Intel MIC Programming Workshop:
KNL MCDRAM and KNC Offloading
Dr. Volker Weinberg (LRZ)
June, 26-28, 2017, LRZ
KNL Cluster and Memory Modes
MCDRAM

with material from Intel, John Cazes et al. (TACC) and Adrian Jackson (EPCC)
Cores are grouped in pairs (tiles)
- 36 possible tiles
- 2D mesh interconnect
- 2 DDR memory controllers
  - 6 channels DDR4
  - Up to 90 GB/s
- 16 GB MCDRAM
  - Up to 475 GB/s
● Basic unit for replication
● Each tile consists of 2 cores, 2 vector-processing units (VPU) per core, a 1 MB L2 Cache shared between the 2 cores
● CHA (caching/home agent)
  – Serves as the point where the tile connects to the mesh
  – Holds a portion of the distributed tag directory structure
2D Mesh Interconnect

- Tiles are connected by a cache-coherent, **2D mesh interconnect**
- Provides a more scalable way to connect the tiles by providing higher bandwidth and lower latency compared to KNC **1D ring interconnect**
- **MESIF** (**M**odified, **E**xclusive, **S**hared, **I**nvalid, **F**orward) cache-coherent protocol
- Cache lines present in L2 caches are tracked using a **distributed tag directory structure**
- Around **700 GB/s total aggregate bandwidth**
- Mesh is organized into rows and columns of half rings that fold upon themselves at the endpoints
- Mesh enforces a **YX routing rule**
- Mesh at fixed frequency of 1.7 GHz
- Single hop: X-direction 2 clocks, Y-direction: 1 clock
KNL Memory Architecture

- **2 Memory Types**
  - MCDRAM (16 GB)
  - DDR4 (96 GB)

- **3 Memory Modes**
  - Cache
  - Flat
  - Hybrid
    - 25% (4 GB)
    - 50% (8 GB)
    - 75% (12 GB)
KNL Memory: Overview

- **Memory hierarchy on KNL:**
  - DDR4 (96 GB)
  - MCDRAM (16 GB)
  - Tile L2 (1 MB)
  - Core L1 (32 KB)

- **Tile:** set of 2 cores sharing a 1MB L2 cache and connectivity on the mesh

- **Quadrant:** virtual concept, not a hardware property. Way to divide the tiles at a logical level.

- **Tag Directory:** tracks cache line locations in all L2 caches. It provides the block of data or (if not available in L2) a memory address to the memory controller.
• **High-bandwidth** memory integrated on-package
• **8 MCDRAM** devices on KNL, each with 2 GB capacity -> total 16 GB
• Connected to EDC memory controller via **proprietary on-package I/O: OPIO**
• Each device has a **separate read and write bus** connecting it to its EDC
• **Aggregate Stream Triads Bandwidth** for the 8 MCDRAMS is **over 450 GB/s**
• Slighter higher latency than main memory (~10% slower)
KNL Memory: DDR4

- **High-capacity** memory **off-package**
- KNL has direct access to all of main memory
- **2 DDR4 memory controllers** on opposite sides of the chip, each controlling **3 DDR4 channels**
- Maximum total capacity is 384 GB
- **Aggregate Stream Triads Bandwidth** from all 6 DDR4 channels is around **90 GB/s**
KNL Memory Modes

- **Cache**:  
  - MCDRAM serves as cache for transactions to DDR4 memory  
  - Direct-mapped memory-side cache with 64-byte cache-lines  
  - Inclusive of all modified lines in L2 cache  
  - Completely transparent to the user

- **Flat**:  
  - Flat address space  
  - Different NUMA nodes for DDR4 and MCDRAM  
  - `numactl` or `memkind` library can be used for allocation

- **Hybrid**:  
  - 20% / 50% / 75% of MCDRAM set up as cache  
  - Potentially useful for some applications
KNL Memory Modes

(a) Cache Mode
(b) Flat Mode
(c) Hybrid Mode
## Memory Modes: Comparison

### Memory Mode Comparison Table

<table>
<thead>
<tr>
<th>Memory Mode</th>
<th>MCDRAM</th>
<th>DDR4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat</td>
<td>100% Memory</td>
<td>100% Memory</td>
</tr>
<tr>
<td>Hybrid 25%</td>
<td>75% Memory</td>
<td>100% Memory</td>
</tr>
<tr>
<td>Hybrid 50%</td>
<td>50% Memory</td>
<td>100% Memory</td>
</tr>
<tr>
<td>Hybrid 75%</td>
<td>25% Memory</td>
<td>100% Memory</td>
</tr>
<tr>
<td>Cache</td>
<td>0% Memory</td>
<td>100% Memory</td>
</tr>
</tbody>
</table>
Cluster Modes

- Cluster Modes modify the distance that L2 coherency traffic flows go through the mesh
- 5 Cluster Modes supported:
  - All-to-all
  - Quadrant / Hemisphere
  - 2 Sub-NUMA Cluster modes: SNC-4 / SNC-2
- Regardless of the cluster mode selected, all memory (all MCDRAM and all DDR4) is available to all cores, and all memory is fully cache-coherent.
- What differs between the modes is whether the view of MCDRAM or DDR is UMA (Uniform Memory Access) or NUMA.
Cluster Modes: Overview

Cluster modes modify the distance that coherency traffic flows through mesh!

(a) All-to-All (no communication localized)
(b) Quadrant (some communication localized)
(c) Sub-NUMA-4 (SNC-4) (all communication localized)
Cluster Modes

- A **quadrant** is a virtual concept and not a hardware property. Divides tiles into 4 groups at a logical level.
- A **tag directory** is used to track L2 cache line locations and status (which tile and if valid).
- Tag directory is distributed over tiles
  - Each directory component is responsible for an exclusive portion of address space
  - Directory indicates where cache line is: a certain tile’s L2 cache or in memory
Cluster Modes

- A tile’s L2 cache can hold any data
- Tag directory tracks if data is in L2 and which tile’s L2 has data
- Tag directory is distributed across all tiles
  - Each tile has an equal portion of the address space
  - Portion of tag directory in a tile not related to L2 cache in that tile
- Every tile has a Caching-Home Agent (CHA)
  - services queries about it’s portion of the tag directory
Cluster-Modes: UMA vs. NUMA

- **Quadrant-Mode:**
  - Each memory type is **UMA**
  - latency from any given core to any memory location in the same memory type (MCDRAM or DDR) is essentially the same

- **SNC-4 (2) Mode:**
  - Each memory type is **NUMA**
  - Cores and memory divided into 4 quadrants (2 halves) with lower latency for *near memory accesses* (within the same quadrant (half)) and higher latency for *far memory accesses* (within a different quadrant (half))
Cluster-Modes: NUMA Domains

- **Flat-All2All/Quadrant/Hemisphere Mode:**
  - 1 DDR NUMA Domain
  - 1 MCDRAM NUMA Domain

- **Flat-SNC-2:**
  - 2 DDR NUMA Domain
  - 2 MCDRAM NUMA Domain

- **Flat-SNC-4:**
  - 4 DDR NUMA Domain
  - 4 MCDRAM NUMA Domain

- **Cache Mode:**
  - 1 DDR NUMA Domain
  - 0 MCDRAM NUMA Domain

Memory interleaving (technique to spread out consecutive memory access across multiple memory channels in order to parallelise the accesses) differs among the various modes.
Cache Coherency Protocol

- For memory loads/stores:
  - Core (requestor) looks in local L2 cache
- If not there it queries Distributed Tag Directory (DTD) for it:
  - Sends message to tile (CHA) containing DTD entry for that memory address (tag owner) to check if any other tile on the chip has that address in its caches
- If it’s not in any cache then data fetched from memory
  - DTD updates with requestor information
- If it’s in a tile’s L2 cache then:
  - Tag owner sends message to tile where data is (resident)
  - Resident sends data to requestor
Cluster Modes: All to All

- **No affinity between tile and tag directory:** When there is an L2 miss in a tile the directory tag may be anywhere on the chip
- **No affinity between tag directory and memory:** Data associated to a directory tag may be anywhere on the chip

1. L2 Miss: data not in local L2
2. Directory access: look for tag in directory
3. Memory access: look for data in memory
4. Data: send data to original tile
Cluster Modes: Quadrant

- **No affinity** between tile and tag directory:
  When there is an L2 miss in a tile the directory tag may be anywhere on the chip

- **Affinity** between tag directory and memory:
  Data associated to a directory tag will be in the same quadrant that the directory tag is located

1. L2 Miss: data not in local L2
2. Directory access: look for tag in directory
3. Memory access: look for data in memory
4. Data: send data to original tile
Cluster Modes: SNC-4

1. L2 Miss: data not in local L2
2. Directory access: look for tag in directory
3. Memory access: look for data in memory
4. Data: send data to original tile

