Optimizing for Intel's Knights Landing - Memory Hierarchy

Hal Finkel
hfinkel@anl.gov
2017-07-26
Caches

- KNL cores are paired into a “tile”, which share an 1 MB L2 cache
- L2 cache can deliver 1 read cache line and 0.5 write cache lines per cycle
- Each core has its own 32 KB L1 I-cache and 32 KB L1 data cache
- The cache is “writeback” - the processor reads a cache line to write to it
- Each cache line is 64 bytes (the size of one 512-bit vector)

http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7453080
Memory Requests

CPU Load Pipeline

L1 Data Cache

Cache line containing x

L2 Cache

Cache line containing x

Memory Controller (request coalescing buffer)

(MC)DRAM

Always fetch whole cache lines (arrange your data accordingly)

GPUs have coalescing buffers here too, and they can afford to wait longer!
Tiles are arranged on a mesh
L2 caches are coherent, so we need tag directories to keep track of which tile owns which cache lines
How the cache lines are mapped to tag directories has three modes (selected at boot time): all-to-all, quadrant, and sub-NUMA clustering
Address uniformly hashed across all distributed directories

No affinity between Tile, Directory and Memory

Most general mode. Lower performance than other modes.

Typical Read L2 miss
1. L2 miss encountered
2. Send request to the distributed directory
3. Miss in the directory. Forward to memory
4. Memory sends the data to the requestor
KNL Quadrant Mode

Chip divided into four virtual Quadrants

Address hashed to a Directory in the same quadrant as the Memory

Affinity between the Directory and Memory

Lower latency and higher BW than all-to-all. SW Transparent.

1) L2 miss, 2) Directory access, 3) Memory access, 4) Data return

Avinash Sodani CGO PPoPP HPCA Keynote 2016
Note, however, that quadrants are not symmetric!

Each Quadrant (Cluster) exposed as a separate NUMA domain to OS.

Affinity between Tile, Directory and Memory

Local communication. Lowest latency of all modes.

SW needs to NUMA optimize to get benefit.

Run one MPI rank per quadrant?

1) L2 miss, 2) Directory access, 3) Memory access, 4) Data return
HBM Modes

Three Modes. Selected at boot

Cache Mode
- SW-Transparent, Mem-side cache
- Direct mapped. 64B lines.
- Tags part of line
- Covers whole DDR range

Flat Mode
- MCDRAM as regular memory
- SW-Managed
- Same address space

Hybrid Mode
- Part cache, Part memory
- 25% or 50% cache
- Benefits of both
How Flat Mode Looks

MCDRAM exposed as a separate NUMA node

Memory allocated in DDR by default → Keeps non-critical data out of MCDRAM. Apps explicitly allocate critical data in MCDRAM. Using two methods:

- “Fast Malloc” functions in High BW library (https://github.com/memkind/memkind)
  - Built on top to existing libnuma API
- “FASTMEM” Compiler Annotation for Intel Fortran

Flat MCDRAM with existing NUMA support in Legacy OS

Avinash Sodani CGO PPoPP HPCA Keynote 2016
Flat Mode Memory Management

**C/C++** (*https://github.com/memkind*)

Allocate into DDR

```c
float *fv;
fv = (float *)malloc(sizeof(float) * 100);
```

Allocate into MCDRAM

```c
float *fv;
fv = (float *)hbw_malloc(sizeof(float) * 100);
```

**Intel Fortran**

Allocate into MCDRAM

```fortran
!DEC$ ATTRIBUTES Fastmem :: A

NSIZE=1024

allocate array 'A' from MCDRAM

ALLOCATE (A(1:NSIZE))
```

Avinash Sodani CGO PPoPP HPCA Keynote 2016
Requesting Different Memory Modes

On ALCF's Theta system:

qsub -q default -n <N> -t 60 --attrs mcdram=cache:numa=quad ./myscript.sh

where the options for mcdram and numa are

- mcdram: cache, flat, split, equal
- numa: a2a, quad, hemi, snc2, snc4

For more information, see:
- http://www.alcf.anl.gov/user-guides/cobalt-job-control-xc40
- https://www.alcf.anl.gov/user-guides/xc40-memory-modes

On NERSC's Cori system, in your batch script place:

#SBATCH -C knl,quad,flat or #SBATCH -C knl,quad,cache

For more information, see:
Where should memory go by default?

