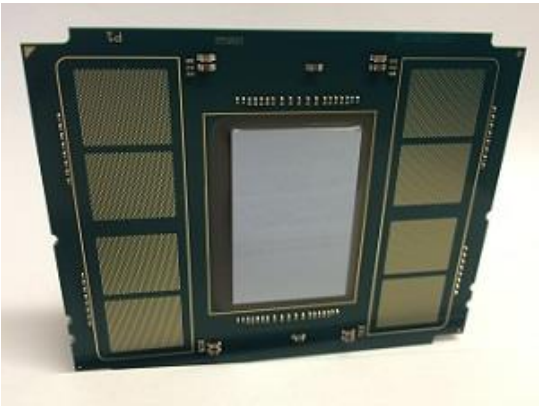
Code for Speed with High Bandwidth Memory on Intel® Xeon Phi™ Processors

Ruchira Sasanka, Karthik Raman

Development Tools Webinars

October 11th , 2016



Legal

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice.

All products, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.
Intel processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Any code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user.

Intel product plans in this presentation do not constitute Intel plan of record product roadmaps. Please contact your Intel representative to obtain Intel’s current plan of record product roadmaps.

Performance claims: Software and workloads used in performance tests may have been optimized for performance only on Intel® microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.Intel.com/performance

Intel, the Intel logo, Intel Xeon Phi, VTune, and Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.

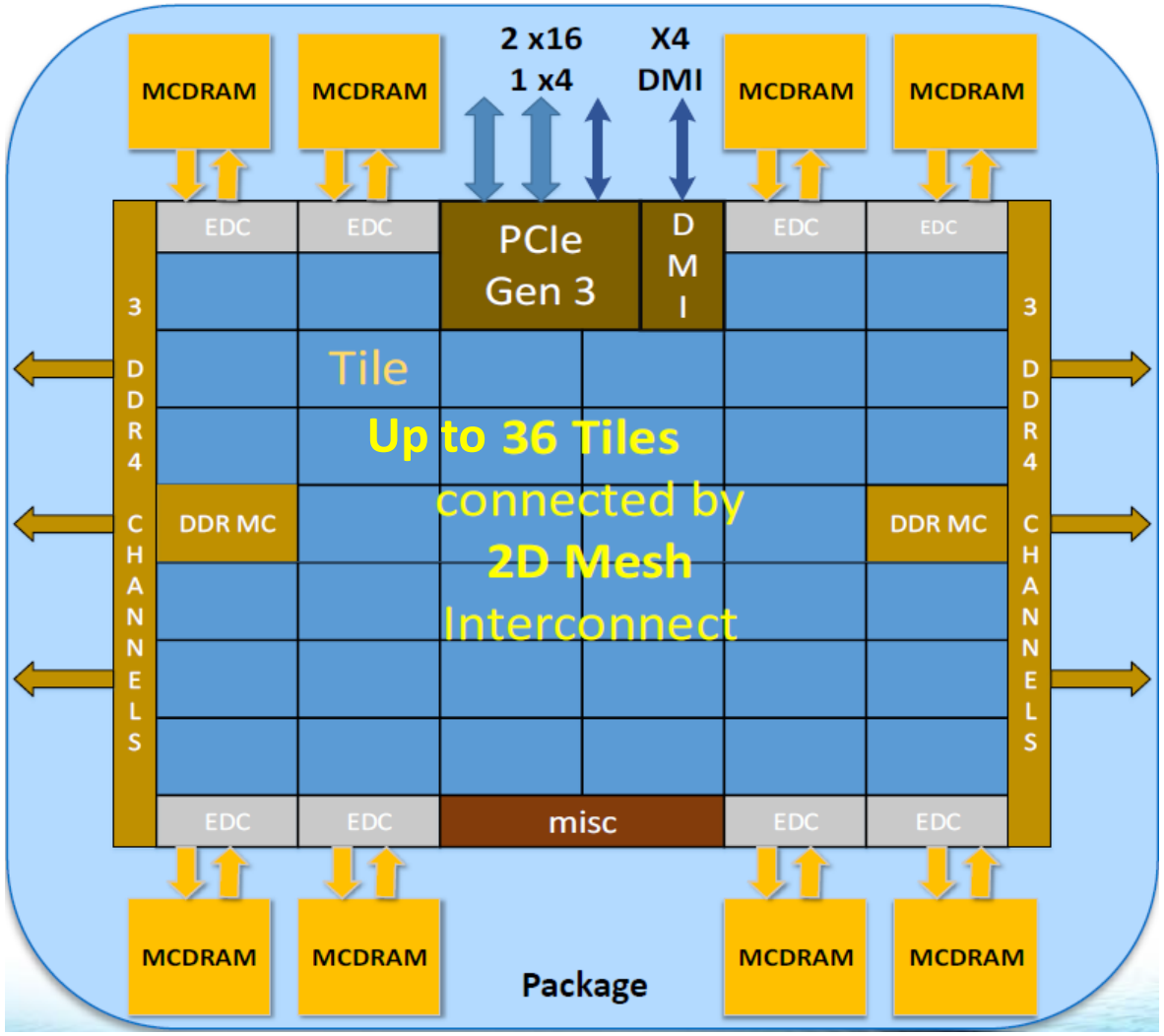
Acronyms

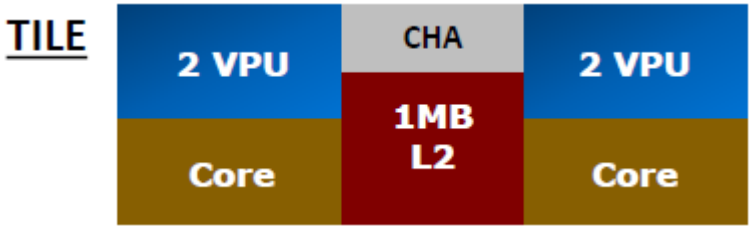
- BW : Bandwidth
- DDR : Double Data Rate (DRAM)
- Haswell (HSW) : Intel® Xeon® processor E5-2697v3 Family (code-named Haswell)
- Broadwell (BDW) : Intel® Xeon® processor E5-2697v4 Family (code-named Broadwell)
- KNL : 2nd generation Intel® Xeon Phi™ processor (code-named Knights Landing)
- MCDRAM : Multi-Channel DRAM (High-bandwidth memory/HBM)
- SNC : Sub Numa Clustering (one of the KNL cluster modes)
- NUMA : Non-Uniform Memory Access
- RHEL : Red Hat Enterprise Linux*

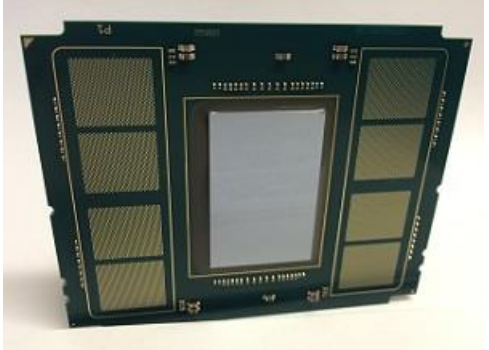
Agenda

- What is MCDRAM?
  - Introduction to KNL and MCDRAM Modes
- Does my application need MCDRAM?
  - Finding BW intensive code/data structures using Intel® VTune™ Amplifier
- How do I allocate data structures in MCDRAM?
  - Using numactl, memkind/AutoHBW libraries
- How do I test my MCDRAM-enabled apps?
  - Evaluate performance on KNL system
- Summary

