

Making use of a Heterogenous Memory System

Smarter Use of Heterogenous Memory System

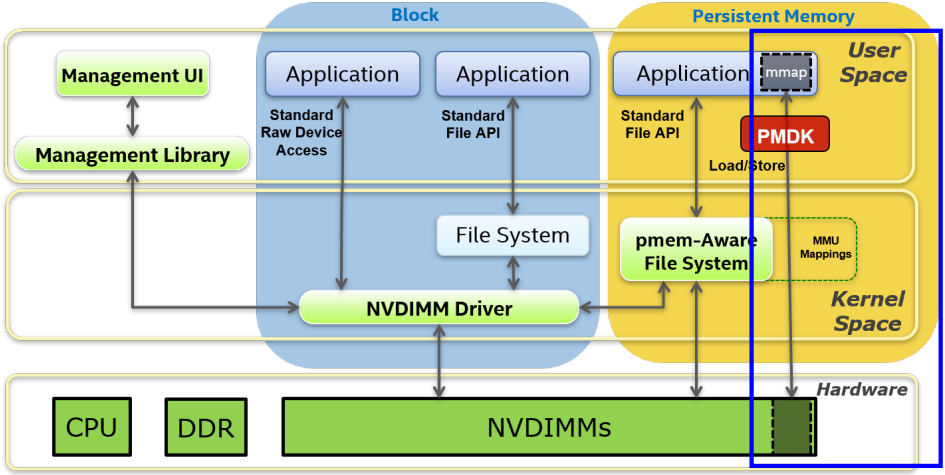
- observation: MGS most affected by Memory Mode $\rightarrow$ references to DCPM
- Can we do better than Memory Mode?

- exploit heterogenous memory system (HMS) by **application data partitioning**
- put MGS data in DRAM, compute domain parts in DCPM
  - **manual processs** to decide on memory type for application data
  - decision based on size and "hotness" (relation to MGS)

- make use of **AppDirect Mode** to allow choice of memory type/data location

Technical Background – The SNIA NVM Programming Model

A Storage Networking Industry Association Standard



source https://docs.pmem.io/persistent-memory/getting-started-guide/what-is-pmdk

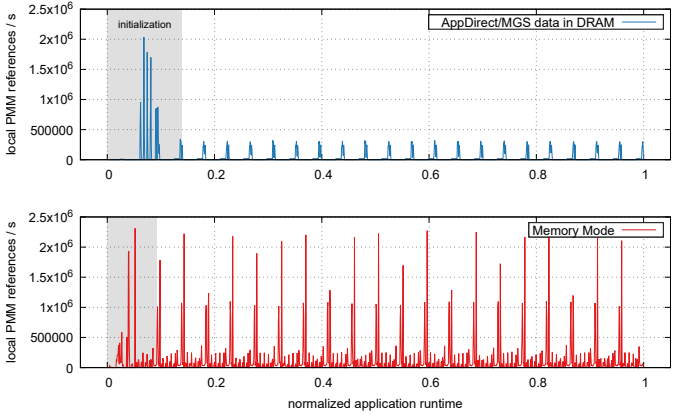
Controlling the Data Location

- challenge again: Fortran – How to control data location?
- reuse **custom memory allocator**: add function call to steer data location
  - functionality similar to `libmemkind`
  - **minimal changes**: at most two library calls per allocation

```
old_mode = hpp_set_mode(HPPA_AS_PMEM)       !-- force allocation from DCPM
ALLOCATE( d(nzb+1:nzt,nys:nyn,nxl:nxr) )    !-- unchanged allocation
hpp_set_mode(old_mode)                      !-- restore previous mode
```

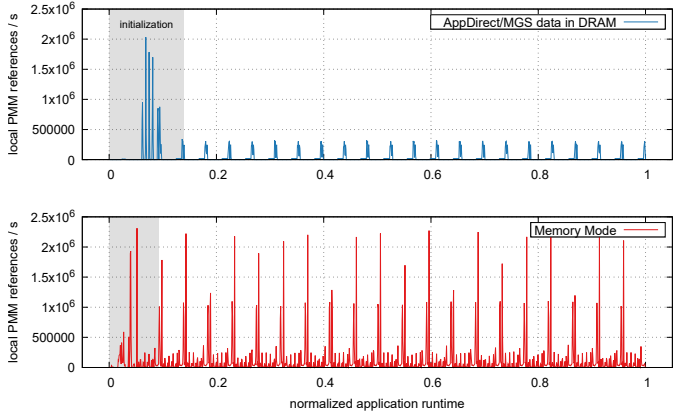
Results: The Good Parts

- able to run problem sizes impossible in DRAM
- not able to run maximum size from memory mode $\rightarrow$ DRAM capacity limit
- **Multigrid Solver faster**: speedup 1.08$\times$ for largest size (100 s faster for size *e*)
  - accesses to persistent memory eliminated



Results: The Not So Good Parts

- initialization takes more time → direct writes to DCPM
- "boundary crossing" = memory transfers DRAM→DCPM consumes (more) time
- benefits and drawbacks balance out → **no overall speedup**



Challenges – Apart from PALM

- **identify** suited memory type for application data
  - hand-crafted solution for PALM use case
  - decision criteria: size, access pattern → bandwidth vs. latency, hot/warm data …
  - automatic approaches from literature questionable
  - preferences and hints vs. strict requirements
  - tools and runtimes need improvement

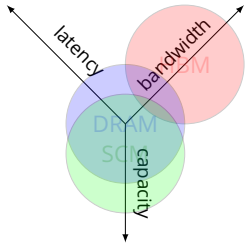


Figure: Decision space for different memory types (simplified!)