- **Affinity between tile and tag directory:**
  When there is an L2 miss in a tile the directory tag will be in the same quadrant

- **Affinity between tag directory and memory:**
  Data associated to a directory tag will be in the same quadrant that the directory tag is located
Cluster Modes: Some more remarks

- **All-to-All Mode:**
  - Most general mode. Lower performance than other modes, ignore
  - Only mode that can be used when DDR DIMMS have not identical capacity

- **Quadrant-Mode:**
  - Lower latency and higher bandwidth than all-to-all.
  - Will always give reasonable performance
  - SW transparent, no special NUMA optimisation
    - 1 NUMA region for MCDRAM
    - 1 NUMA region for DDR
  - Specially well suited for MPI applications with 1 rank per KNL

- **SNC-4:**
  - Each Quadrant exposed as a separate NUMA domain (like 4-Socket Xeon)
  - Well suited for MPI applications with 4 or n*4 ranks per KNL
  - SW needs to NUMA optimise to get benefit
  - Good for NUMA-aware code
Cluster-Modes & Memory Modes Combinations

● 5 Flat Memory Mode Variants:
  – Flat-A2A
  – Flat-Quadrant
  – Flat-Hemisphere
  – Flat-SNC4
  – Flat-SNC2

● 5 Cache Memory Mode Variants
  – Cache-A2A
  – Cache-Quadrant
  – Cache-Hemisphere
  – Cache-SNC4
  – Cache-SNC2

● 5 x 3 = 15 Hybrid Variants
Using numactl

- use only DDR (default)
  
  numactl --membind=0 ./a.out

- use only MCDRAM in flat-quadrant mode
  
  numactl --membind=1 ./a.out

- use MCDRAM if possible in flat-quadrant mode; else DDR
  
  numactl --preferred=1 ./a.out

- show numactl settings
  
  numactl --hardware

- list available numactl options
  
  numactl --help
Using numactl in various modes

- **Flat-quadrant mode**: use only MCDRAM
  numactl --membind=1 ./a.out

- **Flat-SNC2 mode**: use only MCDRAM
  numactl --membind=2,3 ./a.out

- **Flat-SNC4 mode**: use only MCDRAM
  numactl --membind=4,5,6,7 ./a.out
Changing of Memory and Cluster Modes

- KNL = single chip solution that can change the design of a machine at a level that has traditionally unchangeable
- Operating Systems and applications are not prepared for dynamically changing NUMA distances or changing memory and caching structures
- → Changing either cluster mode or memory mode requires a rebuild of tag directories
  - Requires reboot
  - Takes 15-20 minutes
Selection of Memory Modes in BIOS
Selection of Cluster Modes in BIOS

Uncore Configuration

Select Cluster Mode:
Auto will try to set Quadrant model if supported.

Options:
- All2011
- SNC-2
- SNC-4
- Hemisphere
- Quadrant
- Auto
- <Enable>
- <Store>

Copyright (c) 2006-2016, Intel Corporation
Selection of Cluster Modes via Script @ LRZ

mcct03:~ # sudo /usr/local/bin/SetKnlMode
ERROR: A valid mode was not specified.

Usage: SetKnlMode [-fq] -m MODE

-f - Force setting of mode, even if it appears to be already set.
-q - Reboot after 1 minutes, instead of 5 minutes.
-m MODE - Name the mode to change to. Available modes are:
  - CacheQuadrant
  - CacheSNC-4
  - FlatQuadrant
  - FlatSNC-4
  - HybridSNC-4

Uses Supermicro Update Manager BIOS Management:
/usr/local/sbin/sum -c ChangeBiosCfg --file filename
● Following the suggestions of the Intel experts, we finally adopted one configuration only for all the KNL racks serving the knldebug and knlprod (academic) queues, namely:

  cache/quadrant

● The queues serving the Marconi FUSION partition allow instead the use of nodes in flat/quadrant or cache/quadrant modes
Using the memkind library

- https://github.com/memkind/memkind

- The *memkind* library is a **user extensible heap manager** built on top of *jemalloc* (http://jemalloc.net/) which enables control of memory characteristics and a **partitioning of the heap between kinds of memory**.

- The *memkind* library delivers two interfaces:
  - *hbwmalloc.h* - recommended for high-bandwidth memory use cases (stable) → `man memkind`
  - *memkind.h* - generic interface for more complex use cases (partially unstable) → `man hbwmalloc`
SYNOPSIS

#include <hbwmalloc.h>
int hbw_check_available(void);
void* hbw_malloc(size_t size);
void* hbwcalloc(size_t nmemb, size_t size);
void* hbwrealloc (void *ptr, size_t size);
void hbw_free (void *ptr);
int hbw_posix_memalign(void **memptr, size_t alignment, size_t size);
int hbw_posix_memalign_psize(void **memptr, size_t alignment, size_t size,
                             hbwpagesize_t pagesize);
hbw_policy_t hbw_get_policy(void);
int hbw_set_policy(hbw_policy_t mode);
int hbw_verify_memory_region(void *addr, size_t size, int flags);

Details: man hbwmalloc
Using the memkind library: Policies

hbw_set_policy() sets the current fallback policy. The policy can be modified only once in the lifetime of an application and before calling hbw_*alloc() or hbw_posix_memalign*() function.

Note: If the policy is not set, than HBW_POLICY_PREFERRED will be used by default.

HBW_POLICY_BIND
If insufficient high bandwidth memory from the nearest NUMA node is available to satisfy a request, the allocated pointer is set to NULL and errno is set to ENOMEM. If insufficient high bandwidth memory pages are available at fault time the Out Of Memory (OOM) killer is triggered. Note that pages are faulted exclusively from the high bandwidth NUMA node nearest at time of allocation, not at time of fault.

HBW_POLICY_PREFERRED
If insufficient memory is available from the high bandwidth NUMA node closest at allocation time, fall back to standard memory (default) with the smallest NUMA distance.

HBW_POLICY_INTERLEAVE
Interleave faulted pages from across all high bandwidth NUMA nodes using standard size pages (the Transparent Huge Page feature is disabled).
Using the memkind library: alloc

**Traditional:**

```c
#include <stdlib.h>
...
double *A;
A = (double*) malloc(sizeof(double) * N);
...
Free(A);
```

**Memkind library:**

```c
#include <hbwmalloc.h>
...
double *A;
A = (double*) hbw_malloc(sizeof(double) * N);
...
hbw_free(A);
```
Using the memkind library:

posix_memalign

**Traditional:**

```c
#include <stdlib.h>
...
int ret; double *A;
ret = posix_memalign((void *)A, 64, sizeof(double)*N);
if (ret!=0) //error
...
free (A);
```

**Memkind library:**

```c
#include <hbwmalloc.h>
...
int ret; double *A;
ret = hbw_posix_memalign((void*) A, 64, sizeof(double)*N);
if (ret!=0) //error
...
hbw_free (A);
```
Intel Fortran Extensions for MCDRAM

Traditional:
real, allocatable :: A(:)
...
ALLOCATE (A(1:1024))

MCDRAM:
real, allocatable :: A(:)
!DIR$ ATTRIBUTES FASTMEM :: A
!DIR$ ATTRIBUTES FASTMEM, ALIGN:64 :: A   ! Alternative for alignment
...
ALLOCATE (A(1:1024))

Alternativ:
real, allocatable :: A(:)
!dir# FASTMEM
ALLOCATE (A(1:1024))

FOR_SET_FASTMEM_POLICY(...) to change policy.
OpenMP 5.0 MCDRAM Support

- Memory Management Support for OpenMP 5.0
- Support for new types of memory: High Bandwidth Memory, Non-volatile memory etc.


- This Technical Report augments the OpenMP TR 4 document with language features for managing memory on systems with heterogeneous memories.
- To be released approx. Nov. 2018
The following slides contain material obtained on the Stampede2 Supercomputer at Texas Advanced Computing Center, The University of Texas at Austin.

https://portal.tacc.utexas.edu/user-guides/stampede2

Stampede2 is the flagship supercomputer at the Texas Advanced Computing Center (TACC). It will enter full production in the Fall 2017 as an 18-petaflop national resource that builds on the successes of the original Stampede system it replaces.

Thanks to TACC for access to Stampede2 during an ISC‘17 Tutorial.
Stampede2 Supercomputer
Stampede2 Supercomputer

Model: Intel Xeon Phi 7250

Total cores per KNL node: 68 cores on a single socket
Hardware threads per core: 4
Hardware threads per node: 68 x 4 = 272
Clock rate: 1.4GHz

RAM: 96GB DDR4 plus 16GB high-speed MCDRAM. Configurable in two important ways; see Programming and Performance for more info.