KNL Overview







Chip: Up to 36 Tiles interconnected by **2D Mesh**
Tile: 2 Cores + 2 VPU/core + 1 MB L2

Memory: MCDRAM: 16 GB on-package; High BW
DDR4: 6 channels @ 2400 up to 384 GB
IO: 36 lanes PCIe* Gen3. 4 lanes of DMI for chipset
Node: 1-Socket only
Fabric: Intel® Omni-Path Architecture on-package (not shown)

Vector Peak Perf: 3+TF DP and 6+TF SP Flops
Scalar Perf: ~3x over Knights Corner
Streams Triad (GB/s): MCDRAM : 400+; DDR: 90+

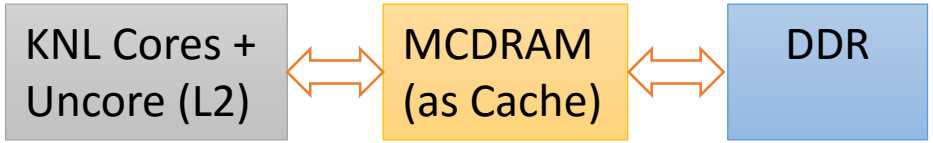
MCDRAM ~5X Higher BW than DDR

- Source Intel: All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice. KNL data are preliminary based on current expectations and are subject to change without notice. 1Binary Compatible with Intel Xeon processors using Haswell Instruction Set (except TSX). 2Bandwidth numbers are based on STREAM-like memory access pattern when MCDRAM used as flat memory. Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance. *Other names and brands may be claimed as the property of others.

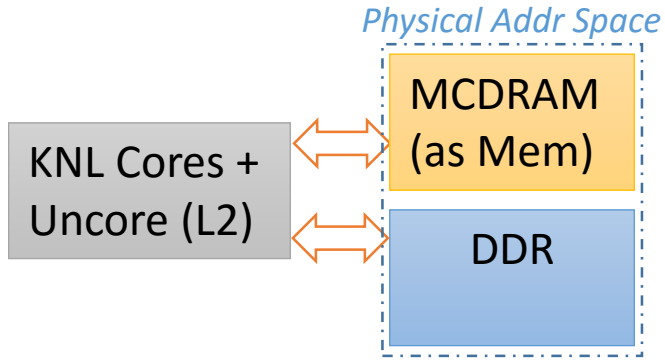
More Info: https://software.intel.com/en-us/articles/what-disclosures-has-intel-made-about-knights-landing

MCDRAM Modes

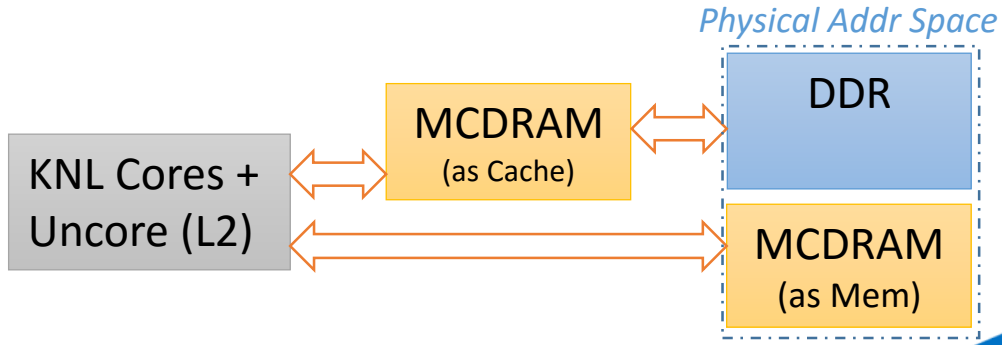
- Cache mode
  - No source changes needed to use
  - Misses are expensive (higher latency)
    - Needs MCDRAM access + DDR access



- Flat mode
  - MCDRAM mapped to physical address space
  - Exposed as a NUMA node
    - Use numactl --hardware, lscpu to display configuration
  - Accessed through memkind library or numactl

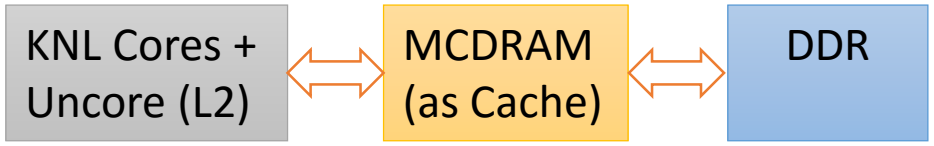


- Hybrid
  - Combination of the above two
    - E.g., 8 GB in cache + 8 GB in Flat Mode



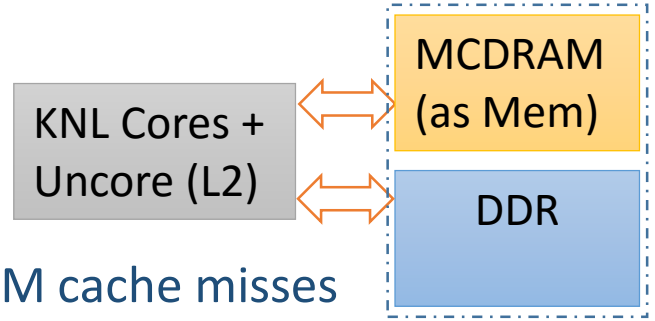
MCDRAM as Cache

- Upside
  - No software modifications required
  - Bandwidth benefit (over DDR)
- Downside
  - Higher latency for DDR access
    - i.e., for cache misses
  - Misses limited by DDR BW
  - All memory is transferred as:
    - DDR -> MCDRAM -> L2
    - Memory side cache
  - Less addressable memory
  - Conflicts due to direct mapped cache



MCDRAM as Flat Mode

- Upside
  - Maximum BW
  - Lower latency
    - i.e., no MCDRAM cache misses
  - Maximum addressable memory
  - Isolation of MCDRAM for high-performance application use only
- Downside
  - Software modifications (or interposer library) required
    - to use DDR and MCDRAM in the same app
  - Which data structures should go where?
  - MCDRAM is a finite resource and tracking it adds complexity



Agenda

- What is MCDRAM?
  - Introduction to KNL and MCDRAM Modes
- Does my application need MCDRAM?
  - Finding BW intensive code/data structures using Intel® VTune™ Amplifier
- How do I allocate data structures in MCDRAM?
  - Using numactl, memkind/AutoHBW libraries
- How do I test my MCDRAM-enabled apps?
  - Evaluate performance on KNL system