Bind to MCDRAM by default:

numactl -m 1 ./app

Prefer MCDRAM, but fall back to DDR if necessary:

numactl -p 1 ./app

If you bind to MCDRAM by default, how can you explicitly allocate in DDR?

#include <numa.h>
void *numa_alloc_onnode(size_t size, int node);

Note that node = 0 is DDR and node = 1 is MCDRAM.

If you're allocating in DDR by default (which is the default), you can use libnuma or hbw_malloc to allocate in MCDRAM.

You may also find other parts of libnuma useful, see: http://man7.org/linux/man-pages/man3/numa.3.html
SNC2/SNC4 are different...

- [https://colfaxresearch.com/knl-numa/](https://colfaxresearch.com/knl-numa/)
- [https://colfaxresearch.com/knl-mcdram/](https://colfaxresearch.com/knl-mcdram/)

Here, you might use: `numactl -m 4,5,6,7 ./app`
SNC2/SNC4 and nested OpenMP...

Listing 2: Example of nested OpenMP parallelism with 4 teams of 64 threads.

https://colfaxresearch.com/knl-numa/
SNC2/SNC4 and nested OpenMP...

Be aware of the "first touch" locality policy…

Listing 8: Parallel first touch for NUMA locality.

```
int main() {
    float *A = new float[N];
    #pragma omp parallel for
    for (int i=0; i < N; i++)
    A[i] = 0.0f;
}
```

https://colfaxresearch.com/knl-numa/

- Using the memory from subsequent parallel regions will be slower if you remove the parallel for from the initialization.
- However, don't forget about libnuma!
How fast is fast?

Table 3 - STREAM Triad Performance

<table>
<thead>
<tr>
<th>Mode</th>
<th>Size</th>
<th>Triad</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat - MCDRAM</td>
<td>7.5 GB</td>
<td>485 GB/s</td>
</tr>
<tr>
<td>Flat - DRAM</td>
<td>7.5 GB</td>
<td>88 GB/s</td>
</tr>
<tr>
<td>Cache <em>(Fits in MCDRAM)</em></td>
<td>7.5 GB</td>
<td>352 GB/s</td>
</tr>
<tr>
<td>Cache <em>(From DDR)</em></td>
<td>120.0 GB</td>
<td>59 GB/s</td>
</tr>
</tbody>
</table>
How fast is fast?

(b) Filling Tiles.

Figure 9: Memory bandwidth achieved with our triad benchmark in SNC4-flat mode.

Higher bandwidth, but also higher latency?

Figure 4: Latency of cache line transfers between core 0 and every other core in SNC4-flat mode for M, E and I states.

See also: https://arxiv.org/pdf/1704.08273.pdf
For streaming...

```c
#include <stdio.h>

void foo1(float *a, float *b, float *c, int n) {
    int i;
    #pragma vector aligned nontemporal
    for (i=0; i<n; i++) {
        a[i] *= b[i] + c[i];
    }
}
```

Non-temporal loads/stores can help, especially in cache mode.
Contact support

If something is wrong, or you need help for any other reason, contact the facility's support service:

Our people set us apart

(system reservations for debugging are often possible, just ask!)

These people are not scary!
Some final advice...

Don’t guess! Profile! Your performance bottlenecks might be very different on different systems.

And don’t be afraid to ask questions...

Any questions?

› ALCF is supported by DOE/SC under contract DE-AC02-06CH11357
Some Things to Try

Start with the STREAM benchmark and associated tuning hints…


And then also:

• Try using nontemporal memory accesses (e.g.,

• Try accessing random memory indices to be sensitive to latency (if you need a simple rand, see:
  https://github.com/cloudius-systems/musl/blob/master/src/prng/rand.c)