All but 508 KNL nodes have a 132GB /tmp partition on a 200GB Solid State Drive (SSD). The 508 KNLs originally installed as the

Local storage: Stampede1 KNL sub-system each have a 58GB /tmp partition on 112GB SSDs. The latter nodes currently make up the flat-quadrant and flat-snc4 queues.
Useful commands and system files

- /proc/cpuinfo
- /proc/meminfo
- numastat -H
- numastat -m (includes huge page info, too)
- numastat -p pid
- /sys/devices/system/node/node*/meminfo and other files
- /usr/bin/memkind-hbw-nodes
<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>proc</td>
<td>/proc/cpuinfo</td>
</tr>
<tr>
<td>processor</td>
<td>271</td>
</tr>
<tr>
<td>vendor_id</td>
<td>GenuineIntel</td>
</tr>
<tr>
<td>cpu family</td>
<td>6</td>
</tr>
<tr>
<td>model</td>
<td>87</td>
</tr>
<tr>
<td>model name</td>
<td>Intel(R) Xeon Phi(TM) CPU 7250 @ 1.40GHz</td>
</tr>
<tr>
<td>stepping</td>
<td>1</td>
</tr>
<tr>
<td>microcode</td>
<td>0x1ac</td>
</tr>
<tr>
<td>cpu MHz</td>
<td>1543.445</td>
</tr>
<tr>
<td>cache size</td>
<td>1024 KB</td>
</tr>
<tr>
<td>physical id</td>
<td>0</td>
</tr>
<tr>
<td>siblings</td>
<td>272</td>
</tr>
<tr>
<td>core id</td>
<td>73</td>
</tr>
<tr>
<td>cpu cores</td>
<td>68</td>
</tr>
<tr>
<td>apicid</td>
<td>295</td>
</tr>
<tr>
<td>initial apicid</td>
<td>295</td>
</tr>
<tr>
<td>fpu</td>
<td>yes</td>
</tr>
<tr>
<td>fpu_exception</td>
<td>yes</td>
</tr>
<tr>
<td>cpuid level</td>
<td>13</td>
</tr>
<tr>
<td>wp</td>
<td>yes</td>
</tr>
<tr>
<td>flags</td>
<td>fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 cflush dts acpi mmx fxsr sse sse2 ssht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperff perf</td>
</tr>
<tr>
<td>eagerfpu</td>
<td>pni pclmulqdq dtes64 monitor ds_cpl est tm2 ssse3 fma cx16 xtpr pdcz sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer xsave avx f16c rdrand lahf_lm abm 3dnowprefetch ida arat epb pln pts dtherm fsbgbase tsc_adjust bmi1 avx2 smep bmi2 erms avx512f rdseed adx avx512pf avx512er avx512cd xsaveopt</td>
</tr>
<tr>
<td>bogomips</td>
<td>2793.63</td>
</tr>
<tr>
<td>cflush size</td>
<td>64</td>
</tr>
<tr>
<td>cache_alignment</td>
<td>64</td>
</tr>
<tr>
<td>address sizes</td>
<td>46 bits physical, 48 bits virtual</td>
</tr>
<tr>
<td>power management:</td>
<td></td>
</tr>
<tr>
<td>c403-003.stampede2(9)</td>
<td></td>
</tr>
</tbody>
</table>
Flat-Quadrant Mode:
- MemTotal: 115218908 kB
- MemFree: 108756608 kB
- MemAvailable: 108562240 kB

Cache-Quadrant Mode:
- MemTotal: 98696336 kB
- MemFree: 92462428 kB
- MemAvailable: 92282108 kB

Flat-SNC-4 Mode:
- MemTotal: 115217380 kB
- MemFree: 109216500 kB
- MemAvailable: 108983732 kB
numactl -H in Flat-Quadrant Mode

available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82
83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128
129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149
150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170
171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191
192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212
213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254
255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
node 0 size: 98207 MB
node 0 free: 90483 MB
node 1 cpus:
node 1 size: 16384 MB
node 1 free: 15723 MB
node distances:
node 0  1
  0:  10  31
  1:  31  10
numactl -H in Cache Mode

available: 1 nodes (0)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82
83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128
129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149
150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170
171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191
192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212
213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254
255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271

node 0 size: 98199 MB
node 0 free: 90294 MB
node distances:
node 0
  0: 10
numactl -H in Flat-SNC4 Mode

available: 8 nodes (0-7)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221
node 0 size: 24479 MB
node 0 free: 21852 MB
node 1 cpus: 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239
node 1 size: 24576 MB
node 1 free: 22867 MB
node 2 cpus: 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
node 2 size: 24576 MB
node 2 free: 22887 MB
node 3 cpus: 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
node 3 size: 24576 MB
node 3 free: 23144 MB
numactl -H in Flat-SNC4 Mode contnd.

node 4 cpus:
node 4 size: 4096 MB
node 4 free: 3968 MB
node 5 cpus:
node 5 size: 4096 MB
node 5 free: 3976 MB
node 6 cpus:
node 6 size: 4096 MB
node 6 free: 3976 MB
node 7 cpus:
node 7 size: 4096 MB
node 7 free: 3975 MB

node distances:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>31</td>
<td>41</td>
<td>41</td>
<td>41</td>
</tr>
<tr>
<td>1</td>
<td>21</td>
<td>10</td>
<td>21</td>
<td>21</td>
<td>41</td>
<td>31</td>
<td>41</td>
<td>41</td>
</tr>
<tr>
<td>2</td>
<td>21</td>
<td>21</td>
<td>10</td>
<td>21</td>
<td>41</td>
<td>41</td>
<td>31</td>
<td>41</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>10</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>31</td>
</tr>
<tr>
<td>4</td>
<td>31</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>10</td>
<td>41</td>
<td>41</td>
<td>41</td>
</tr>
<tr>
<td>5</td>
<td>41</td>
<td>31</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>10</td>
<td>41</td>
<td>41</td>
</tr>
<tr>
<td>6</td>
<td>41</td>
<td>41</td>
<td>31</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>10</td>
<td>41</td>
</tr>
<tr>
<td>7</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>31</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>10</td>
</tr>
</tbody>
</table>

Distances:
10 „near“ DDR
21 „far“ DDR
31 „near“ MCDRAM
41 „far“ MCDRAM

Affinitization of DDR and MCDRAM to the divisions of the KNL!
Flat-Quadrant:
c455-001.stampede2(17)$ memkind-hbw-nodes 1
c455-001.stampede2(18)$

Flat-SNC4:
c463-001.stampede2(1)$ memkind-hbw-nodes 4,5,6,7
c463-001.stampede2(2)$

Cache:
c403-001.stampede2(2)$ memkind-hbw-nodes
c403-001.stampede2(3)$
### STREAM Benchmark in Cache Mode

**export OMP_NUM_THREADS=1**

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>17843.4</td>
<td>0.009051</td>
<td>0.008967</td>
<td>0.009453</td>
</tr>
<tr>
<td>Scale:</td>
<td>14305.0</td>
<td>0.011287</td>
<td>0.011185</td>
<td>0.011523</td>
</tr>
<tr>
<td>Add:</td>
<td>15736.8</td>
<td>0.015452</td>
<td>0.015251</td>
<td>0.015739</td>
</tr>
<tr>
<td>Triad:</td>
<td>15622.9</td>
<td>0.015512</td>
<td>0.015362</td>
<td>0.015851</td>
</tr>
</tbody>
</table>

**export OMP_NUM_THREADS=68**

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>263585.5</td>
<td>0.000624</td>
<td>0.000607</td>
<td>0.000640</td>
</tr>
<tr>
<td>Scale:</td>
<td>269297.2</td>
<td>0.000612</td>
<td>0.000594</td>
<td>0.000631</td>
</tr>
<tr>
<td>Add:</td>
<td>325244.9</td>
<td>0.000772</td>
<td>0.000738</td>
<td>0.000798</td>
</tr>
<tr>
<td>Triad:</td>
<td>308499.2</td>
<td>0.000803</td>
<td>0.000778</td>
<td>0.000872</td>
</tr>
</tbody>
</table>
STREAM Benchmark in Flat-Quadrant Mode

c455-001.stampede2(27)$ export OMP_NUM_THREADS=1
c455-001.stampede2(30)$ ./stream

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>18344.3</td>
<td>0.008848</td>
<td>0.008722</td>
<td>0.009068</td>
</tr>
<tr>
<td>Scale:</td>
<td>15184.7</td>
<td>0.010682</td>
<td>0.010537</td>
<td>0.011006</td>
</tr>
<tr>
<td>Add:</td>
<td>16788.1</td>
<td>0.014392</td>
<td>0.014296</td>
<td>0.014504</td>
</tr>
<tr>
<td>Triad:</td>
<td>16712.0</td>
<td>0.014462</td>
<td>0.014361</td>
<td>0.014548</td>
</tr>
</tbody>
</table>