Intel® VTune™ Amplifier Memory Access Analysis

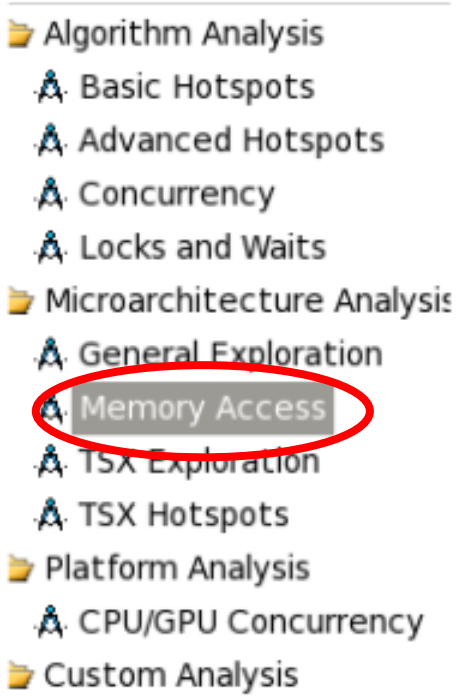
- Intel® VTune™ Amplifier introduces new analysis type to find memory related issues:
  - Memory bandwidth characteristics of an application (including QPI bandwidth – for Intel Xeon® processors)
  - Memory object analysis for KNL MCDRAM
- Memory Object analysis
  - Detects dynamic and static memory objects (allocated on heap and stack)
  - Attributes performance events to memory objects (arrays/data structures)
    - Helps to identify suitable candidates for KNL MCDRAM allocation
- Available starting with Intel® VTune™ Amplifier XE 2016
  - Use the latest Intel® VTune™ Amplifier XE 2017

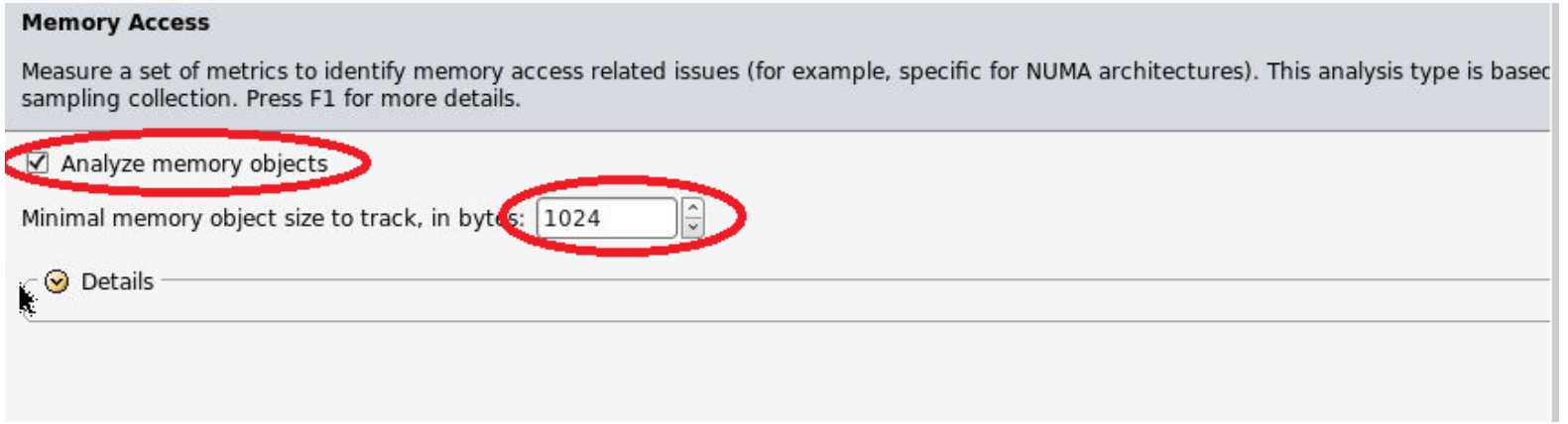
Instructions for Data Collection

- Linux* command line (on Knights Landing):

```
amplxe-cl -collect memory-access -data-limit=0 -r <result-dir> -knob analyze-mem-objects=true -knob mem-object-size-min-thres=1024 ./BlackScholesSP.knl
```

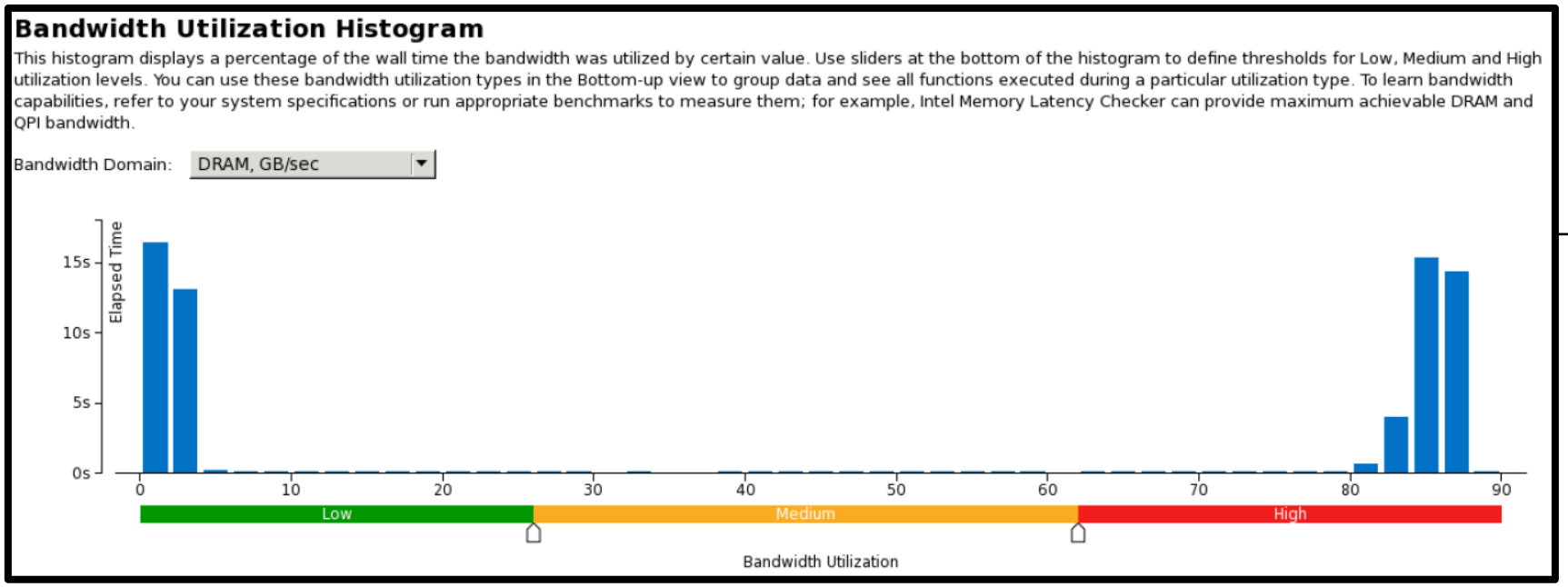
- Using Intel® VTune™ Amplifier GUI:
  - In **Analysis Type** tab ; under “Microarchitecture Analysis” menu:





*Other names and brands may be claimed as the property of others.

