Agenda

- Memory Pyramids
- NUMA Again
- Memory Tiering
- Discussion
Memory Pyramids
Memory Pyramids

Latency

Register: 0.2 ns
Cache: 1-40 ns
HBM: 50-100ns
Local CPU-DDR: 60-100ns
Remote CPU-DDR: 120-200ns
CXL-DDR: 180-260ns
NVM: 300-400ns
Over network: 2-4µs

Bandwidth

Register: +∞
Cache: 200GB/s-3TB/s
HBM: 256GB/s-2TB/s
Local CPU-DDR: 50GB/s-200GB/s
Remote CPU-DDR: 50GB/s-200GB/s
CXL-DDR: 60-120GB/s
NVM: 4-60GB/s
Over network: 10-80GB/s

Capacity

Register: KBytes
Cache: Mbytes
HBM: GBytes
Local CPU-DDR: TBytes
Remote CPU-DDR: TBytes
CXL-DDR: TBytes
NVM: PBytes
Over network: +∞
Capacity / Speed / TCO

Capacity growth faster than memory performance

Physics $\Rightarrow$ Capacity++ $\Rightarrow$ Power++

Memory large chunk of TCO

(Power: 15W / DIMM $\Rightarrow$ ~120W to 240W & upfront cost)
CPU core counts keep growing
⇒ Bandwidth per cores barely improve
⇒ Capacity per cores improve slower than needs

Density improvement coming to an end
Memory Tiering

Going toward deeper hierarchy
Performance

Same goal: keep improving compute performance

⇒ Hot data must be in the fastest memory

Through hardware == Like cache

Through software == Memory placement
Tenet

- Application can not access all its memory at the same time (not enough CPUs)
- Some data structure in an application are access more often than others
- Which data is access more often can change over application lifetime
- Some applications have predictable access pattern
- Others applications have random access pattern
- Some applications can categorize its data into buckets:
  - From most frequently accessed to least frequently accessed
Explicit vs Implicit

Explicit placement:
Application place its data to most appropriate memory

Implicit placement:
Kernel/Daemon place application memory
NUMA Again
NUMA Again

Asymmetric bandwidth & latency $\Rightarrow$ NUMA again

But even worse NUMA overlay on memory tiering
NUMA lessons

Few applications are NUMA aware
Large applications often are
Smaller applications can be through library
\((memory\ allocation)\)

⇒ Mechanism like autoNUMA
Memory Tiering
Definitions

Cold Page: a page that was not accessed in the last $N$ ms.

Hot Page: accessed more than $K$ times in the last $L$ ms.

Threshold $(N,K,L)$ can vary over time.
Overall Flow

Fast Tier Memory -> Cold Page Detection -> Slow Tier Memory

Slow Tier Memory -> Hot Page Detection -> Fast Tier Memory
Measuring Success

What metric can we use to measure success?

% of memory access to fastest memory for a thread
⇒ More access to fast memory → Better perf

Not a silver bullet
Applications with background activity dominating memory access
Kernel Components

Cold Pages detection

Hot Pages detection

Page Migration

Policy & Management
Discussion
Cold Page Detection

- LRU, MGLRU
  - Good candidates \(\Rightarrow\) migrate before reclaim
- Access Bit
  - Clear Access Bit \(\Rightarrow\) No access over N ms
- DAMON
DAMON

DAMON == Data MONitor access

Is DAMON good enough?
Better to have cold page monitoring != hot page
Cold page do not need region
Region can hide cold pages
Hot Page Detection

Hot Page
⇒ Many access over short periods of time
⇒ Software monitoring need high frequency sampling
⇒ Large overhead to do software sampling

We want hardware for hot page detection
● Heatmap
● N most recently used address
Page Migration

Existing kernel API good enough?
● move_pages()
● migrate_pages()

Do we want a more asynchronous API?

Something like io_uring but for memory?
● Memory Migration
● Memory reclaim
● Virtual Address Space manipulation
Policy & Management:

- In kernel? Like LRU
  - One solution fits all?
- In userspace?
  - Different strategy per application groups

memcg point toward userspace being a better places
Configuration

How do we want to configure memory tiers?

- sysfs API
- Disconnect from NUMA distance

Can help bridge kernel & userspace management

See [RFC: Memory Tiering Kernel Interfaces (v4)](http://example.com/rfc-memory-tiering-kernel-interfaces-v4)