455-001.stampede2(21)$ numactl -m 0 ./stream

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>18377.9</td>
<td>0.008787</td>
<td>0.008706</td>
<td>0.008876</td>
</tr>
<tr>
<td>Scale:</td>
<td>15207.8</td>
<td>0.010655</td>
<td>0.010521</td>
<td>0.010875</td>
</tr>
<tr>
<td>Add:</td>
<td>16677.2</td>
<td>0.014519</td>
<td>0.014391</td>
<td>0.014714</td>
</tr>
<tr>
<td>Triad:</td>
<td>16724.8</td>
<td>0.014447</td>
<td>0.014350</td>
<td>0.014541</td>
</tr>
</tbody>
</table>

c455-001.stampede2(24)$ numactl -m 1 ./stream

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>17999.9</td>
<td>0.009024</td>
<td>0.00889</td>
<td>0.009474</td>
</tr>
<tr>
<td>Scale:</td>
<td>14455.0</td>
<td>0.011269</td>
<td>0.011069</td>
<td>0.011690</td>
</tr>
<tr>
<td>Add:</td>
<td>15846.7</td>
<td>0.015331</td>
<td>0.015145</td>
<td>0.016039</td>
</tr>
<tr>
<td>Triad:</td>
<td>15850.0</td>
<td>0.015344</td>
<td>0.015142</td>
<td>0.015800</td>
</tr>
</tbody>
</table>

------------------------------------------------------------------------
STREAM Benchmark in Flat-Quadrant Mode

```
c455-001.stampede2(27)$ export OMP_NUM_THREADS=68
c455-001.stampede2(30)$ ./stream
```

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>82768.7</td>
<td>0.001939</td>
<td>0.001933</td>
<td>0.001945</td>
</tr>
<tr>
<td>Scale:</td>
<td>82901.6</td>
<td>0.001937</td>
<td>0.001930</td>
<td>0.001947</td>
</tr>
<tr>
<td>Add:</td>
<td>88332.1</td>
<td>0.002731</td>
<td>0.002717</td>
<td>0.002741</td>
</tr>
<tr>
<td>Triad:</td>
<td>88425.2</td>
<td>0.002732</td>
<td>0.002714</td>
<td>0.002757</td>
</tr>
</tbody>
</table>

```
ac455-001.stampede2(28)$ numactl -m 0 ./stream
```

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>83334.0</td>
<td>0.001937</td>
<td>0.001920</td>
<td>0.001961</td>
</tr>
<tr>
<td>Scale:</td>
<td>82768.7</td>
<td>0.001938</td>
<td>0.001933</td>
<td>0.001950</td>
</tr>
<tr>
<td>Add:</td>
<td>88131.1</td>
<td>0.002742</td>
<td>0.002723</td>
<td>0.002769</td>
</tr>
<tr>
<td>Triad:</td>
<td>88138.8</td>
<td>0.002745</td>
<td>0.002723</td>
<td>0.002754</td>
</tr>
</tbody>
</table>

```
ac455-001.stampede2(29)$ numactl -m 1 ./stream
```

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>397093.9</td>
<td>0.000434</td>
<td>0.000403</td>
<td>0.000473</td>
</tr>
<tr>
<td>Scale:</td>
<td>426901.2</td>
<td>0.000400</td>
<td>0.000375</td>
<td>0.000428</td>
</tr>
<tr>
<td>Add:</td>
<td>433146.7</td>
<td>0.000578</td>
<td>0.000554</td>
<td>0.000602</td>
</tr>
<tr>
<td>Triad:</td>
<td>345328.6</td>
<td>0.000835</td>
<td>0.000695</td>
<td>0.000944</td>
</tr>
</tbody>
</table>

26.-28.6.2017 Intel MIC Programming Workshop @ LRZ
STREAM Benchmark in Flat-SNC4 Mode

```bash
c455-001.stampede2(27)$ export OMP_NUM_THREADS=68
c455-001.stampede2(30)$ ./stream
```

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>74773.1</td>
<td>0.002148</td>
<td>0.002140</td>
<td>0.002189</td>
</tr>
<tr>
<td>Scale:</td>
<td>78970.2</td>
<td>0.002031</td>
<td>0.002026</td>
<td>0.002038</td>
</tr>
<tr>
<td>Add:</td>
<td>79156.5</td>
<td>0.003040</td>
<td>0.003032</td>
<td>0.003066</td>
</tr>
<tr>
<td>Triad:</td>
<td>79337.4</td>
<td>0.003032</td>
<td>0.003025</td>
<td>0.003050</td>
</tr>
</tbody>
</table>

```bash
c455-001.stampede2(30)$ numactl -m 0 ./stream
```

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>36012.3</td>
<td>0.004460</td>
<td>0.004443</td>
<td>0.004475</td>
</tr>
<tr>
<td>Scale:</td>
<td>36093.6</td>
<td>0.004448</td>
<td>0.004433</td>
<td>0.004468</td>
</tr>
<tr>
<td>Add:</td>
<td>38903.7</td>
<td>0.006185</td>
<td>0.006169</td>
<td>0.006212</td>
</tr>
<tr>
<td>Triad:</td>
<td>38942.8</td>
<td>0.006176</td>
<td>0.006163</td>
<td>0.006206</td>
</tr>
</tbody>
</table>

```bash
c455-001.stampede2(30)$ numactl -m 1 ./stream
```

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>36381.3</td>
<td>0.004488</td>
<td>0.004398</td>
<td>0.004804</td>
</tr>
<tr>
<td>Scale:</td>
<td>36430.6</td>
<td>0.004543</td>
<td>0.004392</td>
<td>0.005053</td>
</tr>
<tr>
<td>Add:</td>
<td>39525.4</td>
<td>0.006552</td>
<td>0.006072</td>
<td>0.007996</td>
</tr>
<tr>
<td>Triad:</td>
<td>39519.2</td>
<td>0.006190</td>
<td>0.006073</td>
<td>0.006780</td>
</tr>
</tbody>
</table>

26.-28.6.2017 Intel MIC Programming Workshop @ LRZ
STREAM Benchmark in Flat-SNC4 Mode

```bash
c455-001.stampede2(27)$ export OMP_NUM_THREADS=68

c455-001.stampede2(27)$ numactl -m 2 ./scratch

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>34669.0</td>
<td>0.004625</td>
<td>0.004615</td>
<td>0.004635</td>
</tr>
<tr>
<td>Scale:</td>
<td>34692.3</td>
<td>0.004621</td>
<td>0.004612</td>
<td>0.004630</td>
</tr>
<tr>
<td>Add:</td>
<td>38199.5</td>
<td>0.006288</td>
<td>0.006283</td>
<td>0.006297</td>
</tr>
<tr>
<td>Triad:</td>
<td>38215.4</td>
<td>0.006287</td>
<td>0.006280</td>
<td>0.006298</td>
</tr>
</tbody>
</table>

c455-001.stampede2(27)$ numactl -m 3 ./scratch

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>34535.2</td>
<td>0.004650</td>
<td>0.004633</td>
<td>0.004665</td>
</tr>
<tr>
<td>Scale:</td>
<td>34601.1</td>
<td>0.004644</td>
<td>0.004624</td>
<td>0.004664</td>
</tr>
<tr>
<td>Add:</td>
<td>38161.8</td>
<td>0.006322</td>
<td>0.006289</td>
<td>0.006349</td>
</tr>
<tr>
<td>Triad:</td>
<td>38198.0</td>
<td>0.006324</td>
<td>0.006283</td>
<td>0.006346</td>
</tr>
</tbody>
</table>

c455-001.stampede2(27)$ numactl -m 4 ./scratch

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>105186.3</td>
<td>0.001539</td>
<td>0.001521</td>
<td>0.001560</td>
</tr>
<tr>
<td>Scale:</td>
<td>105334.9</td>
<td>0.001541</td>
<td>0.001519</td>
<td>0.001558</td>
</tr>
<tr>
<td>Add:</td>
<td>101516.0</td>
<td>0.002378</td>
<td>0.002364</td>
<td>0.002406</td>
</tr>
<tr>
<td>Triad:</td>
<td>102040.8</td>
<td>0.002371</td>
<td>0.002352</td>
<td>0.002398</td>
</tr>
</tbody>
</table>
```