Summary View: Bandwidth Histogram

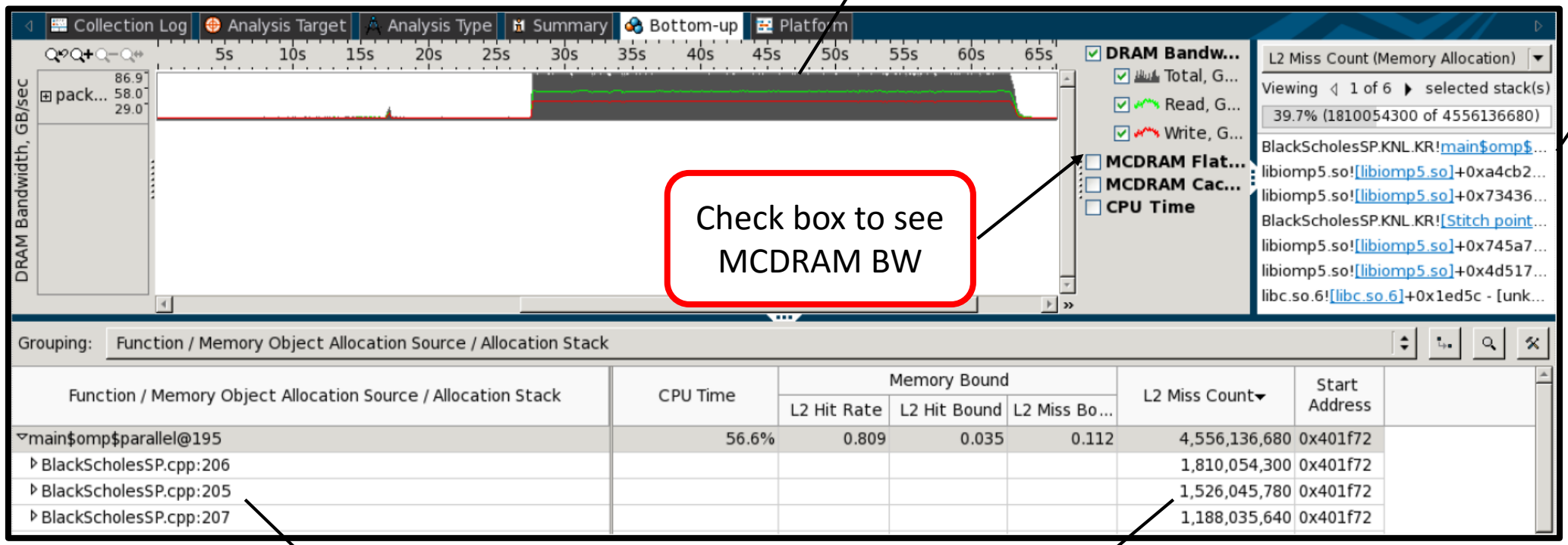


Bandwidth Histogram: Shows amount of wall time (y-axis) the bandwidth was utilized (x-axis) by a certain value for your application

Bottom-Up View

Time-line view of BW utilization

Memory allocation call stack



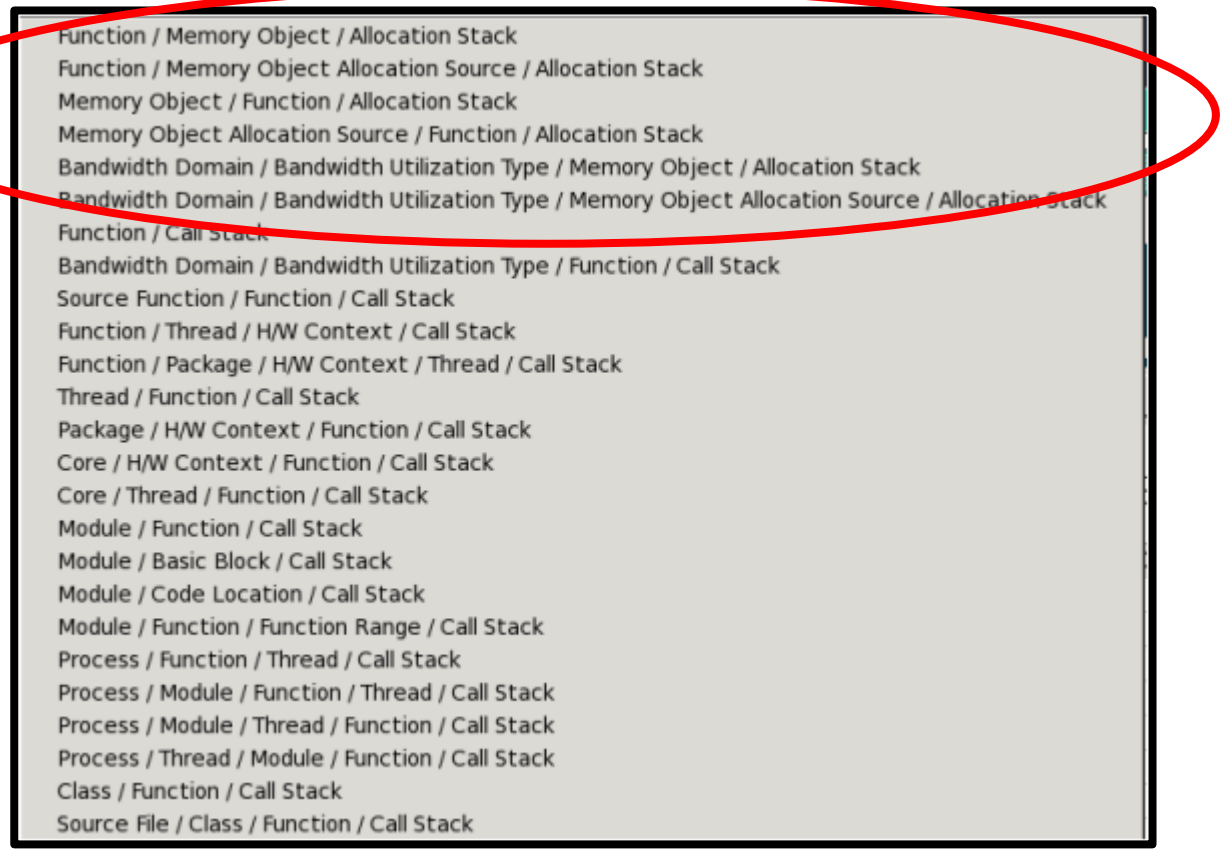
Check box to see MCDRAM BW

Memory objects sorted per function with corresponding allocation source lines and size

Performance counters and metrics to identify functions facing memory problems

View: Memory Object Grouping

New set of Groupings containing Memory Object



- Select the “Grouping” tab in the middle of the “Bottom Up” view to see the list of available groupings