Challenges – Apart from PALM

- **identify** suited memory type for application data
  - hand-crafted solution for PALM use case
  - decision criteria: size, access pattern → bandwidth vs. latency, hot/warm data …
  - automatic approaches from literature questionable
  - preferences and hints vs. strict requirements
  - tools and runtimes need improvement

- **memory management**: allocation and data movements

- **support** along the software stack
  - OS support ✔
  - low-level libraries ✔ (libmemkind, libnuma)
  - languages ✖ → Remember `FASTMEM` in Intel Fortran for KNL? Julia? Python?
  - high-level APIs (✖) → **OpenMP**'s memory spaces → marginal compiler/RT support

Summary

- Heterogenous memory systems (HMS) require software to adaption.
- HMS usage is feasible and beneficial – even with Fortran.
- Need for better support from language and/or runtime.
- Application data partitioning for HMS is a challenge.

Thanks for your attention!
Questions?

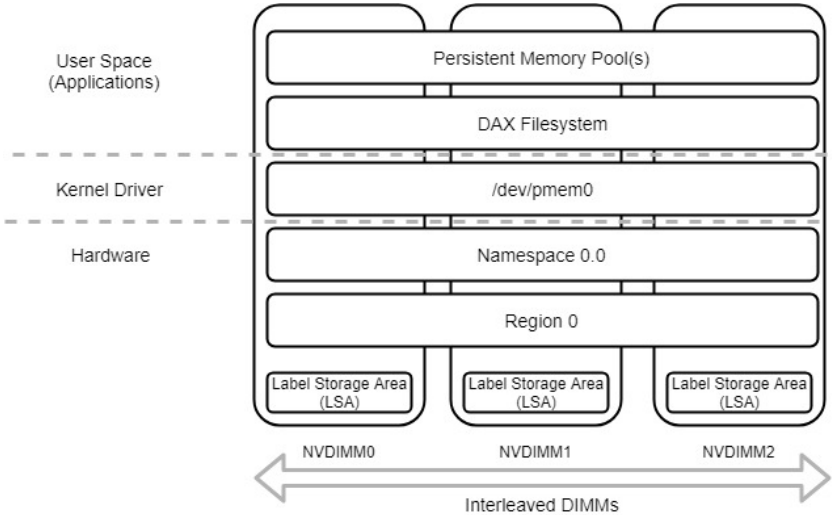
S. Christgau, T. Steinke: Leveraging a Heterogeneous Memory System for a Legacy Fortran Code: The Interplay of Storage Class Memory, DRAM and OS. 2020 Workshop on Memory Centric High Performance Computing (MCHPC)

Backup

Multigrid Solver (MGS)

- invoked per timestep on overall 3D compute domain
- MGS uses Red-Black Gauss-Seidel scheme, (optionally) cache-optimized
- configurable V or W cycle (used W cycle in study – default value)
- used stripped-down PALM code that includes MGS only (no models)
- PALM also has 3D-FFT mode for solving Poisson (not preferred by code owners)

NVDIMM configuration



Source: `https://docs.pmem.io/ndctl-user-guide/concepts`

Raw Performance Numbers

numbers from: Peng, Gokhale, Green: System Evaluation of the Intel Optane Byte-addressable NVM, 2019

Metric	DRAM	Memory Mode	AppDirect
Latency (seq. read)	79 ns	87 ns	174 ns (2.2×)
Latency (random read)	87 ns	87 ns	302 ns (3.5×)
Bandwidth (seq. read)	105 GB/s	95 GB/s (0.90×)	39 GB/s (0.37×)
Bandwidth (seq. NT-write)	82 GB/s	44 GB/s (0.53×)	12 GB/s (0.14×)
BW write:BW read	0.78×	0.46×	0.31×

note **high read-write-asymmetry** both in latency and bandwidth but also power

Addressing TLB Misses: Huge Pages

- remember CS classes: mapping of virtual to physical memory via page tables
- TLB = cache for address translation
- different page sizes and number of TLB entries on CLX-SP (testbed)

page size	L2 TLB entries	TLB reach
4 kB	1536*	6 MB
2 MB	1536*	3 GB
1 GB	16	16 GB

* = shared entries

Prepare HMS-usage of DCPM under Linux

- configure DCPM to **AppDirect Mode**
- OS and application see DRAM as usual
- create namespace in NVM once $\rightarrow$ creates block device
- format and mount device with **DAX-capable file system** (ext4, xfs)
  - kernel page cache bypass
  - memory access via mapped file (`mmap`) via load and store ops
  - libraries for DCPM usage: libmemkind, pmdk, …

NUMA Operations

- Linux 5.1 has support for using DAX namespace as NUMA node
- always avoid cross-socket traffic in Memory Mode:
  Directory updates land in Persistent Memory in the end!