26.-28.6.2017 Intel MIC Programming Workshop @ LRZ
STREAM Benchmark in Flat-SNC4 Mode

c455-001.stampede2(27)$ export OMP_NUM_THREADS=68

c455-001.stampede2(27)$ numactl -m 5 ./scratch

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>107442.9</td>
<td>0.001497</td>
<td>0.001489</td>
<td>0.001504</td>
</tr>
<tr>
<td>Scale:</td>
<td>107236.9</td>
<td>0.001496</td>
<td>0.001492</td>
<td>0.001503</td>
</tr>
<tr>
<td>Add:</td>
<td>104206.3</td>
<td>0.002312</td>
<td>0.002303</td>
<td>0.002325</td>
</tr>
<tr>
<td>Triad:</td>
<td>104672.2</td>
<td>0.002295</td>
<td>0.002293</td>
<td>0.002304</td>
</tr>
</tbody>
</table>

c455-001.stampede2(27)$ numactl -m 6 ./scratch

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>98834.9</td>
<td>0.001627</td>
<td>0.001619</td>
<td>0.001633</td>
</tr>
<tr>
<td>Scale:</td>
<td>98762.1</td>
<td>0.001628</td>
<td>0.001620</td>
<td>0.001635</td>
</tr>
<tr>
<td>Add:</td>
<td>93823.6</td>
<td>0.002569</td>
<td>0.002558</td>
<td>0.002586</td>
</tr>
<tr>
<td>Triad:</td>
<td>94086.6</td>
<td>0.002561</td>
<td>0.002551</td>
<td>0.002572</td>
</tr>
</tbody>
</table>

c455-001.stampede2(27)$ numactl -m 7 ./scratch

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy:</td>
<td>99938.7</td>
<td>0.001608</td>
<td>0.001601</td>
<td>0.001615</td>
</tr>
<tr>
<td>Scale:</td>
<td>99317.5</td>
<td>0.001614</td>
<td>0.001611</td>
<td>0.001620</td>
</tr>
<tr>
<td>Add:</td>
<td>94599.5</td>
<td>0.002545</td>
<td>0.002537</td>
<td>0.002550</td>
</tr>
<tr>
<td>Triad:</td>
<td>94929.6</td>
<td>0.002535</td>
<td>0.002528</td>
<td>0.002543</td>
</tr>
</tbody>
</table>

26.-28.6.2017 Intel MIC Programming Workshop @ LRZ
Oversubscription of MCDRAM

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 6000000000
int main(int argc, char **argv)
{
    float *list;
    long long int i;

    list = (float *) malloc(N * sizeof(float));
    if(list != NULL) {printf("\nmemory is reserved\n");
        }else {printf("\nNo memory free.\n");}

    printf("Sizeof float:  %i Bytes\n",sizeof(float));
    printf("Sizeof list:  %lli Bytes\n",sizeof(float)*N);
    printf("Sizeof list:  %lli GB\n",sizeof(float)*N/1024/1024/1024);

    for(i=0; i<N; i++) list[i] = i;
    return 0;
}
Oversubscription of MCDRAM

c455-002.stampede2(14)$ time numactl -m 0 ./alloc
memory is reserved
Sizeof float:  4 Bytes
Sizeof list:  24000000000 Bytes
Sizeof list:  22 GB

real   1m38.960s
user   1m35.385s
sys    0m3.571s
c455-002.stampede2(15)$
Oversubscription of MCDRAM

c455-002.stampede2(11)$ time numactl -m 1 ./alloc
memory is reserved
Sizeof float:  4 Bytes
Sizeof list:  24000000000 Bytes
Sizeof list:  22 GB
Killed
real    1m11.186s
user    1m4.327s
sys     0m6.472s
c455-002.stampede2(12)$

c455-002.stampede2(15)$ time numactl --preferred=1 ./alloc
memory is reserved
Sizeof float:  4 Bytes
Sizeof list:  24000000000 Bytes
Sizeof list:  22 GB

real    1m38.754s
user    1m35.031s
sys     0m3.714s
c455-002.stampede2(16)$
Oversubscription of MCDRAM

● Using the memkind library:

```c
#include <hbwmalloc.h>
...
list = (float *) hbw_malloc(N * sizeof(float));
```

c455-001.stampede2(6)$ icc -O0 alloc2.c -lmemkind -o alloc2
c455-001.stampede2(7)$  ./alloc2
hbw_check_available=0

memory is reserved
Sizeof float:  4 Bytes
Sizeof list:  24000000000 Bytes
Sizeof list:  22 GB
c455-001.stampede2(8)"

As default fallback policy is HBW_POLICY_PREFERRED!
Performance

Cluster and Memory Modes Performance Comparison

Relative Speedup

From TACC ISC‘17 Tutorial, Baseline = Flat-A2A DDR4

26.-28.6.2017
Intel MIC Programming Workshop @ LRZ
References


- **Tutorial**: ”Introduction to Manycore Programming”, Texas Advanced Computing Center, 2017. Available under a Creative Commons Attribution Non-Commercial 3.0 Unported License. https://creativecommons.org/licenses/by-nc/3.0/

- **Tutorial**: “Introduction to KNL and the ARCHER KNL Cluster”, 2017, Adrian Jackson, EPCC, licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en_US
Intel Xeon Phi Programming Models: Intel Language Extensions for Offload (LEO) I
Supported Offloading Models

- Intel Language Extensions for Offload (LEO)
- OpenMP 4.x Offloading
- “Mine Yours Ours” (MYO) virtual shared memory model
Intel Offload Directives

- **Syntax:**
  - **C:**
    
    ```
    #pragma offload target(mic) <clauses>
    <statement block>
    ```
  - **Fortran:**
    - ```
        !DIR$ offload target(mic) <clauses>
        <statement>
    ```
    - ```
        !DIR$ omp offload target(mic) <clauses>
        <OpenMP construct>
    ```
Intel Offload Directive

- **C:**
  - Pragma can be before any statement, including a compound statement or an OpenMP parallel pragma

- **Fortran:**
  - If OMP is specified: the next line, other than a comment, must be an OpenMP PARALLEL, PARALLEL SECTIONS, or PARALLEL DO directive.
  - If OMP is not specified, next line must:
    - An OpenMP* PARALLEL, PARALLEL SECTIONS, or PARALLEL DO directive
    - A CALL statement
    - An assignment statement where the right side only calls a function
Intel Offload

- Implements the following steps:

1. Memory allocation on the MIC
2. Data transfer from the host to the MIC
3. Execution on the MIC
4. Data transfer from the MIC to the host
5. Memory deallocation on MIC
#include <stdio.h>

int main (int argc, char* argv[]) {

#pragma offload target(mic)
{
    printf("MIC: Hello world from MIC.\n");
}

printf("Host: Hello world from host.\n");
}
PROGRAM HelloWorld

!DIR$ offload begin target(MIC)
PRINT *, 'MIC: Hello world from MIC'
!DIR$ end offload

PRINT *, 'Host: Hello world from host'
END
Intel Offload: Hello World in C

lu65fok@login12:~/tests> icpc offload1.c -o offload1

lu65fok@login12:~/tests> ./offload1
offload error: cannot offload to MIC - device is not available

lu65fok@i01r13c01:~/tests> ./offload1
Host: Hello world from host.
MIC: Hello world from MIC.
lu65fok@login12:~/tests> ifort offload1.f90 -o offload1

lu65fok@login12:~/tests> ./offload1
offload error: cannot offload to MIC - device is not available

lu65fok@i01r13c01:~/tests> ./offload1
Host: Hello world from host.
MIC: Hello world from MIC.
#include <stdio.h>
#include <unistd.h>

int main (int argc, char* argv[]) {
    char hostname[100];
    gethostname(hostname,sizeof(hostname));

    #pragma offload target(mic)
    {
        char michostname[100];
        gethostname(michostname, sizeof(michostname));
        printf("MIC: Hello world from MIC. I am %s and I have %ld logical cores. I was called from host: %s \\
               ", michostname, sysconf(_SC_NPROCESSORS_ONLN), hostname);
    }
}
lu65fok@login12:~/tests> icpc offload.c -o offload

lu65fok@i01r13c01:~/tests> ./offload
Host: Hello world from host. I am i01r13c01 and I have 32 logical cores.
MIC: Hello world from MIC. I am i01r13c01-mic0 and I have 240 logical cores. I was called from host: i01r13c01
Intel Offload: -offload=optional / mandatory

```
lu65fok@login12:~/tests> icpc -offload=optional offload.c -o offload

lu65fok@login12:~/tests> ./offload
MIC: Hello world from MIC. I am login12 and I have 16 logical cores. I was called from host: login12
Host: Hello world from host. I am login12 and I have 16 logical cores.
```

```
lu65fok@login12:~/tests> icpc -offload=mandatory offload.c -o offload
lu65fok@login12:~/tests> ./offload
offload error: cannot offload to MIC - device is not available
```
lu65fok@login12:~/tests> icpc -offload=none offload.c -o offload
offload.c(13): warning #161: unrecognized #pragma
    #pragma offload target(mic)
    ^
lu65fok@login12:~/tests>

lu65fok@i01r13c01:~/tests> ./offload
MIC: Hello world from MIC. I am i01r13c01 and I have 32 logical cores.
I was called from host: i01r13c01
Host: Hello world from host. I am i01r13c01 and I have 32 logical cores.
#include <stdio.h>
#include <stdlib.h>

int main()
{
    #pragma offload target (mic)
    {
        system("command");
    }

}
null
HOSTNAME=i01r13c01-mic0
HOSTTYPE=k1om
IFS=''