View: Memory Objects

Grouping: Function / Memory Object Allocation Source / Allocation Stack

Function / Memory Object Allocation Source / Allocation Stack	CPU Time	Memory Bound			L2 Miss Count
		L2 Hit Rate	L2 Hit Bound	L2 Miss Bo...	
mainompparallel@195	56.6%	0.809	0.035	0.112	4,556,136,680
BlackScholesSP.cpp:206					1,810,054,300
BlackScholesSP.cpp:205					1,526,045,780
BlackScholesSP.cpp:207					1,188,035,640

```
205         posix_memalign((void **)&StockPrice, SIMDALIGN, mem_size);
206 >       posix_memalign((void **)&OptionStrike, SIMDALIGN, mem_size);
207         posix_memalign((void **)&OptionYears, SIMDALIGN, mem_size);
```

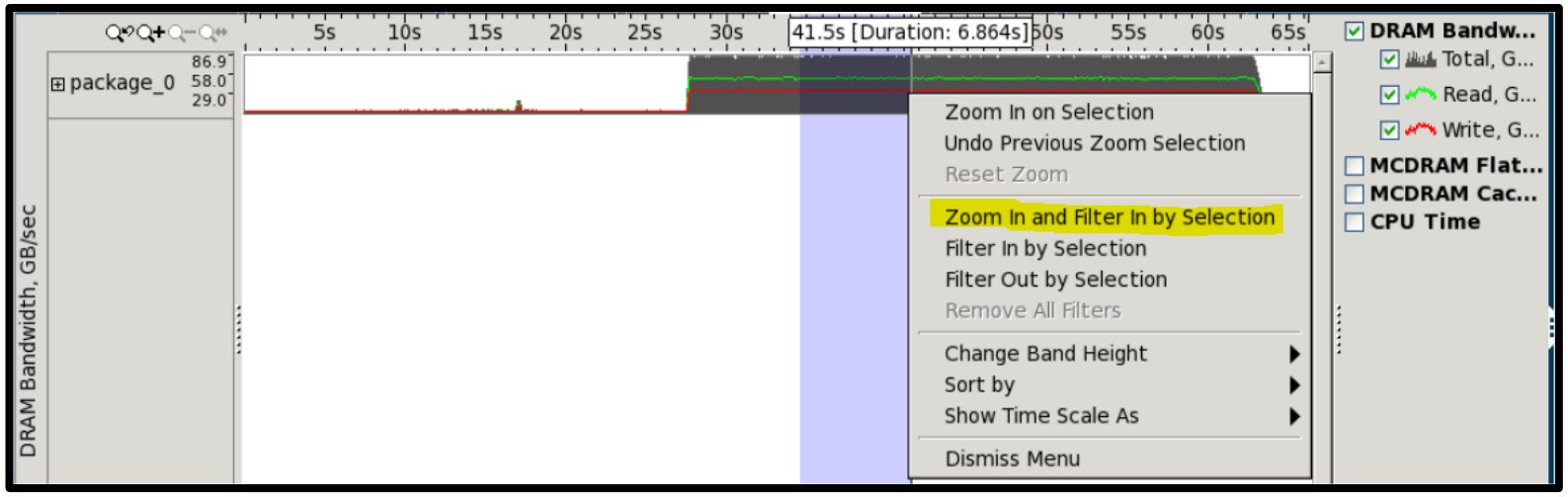
- Memory objects are identified by allocation source line and call stack
- Double-clicking on a memory object brings up the source line where malloc/allocate was called or where global variable was defined

View: Metrics

Grouping: Function / Memory Object Allocation Source / Allocation Stack

Function / Memory Object Allocation Source / Allocation Stack	CPU Time	Memory Bound			L2 Miss Count
		L2 Hit Rate	L2 Hit Bound	L2 Miss Bo...	
mainompparallel@195	56.6%	0.809	0.035	0.112	4,556,136,680
BlackScholesSP.cpp:206					1,810,054,300
BlackScholesSP.cpp:205					1,526,045,780
BlackScholesSP.cpp:207					1,188,035,640

- **Performance metrics**
  - CPU time/memory bound metrics used to identify functions with memory issues
  - L2 Miss Count can be used to characterize/sort memory objects



- **Bandwidth Utilization**
  - Users can select a region with high bandwidth utilization
  - Zoom In and Filter In updates the function/memory object profile
  - Can be used to identify memory objects attributing to high BW

Typical KNL MCDRAM Analysis Workflow

Note: This can be currently done on Haswell/Broadwell systems as well

- Collect "memory-access" data using VTune on KNL (running out of DDR)
- Select *'Function/Memory Object Allocation Source/Allocation Stack'* grouping OR

  'Bandwidth Domain/Bandwidth Utilization Type/Memory Object Allocation Source/Allocation Stack'
- In the bottom-up view, zoom in and filter by high-BW regions
- Observe the memory objects accessed by the functions
  - Sort the memory objects by L2 (or Last Level Cache) Misses
  - Most referenced memory objects in high bandwidth functions are potentially BW limited
- Select memory objects for allocating in KNL MCDRAM based on above analysis
  - Next step is to allocate high-BW memory objects using the HBW API's/Fortran attributes

Current Limitations

- Stack allocated memory
  - Currently stack allocations are denoted as “Unknown”
  - Users can drill down to Source Lines to understand which variables are accessed
  - Filtering can be used to separate unresolved memory objects
    - stack allocations versus heap allocations
- Stores to memory
  - KNL does not have precise store events
    - No DLA (Data Linear Address) associated with stores
  - Alternatively, HSW/BDW can be used to track memory objects associated to stores
- Memory object instrumentations currently available only on Linux*

Agenda

- What is MCDRAM?
  - Introduction to KNL and MCDRAM Modes
- Does my application need MCDRAM?
  - Finding BW intensive code/data structures using Intel® VTune™ Amplifier
- How do I allocate data structures in MCDRAM?
  - Using numactl, memkind/AutoHBW libraries
- How do I test my MCDRAM-enabled apps?
  - Evaluate performance on KNL system

Accessing MCDRAM in Flat Mode

- Option A: Using numactl
  - Works best if the whole app can fit in MCDRAM
- Option B: Using libraries
  - Memkind Library
    - Using library calls or Compiler Directives (Fortran*)
    - Needs source modification
  - AutoHBW (interposer library based on memkind)
    - No source modification needed (based on size of allocations)
    - No fine control over *individual* allocations
- Option C: Direct OS system calls
  - mmap(2), mbind(2), libnuma library (numa(3))
  - Not the preferred method
    - Page-only granularity, OS serialization, no pool management

Option A: Using numactl to Access MCDRAM

- MCDRAM is exposed to OS/software as a NUMA node
- Utility numactl is standard utility for NUMA system control
  - See "man numactl"
  - Do "numactl --hardware" to see the NUMA configuration of your system

```
available: 1 nodes (0)
node 0 cpus: 0 1 ... 286 287
node 0 size: 98200 MB
node 0 free: 91900 MB
node distances:
node   0
  0:  10
```

Cache Mode

```
available: 2 nodes (0-1)
node 0 cpus: 0 1 ... 270 271
node 0 size: 98200 MB
node 0 free: 91631 MB
node 1 cpus:
node 1 size: 16384 MB
node 1 free: 15927 MB
node distances:
node   0   1
  0:  10  31
  1:  31  10
```

Flat Mode

Option A : Using numactl to Access MCDRAM (contd.)