MACHTYPE=k1om-mpss-linux-gnu
OPTERR=1
OPTIND=1
OSTYPE=linux-gnu
PATH=/usr/bin:/bin
POSIXLY_CORRECT=y
PPID=37141
PS4=' '+
PWD=/var/volatile/tmp/coi_procs/1/37141
SHELL=/bin/false
SHELLOPTS=braceexpand:hashall:interactive-comments:posix
SHLVL=1
TERM=dumb
UID=400
_=sh
#pragma offload target (mic)
{
    system("hostname");
    system("uname -a");
    system("whoami");
    system("id");
}
lu65fok@i01r13c01:~/tests> ./system
i01r13c01-mic0
Linux i01r13c01-mic0 2.6.38.8+mpss3.1.2 #1 SMP Wed Dec 18 19:09:36 PST 2013 k1om GNU/Linux
micuser
uid=400(micuser) gid=400(micuser)
Offload: Using several MIC Coprocessors

- To query the number of coprocessors:
  \[
  \text{int nmics = } \_\_\_\text{Offload\_number\_of\_devices()}
  \]

- To specify which coprocessor \( n < \text{nmics} \) should do the computation:
  \[
  \text{#pragma offload target(mic:n)}
  \]

- If \( n > \text{nmics} \) then coprocessor \( (n \% \text{nmics}) \) is used

- **Important for:**
  - Asynchronous offloads
  - Coprocessor-Persistent data
Offloading OpenMP Computations

- **C/C++ & OpenMP:**
  
  ```c
  #pragma offload target(mic)
  #pragma omp parallel for
  for (int i=0;i<n;i++) {
      a[i]=c*b[i]+d;
  }
  ```

- **Fortran & OpenMP**
  
  ```fortran
  !DIR$ offload target(mic)
  !$OMP PARALLEL DO
  do i = 1, n
      a(i) = c*b(i) + d
  end do
  !$omp END PARALLEL DO
  ```
Functions and Variables on the MIC

- C:
  - __attribute__((target(mic))) variables / function
  - __declspec(target(mic)) variables / function
  - #pragma offload_attribute(push, target(mic))
    … multiple lines with variables / functions
    #pragma offload_attribute(pop)

- Fortran:
  !DIR$ attributes offload:mic::: variables / function
<table>
<thead>
<tr>
<th>Clauses</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple coprocessors</td>
<td><code>target(mic[:unit] )</code></td>
<td>Select specific coprocessors</td>
</tr>
<tr>
<td>Conditional offload</td>
<td><code>if (condition) / manadatory</code></td>
<td>Select coprocessor or host compute</td>
</tr>
<tr>
<td>Inputs</td>
<td><code>in(var-list modifiers_{opt})</code></td>
<td>Copy from host to coprocessor</td>
</tr>
<tr>
<td>Outputs</td>
<td><code>out(var-list modifiers_{opt})</code></td>
<td>Copy from coprocessor to host</td>
</tr>
<tr>
<td>Inputs &amp; outputs</td>
<td><code>inout(var-list modifiers_{opt})</code></td>
<td>Copy host to coprocessor and back when offload completes</td>
</tr>
<tr>
<td>Non-copied data</td>
<td><code>nocopy(var-list modifiers_{opt})</code></td>
<td>Data is local to target</td>
</tr>
<tr>
<td>Async. Offload</td>
<td><code>signal(signal-slot)</code></td>
<td>Trigger asynchronous Offload</td>
</tr>
<tr>
<td>Async. Offload</td>
<td><code>wait(signal-slot)</code></td>
<td>Wait for completion</td>
</tr>
</tbody>
</table>
## Intel Offload Modifier Options

<table>
<thead>
<tr>
<th>Modifiers</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify copy length</td>
<td><code>length(N)</code></td>
<td>Copy N elements of pointer’s type</td>
</tr>
<tr>
<td>Coprocessor memory allocation</td>
<td><code>alloc_if (bool)</code></td>
<td>Allocate coprocessor space on this offload (default: TRUE)</td>
</tr>
<tr>
<td>Coprocessor memory release</td>
<td><code>free_if (bool)</code></td>
<td>Free coprocessor space at the end of this offload (default: TRUE)</td>
</tr>
<tr>
<td>Array partial allocation &amp; variable relocation</td>
<td><code>alloc (array-slice) in (var-expr)</code></td>
<td>Enables partial array allocation and data copy into other vars &amp; ranges</td>
</tr>
</tbody>
</table>
Intel Offload: Data Movement

- #pragma offload target(mic) in(in1,in2,…)
  out(out1,out2,…) inout(inout1,inout2,…)

- **At Offload start:**
  - Allocate Memory Space on MIC for all variables
  - Transfer in/inout variables from Host to MIC

- **At Offload end:**
  - Transfer out/inout variables from MIC to Host
  - Deallocate Memory Space on MIC for all variables
Intel Offload: Data Movement

- data = (double*)malloc(n*sizeof(double));
- #pragma offload target(mic) in(data:length(n))

- Copies n doubles to the coprocessor, not n * sizeof(double) Bytes
- ditto for out() and inout()
An example for Offloading: Offloading Code

```
#pragma offload target(mic) in(a,b:length(n*n)) inout(c:length(n*n))
{
#pragma omp parallel for
    for( i = 0; i < n; i++ ) {
        for( k = 0; k < n; k++ ) {
            #pragma vector aligned
            #pragma ivdep
            for( j = 0; j < n; j++ ) {
                //c[i][j] = c[i][j] + a[i][k]*b[k][j];
                c[i*n+j] = c[i*n+j] + a[i*n+k]*b[k*n+j];
            }
        }
    }
}
```
Vectorisation Diagnostics

```
lu65fok@login12:~/tests> icc -vec-report2 -openmp offloadmul.c -ooffloadmul
offloadmul.c(35): (col. 5) remark: LOOP WAS VECTORIZED
offloadmul.c(32): (col. 3) remark: loop was not vectorized: not inner loop
offloadmul.c(57): (col. 2) remark: LOOP WAS VECTORIZED
offloadmul.c(54): (col. 7) remark: loop was not vectorized: not inner loop
offloadmul.c(53): (col. 5) remark: loop was not vectorized: not inner loop
offloadmul.c(8): (col. 9) remark: loop was not vectorized: existence of vector dependence
offloadmul.c(7): (col. 5) remark: loop was not vectorized: not inner loop
offloadmul.c(57): (col. 2) remark: *MIC* LOOP WAS VECTORIZED
offloadmul.c(54): (col. 7) remark: *MIC* loop was not vectorized: not inner loop
offloadmul.c(53): (col. 5) remark: *MIC* loop was not vectorized: not inner loop

```
Intel Offload: Example

```c
__attribute__((target(mic))) void mxm(int n, double * restrict a, double * restrict b,
        double *restrict c ){
    int i,j,k;
    for( i = 0; i < n; i++ ) {
        ...
    }
}

main(){
    ...
    #pragma offload target(mic) in(a,b:length(n*n)) inout(c:length(n*n))
    {
        mxm(n,a,b,c);
    }
}
```
Offload Diagnostics

```bash
u65fok@i01r13c06:~/tests> export OFFLOAD_REPORT=2

lu65fok@i01r13c06:~/tests> ./offloadmul

[Offload] [MIC 0] [File]  offloadmul.c
[Offload] [MIC 0] [Line]  50
[Offload] [MIC 0] [Tag]   Tag 0
[Offload] [HOST]  [Tag 0] [CPU Time]        51.927456(seconds)
[Offload] [MIC 0] [Tag 0] [CPU->MIC Data]   24000016 (bytes)
[Offload] [MIC 0] [Tag 0] [MIC Time]        50.835065(seconds)
[Offload] [MIC 0] [Tag 0] [MIC->CPU Data]   8000016 (bytes)
```
lu65fok@i01r13c06:~/tests> export H_TRACE=1

lu65fok@i01r13c06:~/tests> ./offloadmul

HOST: Offload function
__offload_entry_offloadmul_c_50mainicc638762473Jnx4JU,
is_empty=0, #varDescs=7, #waits=0, signal=none
HOST: Total pointer data sent to target: [24000000] bytes
HOST: Total copyin data sent to target: [16] bytes
HOST: Total pointer data received from target: [8000000] bytes
MIC0: Total copyin data received from host: [16] bytes
MIC0: Total copyout data sent to host: [16] bytes
HOST: Total copyout data received from target: [16] bytes

Offload Diagnostics

lu65fok@i01r13c06:$ export H_TIME=1

lu65fok@i01r13c06:$ ./offloadmul

[Offload] [MIC 0] [File] offloadmul.c

[Offload] [MIC 0] [Line] 50

[Offload] [MIC 0] [Tag] Tag 0

[Offload] [HOST] [Tag 0] [CPU Time] 51.920016(seconds)

[Offload] [MIC 0] [Tag 0] [MIC Time] 50.831497(seconds)

******************************************************************************************

          timer data   (sec)

******************************************************************************************

lu65fok@i01r13c06:$
Environment Variables

● Host environment variables are automatically forwarded to the coprocessor when offload mode is used.