- If the total memory footprint of your app is smaller than the size of MCDRAM
  - Use numactl to allocate all of its memory from MCDRAM
  - numactl --membind=*mcdram_id* <*your_command*>
    - Where *mcdram_id* is the ID of MCDRAM "node"
  - Caution: If footprint > MCDRAM size and swapping enabled, app swaps to disk
- If the total memory footprint of your app is larger than the size of MCDRAM
  - You can still use numactl to allocate ***part*** of your app in MCDRAM
    - numactl --preferred=*mcdram_id* <*your_command*>
      - Allocations that don't fit into MCDRAM spills over to DDR
      - **Tip:** Touch high BW structures first to put them in MCDRAM (First Touch Policy)
        - This works even for stack variables
    - numactl --interleave=*nodes* <*your_command*>
      - Allocations are interleaved across all *nodes*

Option B.1: Using Memkind Library to Access MCDRAM

Allocate 1000 floats from DDR

```
float   *fv;

fv = (float *)malloc(sizeof(float) * 1000);
```

Allocate 1000 floats from MCDRAM

```
#include <hbwmalloc.h>

float   *fv;

fv = (float *)hbw_malloc(sizeof(float) * 1000);
```

Allocate arrays from MCDRAM and DDR in Intel® Fortran Compiler

```
c      Declare arrays to be dynamic
       REAL, ALLOCATABLE :: A(:), B(:), C(:)

!DEC$ ATTRIBUTES FASTMEM :: A

       NSIZE=1024
c
c      allocate array 'A' from MCDRAM
c
       ALLOCATE (A(1:NSIZE))
c
c      Allocate arrays that will come from DDR
c
       ALLOCATE  (B(NSIZE), C(NSIZE))
```

Option B.2: AutoHBW

- AutoHBW: Interposer Library that comes with memkind
  - Automatically allocates memory from MCDRAM
    - If a heap allocation (e.g., malloc/calloc) is larger than a given threshold
- Simplest way to experiment with MCDRAM memory is with AutoHBW library:
  - `LD_PRELOAD=libautohbw.so:libmemkind.so ./app`
- Environment variables (see autohbw_README)
  - `AUTO_HBW_SIZE=`*`x[:y]`*
    Any allocation larger than x and smaller than y should be allocated in HBW memory
  - `AUTO_HBW_MEM_TYPE=`*`memory_type`*
    Sets the "kind" of HBW memory that should be allocated (e.g. MEMKIND_HBW)
  - `AUTO_HBW_LOG=`*`level`*
    Extra Logging information
- Finding source locations of arrays
  - `export AUTO_HBW_LOG=2`
  - `./app_name >` *`log.txt`*
  - `autohbw_get_src_lines.pl` *`log.txt app_name`*

Obtaining Memkind Library

- Homepage: http://memkind.github.io/memkind
  - Join Mailing list: https://lists.01.org/mailman/listinfo/memkind
- Download package
  - On Fedora* 21 and above: `yum install memkind`
  - On RHEL* 7: `yum install epel-release; yum install memkind`
- Alternative(1), you can build from source
  - `git clone https://github.com/memkind/memkind.git`
  - See CONTRIBUTING file for build instructions
- Alternative(2), Download via XPPS (Intel® Xeon Phi™ Processor Software)
  - Download XPPS (https://software.intel.com/en-us/articles/xeon-phi-software)
  - Untar and install the "memkind" rpm from:
    `xppsl-<ver>/<OS-ver>/srpms/xppsl-memkind-<ver>.src.rpm`
  - Must use the memkind *src.rpm* to get AutoHBW library

Summary: Example Command Lines

1. **Cache mode**

```
available: 1 nodes (0)
node 0 cpus: 0 1 ... 286 287
node 0 size: 98200 MB
node 0 free: 91900 MB
node distances:
node   0
  0:  10
```

```
$ ./app
$ mpirun -np 72 ./app
```

Cache mode does not need additional SW support

Use "numactl –hardware" to get this output

2. **Flat mode**

```
available: 2 nodes (0-1)
node 0 cpus: 0 1 ... 270 271
node 0 size: 98200 MB
node 0 free: 91631 MB
node 1 cpus:
node 1 size: 16384 MB
node 1 free: 15927 MB
node distances:
node   0   1
  0:  10  31
  1:  31  10
```

```
$ mpirun -np 68 numactl -m 1 ./app              # MCDRAM BIND
$ mpirun -np 68 numactl --preferred=1 ./app     # MCDRAM PREFERRED

$ LD_PRELOAD=libautohbw.so:libmemkind.so ./app # AutoHBW
$ mpirun -np 68 ./app                           # DDR4 (default)
```

For the last case, the app should explicitly allocate high bandwidth data in MCDRAM, using memkind APIs and/or Fortran "fastmem" attributes

Agenda

- What is MCDRAM?
  - Introduction to KNL and MCDRAM Modes
- Does my application need MCDRAM?
  - Finding BW intensive code/data structures using Intel® VTune™ Amplifier
- How do I allocate data structures in MCDRAM?
  - Using numactl, memkind/AutoHBW libraries
- How do I test my MCDRAM-enabled apps?
  - Performance evaluation on KNL System

Observing MCDRAM Memory Allocations

- Where is MCDRAM usage printed?
  - `numastat –m`
    - Printed for each NUMA node
    - Includes Huge Pages info
  - `numastat -p <pid>` OR `numastat -p exec_name`
    - Info about process <pid>
    - E.g., `watch -n 1 numastat -p exec_name`
  - `cat /sys/devices/system/node/node*/meminfo`
    - Info about each NUMA node
  - `cat /proc/meminfo`
    - Aggregate info for system
    - Shows swap usage
- Utilities that provide MCDRAM node info
  - `<memkind_install_dir>/bin/memkind-hbw-nodes`
  - `numactl --hardware`
  - `lscpu`

MCDRAM Performance Evaluation on KNL

- Running BlackScholes SP using numactl

  % Compile: icpc -O3 BlackScholesSP.cpp -xMIC-AVX512 -qopenmp -fimf-precision=low -fimf-domain-exclusion=31 -no-prec-div -no-prec-sqrt -o BlackScholesSP.knl

  % Environment: export KMP_PLACE_THREADS=68C,4T; export KMP_AFFINITY="compact,granularity=thread"

  % Run: **numactl --membind=1** ./BlackScholesSP.knl

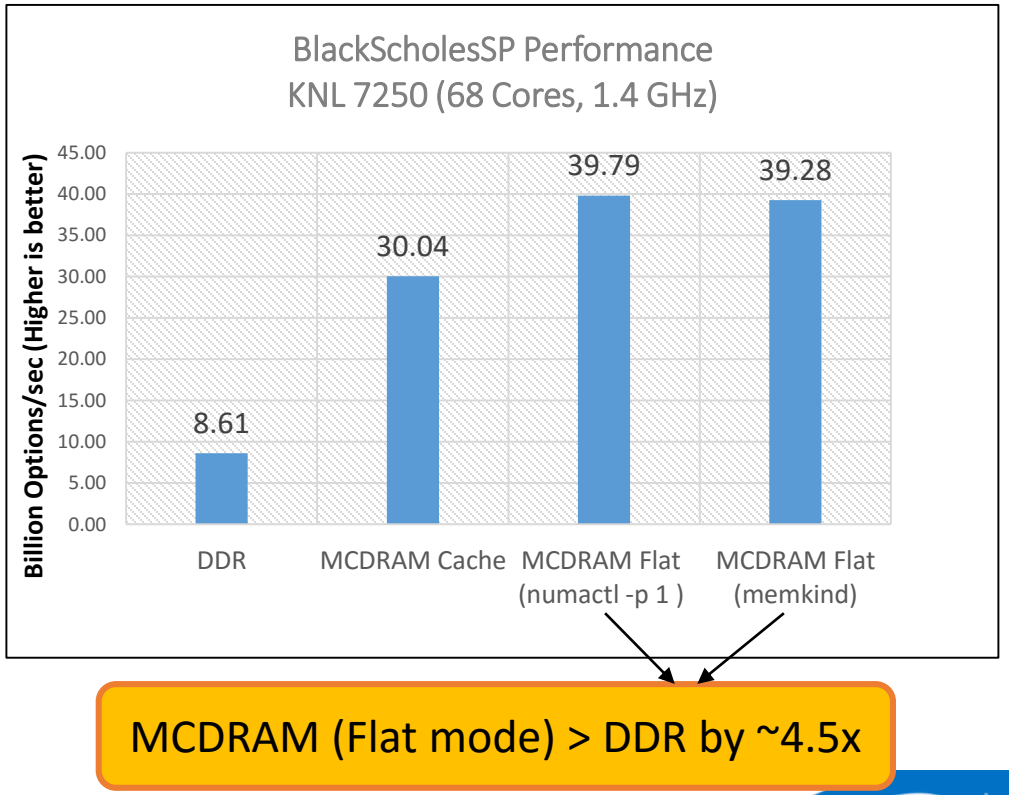
- Running BlackScholes SP using hbwmalloc

  % Same compiler flags as above

  Just add "**-lmemkind** (link with memkind library)"

  %./BlackscholesSP.knl.hbw

```
#pragma omp parallel reduction(+ : sum_delta) reduction(+ : sum_ref)
{
 …
   float *Call,  *Put, *Stock, *Strike, *Time;
   hbw_posix_memalign((void **)&Stock, ALIGN, mem_size);
   hbw_posix_memalign((void **)&Strike, ALIGN, msize);
   hbw_posix_memalign((void **)&Time, ALIGN, msize);
   hbw_posix_memalign((void **)&Call, ALIGN, msize);
   hbw_posix_memalign((void **)&Put, ALIGN, msize);



   // numerical pricing activities

   hbw_free(Call);
   hbw_free(Put);
   hbw_free(Stock);
   hbw_free(Strike);
   hbw_free(Time);
} //end of omp parallel section
```



New Intel® MPI Affinity Features

- Supports memory allocation of MPI processes to different memory types (DDR, MCDRAM…) exposed as distinct NUMA nodes
- **I_MPI_HBW_POLICY = <value>**

<value>	The memory allocation policy used.
hbw_preferred	Allocate the local HBW memory for each process. If the HBW memory is not available, allocate the local dynamic random access memory.
hbw_bind	Allocate only the local HBW memory for each process.
hbw_interleave	Allocate the HBW memory and dynamic random access memory on the local node in the round-robin manner.

- Old MPI execution command line:
  - `mpirun -n 16 numactl --preferred 1 ./app`
- New simplified command line:
  - `I_MPI_HBW_POLICY=hbw_preferred mpirun -n 16 ./app`
- These new affinity features (and more) are already available in Intel® MPI 2017 release

Summary: Your Options

- Do nothing
  - If DDR BW is sufficient for your app
    - Use Intel® VTune™ Amplifier to verify
- Use numactl to place app in MCDRAM
  - Works well if the entire app fits within MCDRAM
    - Use numastat/vmstat/top to observe memory footprint
  - Can use numactl --preferred if app does not fit completely in MCDRAM
- Use MCDRAM cache mode
  - Trivial to try; no source changes
- Use AutoHBW
  - Can try different parameters with low effort; no source changes
- Use memkind API
  - Use Intel® VTune™ Amplifier to identify high-BW structures

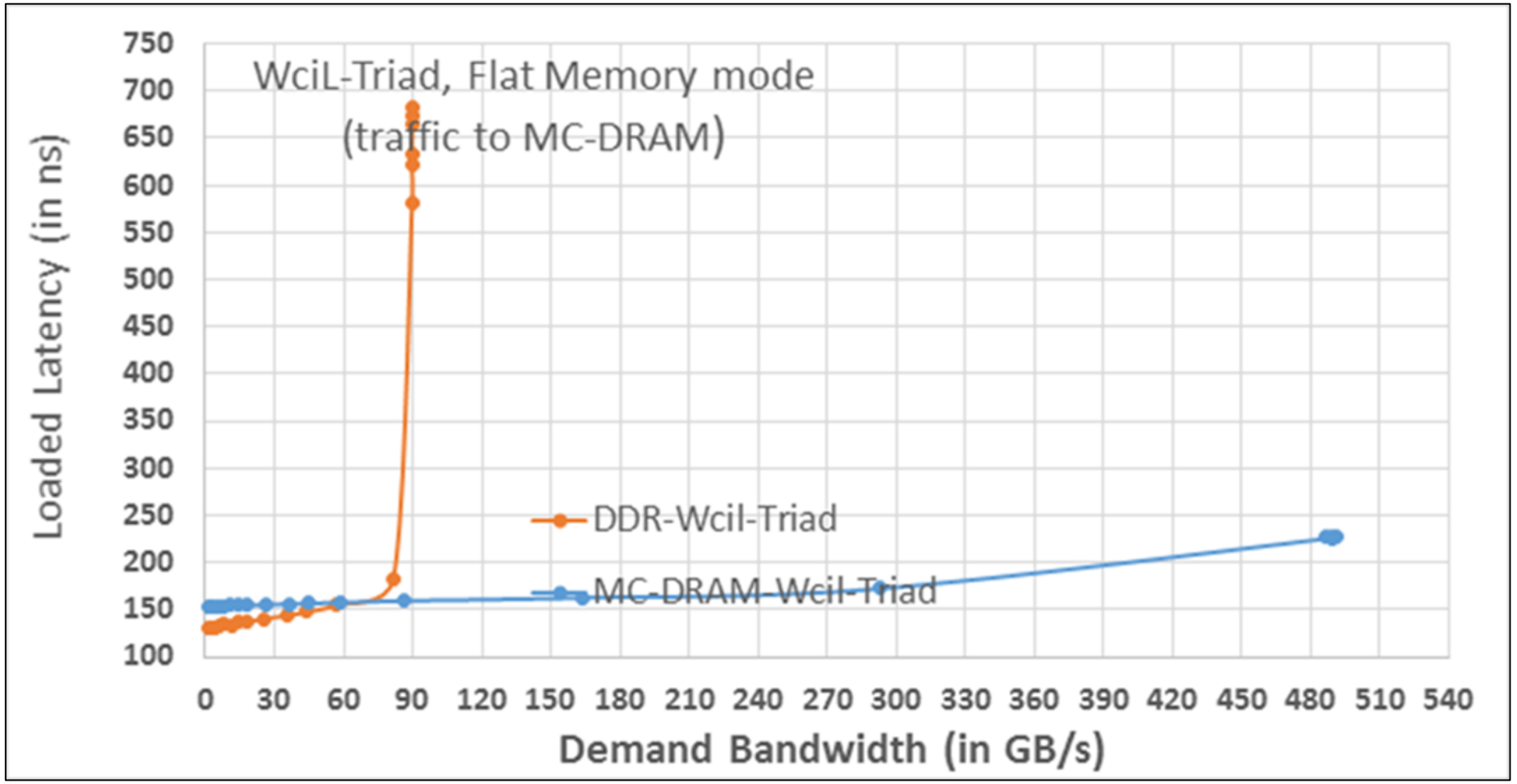
Thank You!