● To avoid names collisions:
  – Set MIC_ENVIRONMENT_PREFIX=MIC on the host
  – Then only names with prefix MIC_ are forwarded to the coprocessor with prefix stripped
  – Exception: MIC_LD_LIBRARY_PATH is never passed to the coprocessor.
  – Value of LD_LIBRARY_PATH cannot be changed via forwarding of environment variables.
#include <stdio.h>
#include <stdlib.h>

int main()
{
    #pragma offload target (mic)
    {
        char* varmic = getenv("VAR");
        if (varmic) {
            printf("VAR=%s on MIC.\n", varmic);
        } else {
            printf("VAR is not defined on MIC.\n");
        }
    }
    char* varhost = getenv("VAR");
    if (varhost) {
        printf("VAR=%s on host.\n", varhost);
    } else {
        printf("VAR is not defined on host.\n");
    }
}
Environment Variables on the MIC

lu65fok@i01r13c01:~/tests> ./env
VAR is not defined on host.
VAR is not defined on MIC.

lu65fok@i01r13c01:~/tests> export VAR=299792458
lu65fok@i01r13c01:~/tests> ./env
VAR=299792458 on host.
VAR=299792458 on MIC.

lu65fok@i01r13c01:~/tests> export MIC_ENV_PREFIX=MIC
lu65fok@i01r13c01:~/tests> ./env
VAR=299792458 on host.
VAR is not defined on MIC.

lu65fok@i01r13c01:~/tests> export MIC_VAR=3.141592653
lu65fok@i01r13c01:~/tests> ./env
VAR=299792458 on host.
VAR=3.141592653 on MIC.
The Preprocessor Macro ___MIC___

- The macro ___MIC___ is only defined in code version for MIC, not in the fallback version for the host.
- Allows to check where the code is running.
- Allows to write multiversioned code.
- ___MIC___ also defined in native mode.
The Preprocessor Macro __MIC__

```c
#pragma offload target(mic)
{
    #ifdef __MIC__
        printf("Hello from MIC (offload succeeded).\n");
    #else
        printf("Hello from host (offload to MIC failed!).\n");
    #endif
}
```

lu65fok@login12:~/tests> icpc -offload=optional offload-mic.c
lu65fok@login12:~/tests> ./a.out
Hello from host (offload to MIC failed!).
lu65fok@i01r13c06:~/tests> ./a.out
Hello from MIC (offload succeeded).
Lab: Offload Mode I
Intel Xeon Phi Programming Models: Intel Language Extensions for Offload (LEO) II
Data Traffic without Computation

- 2 possibilities:
  - Blank body of `#pragma offload`, i.e.
    ```
    #pragma offload target(mic) in (data: length(n))
    {}
    ```
  - Use a special pragma `offload_transfer`, i.e.
    ```
    #pragma offload_transfer target(mic) in(data: length(n))
    ```
Asynchronous Offload

- Asynchronous Data Transfer helps to:
  - Overlap computations on host and MIC(s).
  - Work can be distributed to multiple coprocessors.
  - Data transfer time can be masked.
Asynchronous Offload

- To allow asynchronous data transfer, the specifiers `signal()` and `wait()` can be used, i.e.

```c
#pragma offload_transfer target(mic:0) in(data : length(n))
signal(data)

// work on other data concurrent to data transfer …
#pragma offload target(mic:0) wait(data)

nocopy(data : length(N)) out(result : length(N))
{
    ....
    result[i]=data[i] + ...;
}
```

Any pointer type variable can serve as a signal!

Device number must be specified!
Asynchronous Offload

● Alternative to the wait() clause, a new pragma can be used:
  #pragma offload_wait target(mic:0) wait(data)

● Useful if no other offload or data transfer is necessary at the synchronisation point.
Asynchronous Offload to Multiple Coprocessors

```c
char* offload0;
char* offload1;
#pragma offload target(mic:0) signal(offload0) 
       in(data0 : length(N)) out(result0 : length(N))
{
    Calculate(data0, result0);
}
#pragma offload target(mic:1) signal(offload1) 
       in(data1 : length(N)) out(result1 : length(N))
{
    Calculate(data1, result1);
}
#pragma offload_wait target(mic:0) wait(offload0)
#pragma offload_wait target(mic:1) wait(offload1)
```
#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
        {
            // section running on the coprocessor
            #pragma offload target(mic) in(a,b:length(n*n)) inout(c:length(n*n))
            {
                mxm(n,a,b,c);
            }
        }
        #pragma omp section
        {
            // section running on the host
            mxm(n,d,e,f);
        }
    }
}
Persistent Data

- #define ALLOC alloc_if(1)
- #define FREE free_if(1)
- #define RETAIN free_if(0)
- #define REUSE alloc_if(0)

- To allocate data and keep it for the next offload:
  #pragma offload target(mic) in (p:length(l) ALLOC RETAIN)

- To reuse the data and still keep it on the coprocessor:
  #pragma offload target(mic) in (p:length(l) REUSE RETAIN)

- To reuse the data again and free the memory. (FREE is the default, and does not need to be explicitly specified):
  #pragma offload target(mic) in (p:length(l) REUSE FREE)
Intel Xeon Phi Programming Models: OpenMP 4.x Offload Mode
OpenMP 4.x Execution Model

- Create and destroy threads,
- create and destroy leagues of thread teams,
- assign / distribute work (tasks) to threads and devices,
- specify which data is shared and which is private,
- specify which data must be available to the device,
- coordinate thread access to shared data.

Intel MIC Programming Workshop @ LRZ
26.-28.6.2017
OpenMP 4.0x Device Constructs

- Execute code on a target device
  - `omp target [clause[, clause],…]` structured-block

- Manage the device data environment
  - `map ((map-type:) list) // map clause`
    `map-type := alloc | tofrom | to | from`
  - `omp target data [clause[, clause],…]` structured-block
  - `omp target update [clause[, clause],…]`
  - `omp declare target [variable-definitions-or-declarations]`
  - `omp target enter / exit data [clause[, clause],…]` (new: OpenMP 4.5)

- Workshare for acceleration
  - `omp teams [clause[, clause],…]` structured-block
  - `omp distribute [clause[, clause],…]` for-loops
OpenMP 4.x Offloading Computation

- Use **target** construct to
  - Transfer control from the **host** to the **target device**
  - Map variables between the host and target device data environments
- Host thread waits until offloaded region completed
- Use **nowait** for asynchronous execution

```c
#pragma omp target map(to:b,c,d) map(from:a)
{
    #pragma omp parallel for
    for (i=0; i<count; i++) {
        a[i] = b[i] * c + d;
    }
}
```
OpenMP 4.x Target Construct

- Map variables to a **device data environment** and **execute** the construct on that device.
- **#pragma omp target** `[clause[ [, clause] ... ] new-line structured-block]`
  where **clause** is one of the following:
  - `if([ target :] scalar-expression)`
  - `device(integer-expression)`
  - `private(list)`
  - `firstprivate(list)`
  - `map([[[map-type-modifier[,]] map-type: ] list])`
  - `is_device_ptr(list)`
  - `defaultmap(tofrom:scalar)`
  - `nowait`
  - `depend(dependence-type: list)`
OpenMP 4.x Data mapping

map Clause

```c
extern void init(float*, float*, int);
extern void output(float*, int);

void vec_mult(float* p, float* v1, float* v2, int N)
{
  int i;
  init(v1, v2, N);

  #pragma omp target map(to:v1[0:N], v2[0:N]) map(from:p[0:N])
  #pragma omp parallel for
  for (i=0; i<N; i++)
    p[i] = v1[i] * v2[i];

  output(p, N);
}
```

module mults
contains
subroutine vec_mult(p,v1,v2,N)
  real,dimension(*) :: p, v1, v2
  integer :: N, i
  call init(v1, v2, N)
  !$omp target map(to: v1(0:N), v2(0:N)) map(from:p(0:N))
  !$omp parallel do
do i=0,N
p(i) = v1(i) * v2(i)
  end do
  !$omp end target
  call output(p, N)
end subroutine
end module

- On entry to the target region:
  - Allocate corresponding variables v1, v2, and p in the device data environment.
  - Assign the corresponding variables v1 and v2 the value of their respective original variables.
  - The corresponding variable p is undefined.

- On exit from the target region:
  - Assign the original variable p the value of its corresponding variable.
  - The original variables v1 and v2 are undefined.
  - Remove the corresponding variables v1, v2, and p from the device data environment.

26.-28.6.2017 Intel MIC Programming Workshop @ LRZ
OpenMP 4.x Data mapping

- Map variables to a device data environment for the extent of the region:
  - `#pragma omp target data clause[ [,] clause] ... ] new-line structured-block`

- Alternatively use 2 standalone directives
  - `#pragma omp target enter data [ clause[ [,] clause]...] new-line`
  - ...
  - `#pragma omp target exit data [ clause[ [,] clause]...] new-line`

- Standalone directive to synchronize data
  - `#pragma omp target update clause[ [,] clause] ... ] new-line`
#pragma omp target data map(alloc:tmp[:N]) map(to:input[:N]) map(from:res)
{
    #pragma omp target
    #pragma omp parallel for
    for (i=0; i<N; i++)
        tmp[i] = some_computation(input[i], i);

    update_input_array_on_the_host(input);

    #pragma omp target update to(input[:N])

    #pragma omp target
    #pragma omp parallel for reduction(+:res)
    for (i=0; i<N; i++)
        res += final_computation(input[i], tmp[i], i)
}
OpenMP 4.x Teams construct

- The `teams` construct creates a league of thread teams and the master thread of each team executes the region.
- `#pragma omp teams [clause [, clause] ... ] new-line structured-block`

  - where `clause` is one of the following:
    - `num_teams(integer-expression)`
    - `thread_limit(integer-expression)`
    - `default(shared | none)`
    - `private(list)`
    - `firstprivate(list)`
    - `shared(list)`
    - `reduction(reduction-identifier : list)`

- The `teams` construct creates a *league* of thread teams
  - The master thread of each team executes the `teams` region
  - The (max.) number of teams is specified by the `num_teams` clause
  - Each team executes with (max.) `thread_limit` threads
  - Threads in different teams cannot synchronize with each other
The `distribute` construct specifies that the iterations of one or more loops will be executed by the thread teams in the context of their implicit tasks. The iterations are distributed across the master threads of all teams that execute the `teams` region to which the `distribute` region binds.

```
#pragma omp distribute [clause[ [,] clause] ... ] new-line for-loops
```

Where `clause` is one of the following:

- `private(list)`
- `firstprivate(list)`
- `lastprivate(list)`
- `collapse(n)`
- `dist_schedule(kind[, chunk_size])`
Composite constructs and shortcuts in OpenMP 4.5

- 2.10.9  omp distribute simd
- 2.10.10 omp distribute parallel for
- 2.10.11 omp distribute parallel for simd
- 2.11.5  omp target parallel
- 2.11.6  omp target parallel for
- 2.11.7  omp target parallel for simd
- 2.11.8  omp target simd
- 2.11.9  omp target teams
- 2.11.10 omp teams distribute
- 2.11.11 omp teams distribute simd
- 2.11.12 omp target teams distribute
- 2.11.13 omp target teams distribute simd
- 2.11.14 omp teams distribute parallel for
- 2.11.15 omp target teams distribute parallel for
- 2.11.16 omp teams distribute parallel for simd
- 2.11.17 omp target teams distribute parallel for simd

Intel MIC Programming Workshop @ LRZ

26.-28.6.2017
OpenMP 4.x Composite constructs and shortcuts

- **omp distribute**
  - omp distribute simd
  - omp distribute parallel for
  - omp distribute parallel for simd

  Iterations distributed across the master threads of all teams in a teams region
  - dito + executed concurrently using SIMD instructions
  - executed in parallel by multiple threads that are members of multiple teams
  - dito + executed concurrently using SIMD instructions

- **omp teams**
  - omp teams distribute
  - omp teams distribute simd
  - omp teams distribute parallel for
  - omp teams distribute parallel for simd

  creates a league of thread teams and the master thread of each team executes the region

- **omp target**
  - omp target simd
  - omp target parallel
  - omp target parallel for
  - omp target parallel for simd

  map variables to a device data environment and execute the construct on that device

- **omp target teams**
  - omp target teams distribute
  - omp target teams distribute simd
  - omp target teams distribute parallel for
  - omp target teams distribute parallel for simd
OpenMP 4.x SuperMIC Test

- #pragma omp target
- u65fok@i01r13c06:~> ./a.out
Hello world from host: I have 32 cores
omp_get_default_device=0
omp_get_num_devices=2
omp_get_num_teams=1
omp_get_team_num=0
omp_is_initial_device=1
Hello world from MIC i01r13c06-mic0: I have 240 cores
omp_get_num_threads=1

- #pragma omp target teams
Hello world from MIC i01r13c06-mic0: I have 240 cores
omp_get_num_threads=236

- #pragma omp target teams num_teams(4)
Hello world from MIC i01r13c06-mic0: I have 240 cores
omp_get_num_threads=59
Hello world from MIC i01r13c06-mic0: I have 240 cores
omp_get_num_threads=59
Hello world from MIC i01r13c06-mic0: I have 240 cores
omp_get_num_threads=59
Hello world from MIC i01r13c06-mic0: I have 240 cores
omp_get_num_threads=59

- #pragma omp target teams num_teams(4) thread_limit(2)
Hello world from MIC i01r13c06-mic0: I have 240 cores
omp_get_num_threads=2
Hello world from MIC i01r13c06-mic0: I have 240 cores
omp_get_num_threads=2
Hello world from MIC i01r13c06-mic0: I have 240 cores
omp_get_num_threads=2
Hello world from MIC i01r13c06-mic0: I have 240 cores
omp_get_num_threads=2

26.-28.6.2017
Intel MIC Programming Workshop @ LRZ
Intel Xeon Phi Programming Models: Intel “Mine Yours Ours” (MYO) virtual shared memory model
● “Mine Yours Ours” virtual shared memory model.
● Alternative to Offload approach. Only available in C++.
● Allows to share not bit-wise compatible complex data (like structures with pointer elements, C++ classes) without data marshalling. LEO Offload Model only allows offloading of bitwise-copyable data!
● Allocation of data at the same virtual addresses on the host and the coprocessor.
● Runtime automatically maintains coherence.
● Syntax based on the keywords `__Cilk_shared` and `__Cilk_offload`. 
#define N 10000
__Cilk_shared int a[N], b[N], c[N];

__Cilk_shared void add() {
    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];
}

int main(int argc, char *argv[]) {
    ...
__Cilk_offload add(); // Function call on coprocessor:
    ...
}
<table>
<thead>
<tr>
<th>Entity</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
<td>int _Cilk_shared f(\text{int}\ \text{x}){\ldots}</td>
<td>Executable code for both host and MIC; may be called from either side</td>
</tr>
<tr>
<td><strong>Global variable</strong></td>
<td>_Cilk_shared int \text{x} = 0</td>
<td>Visible on both sides</td>
</tr>
<tr>
<td><strong>File/Function static</strong></td>
<td>static _Cilk_shared int \text{x}</td>
<td>Visible on both sides, only to code within the file/function</td>
</tr>
<tr>
<td><strong>Class</strong></td>
<td>class _Cilk_shared \text{x} {\ldots}</td>
<td>Class methods, members, and operators are available on both sides</td>
</tr>
<tr>
<td><strong>Pointer to shared data</strong></td>
<td>int _Cilk_shared *\text{p}</td>
<td>\text{p} is local (not shared), can point to shared data</td>
</tr>
<tr>
<td><strong>A shared pointer</strong></td>
<td>int *_Cilk_shared \text{p}</td>
<td>\text{p} is shared, should only point at shared data</td>
</tr>
<tr>
<td><strong>Offloading a function call</strong></td>
<td>\text{x} = _Cilk_offload \text{func}(\text{y})</td>
<td>\text{func} executes on MIC if possible</td>
</tr>
<tr>
<td></td>
<td>\text{x} = _Cilk_offload_to(\text{n}) \text{func}</td>
<td>\text{func} must be executed on specified (n-th) MIC</td>
</tr>
<tr>
<td><strong>Offloading asynchronously</strong></td>
<td>_Cilk_spawn _Cilk_offload \text{func}(\text{y})</td>
<td>Non-blocking offload</td>
</tr>
<tr>
<td><strong>Offload a parallel for-loop</strong></td>
<td>_Cilk_offload _Cilk_for(i=0; i&lt;N; i++) {\ldots}</td>
<td>Loop executes in parallel on MIC</td>
</tr>
</tbody>
</table>
Xeon Phi References

- **Books:**
  - *Parallel Programming and Optimization with Intel Xeon Phi Coprocessors*, Colfax 2013  

- Intel Xeon Phi Programming, Training material, CAPS
- Intel Training Material and Webinars
- V. Weinberg (Editor) et al., *Best Practice Guide - Intel Xeon Phi*,  
Acknowledgements

- IT4Innovation, Ostrava.
- Partnership for Advanced Computing in Europe (PRACE)
- Intel
- BMBF (Federal Ministry of Education and Research)
- Dr. Karl Fürlinger (LMU)
- J. Cazes, R. Evans, K. Milfeld, C. Proctor (TACC)
- Adrian Jackson (EPCC)
Thank you for your participation!