We would like to:

- Acknowledge the many **Intel contributors** for their content and valuable feedback
- Thank all the **Registrants** (i.e. YOU ☺) for attending this talk

BACKUP

MCDRAM vs. DDR: latency vs. bandwidth



Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you purchases, including the performance of that product when combined with other products. KNL results measured on pre-production parts. Any difference in system hardware or software design or configuration may affect actual performance. For more information go to http://www.intel.com/performance

hbw and memkind APIs

- See "man hbwmalloc"

```
int hbw_check_available(void);
void* hbw_malloc(size_t size);
void* hbw_calloc(size_t nmemb, size_t size);
void* hbw_realloc (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, int pagesize);
int hbw_get_policy(void);
void hbw_set_policy(int mode);
```

- See "man memkind" for memkind API

```
void *memkind_malloc(memkind_t kind, size_t size);
void *memkind_calloc(memkind_t kind, size_t num, size_t size);
void *memkind_realloc(memkind_t kind, void *ptr, size_t size);
int memkind_posix_memalign(memkind_t kind, void **memptr, size_t alignment, size_t size);
void memkind_free(memkind_t kind, void *ptr);
```

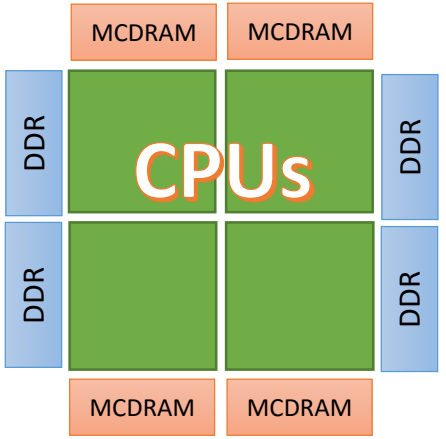
Notes: (1) hbw_* APIs call memkind APIs. (2) Only part of memkind API shown above

Memkind Policies and Memory Types

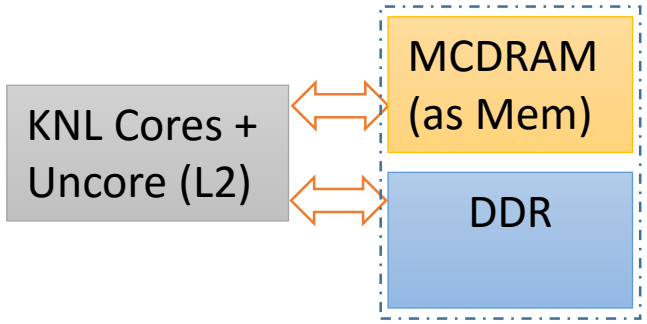
- How do we make sure we get memory only from MCDRAM?
  - This depends on POLICY
    - See man page (man hbwmalloc) and hbw_set_policy() / hbw_get_policy()
  - HBW_POLICY_BIND : Will cause app to die when it runs out of MCDRAM (and swap if enabled)
  - HBW_POLICY_PREFERRED : Will allocate from DDR if MCDRAM not sufficient (default)
- Allocating 2 MB and 1 GB pages
  - Use hbw_posix_memalign_psize()
- Similarly, many "kinds" of memory supported by memkind (see man page: man memkind)
  - MEMKIND_DEFAULT
    - Default allocation using standard memory and default page size.
  - MEMKIND_HBW
    - Allocate from the closest high-bandwidth memory NUMA node at time of allocation.
  - MEMKIND_HBW_PREFERRED
    - If there is not enough HBW memory to satisfy the request, fall back to standard memory.
  - MEMKIND_HUGETLB
    - Allocate using huge pages.
  - MEMKIND_GBTLB
    - Allocate using GB huge pages.
  - MEMKIND_INTERLEAVE
    - Allocate pages interleaved across all NUMA nodes.
  - MEMKIND_PMEM
    - Allocate from file-backed heap.

These can all be used with HBW (e.g. MEMKIND_HBW_HUGETLB); all but INTERLEAVE can be used with HBW_PREFERRED.

Advanced Topic: MCDRAM in SNC4 Mode



- SNC4: Sub-NUMA Clustering
  - KNL die is divided into 4 clusters (similar to a 4-Socket Xeon)
    - SNC4 configured at boot time
    - Use numactl --hardware to find out nodes and distances
      - There are **4 DDR (+CPU) nodes + 4 MCDRAM (no CPU) nodes**, in flat mode



Compare with 2 NUMA nodes

- Running 4-MPI ranks is the easiest way to utilize SNC4
  - Each rank allocates from closest DDR node
    - If a rank allocates MCDRAM, it goes to closest MCDRAM node
- If you run only 1 MPI rank and use numactl to allocate on MCDRAM
  - Specify all MCDRAM nodes
    - E.g., numactl –m 4,5,6,7

New Intel® MPI Affinity Features

- Supports memory allocation of MPI processes to different memory types (DDR, MCDRAM…) exposed as distinct NUMA nodes
- **I_MPI_HBW_POLICY = <value>**

<value>	The memory allocation policy used.
hbw_preferred	Allocate the local HBW memory for each process. If the HBW memory is not available, allocate the local dynamic random access memory.
hbw_bind	Allocate only the local HBW memory for each process.
hbw_interleave	Allocate the HBW memory and dynamic random access memory on the local node in the round-robin manner.

- Old MPI execution command line:
  - *mpirun -perhost 16 -n 4 numactl --preferred 4 ./app : -n 4 numactl --preferred 5 ./app : -n 4 numactl --preferred 6 ./app : -n 4 numactl --preferred 7 ./app*
- New simplified command line:
  - **I_MPI_HBW_POLICY=hbw_preferred mpirun –perhost 16 ./app**
- These new affinity features (and more) are already available in Intel® MPI 2017 release

More KNL Info

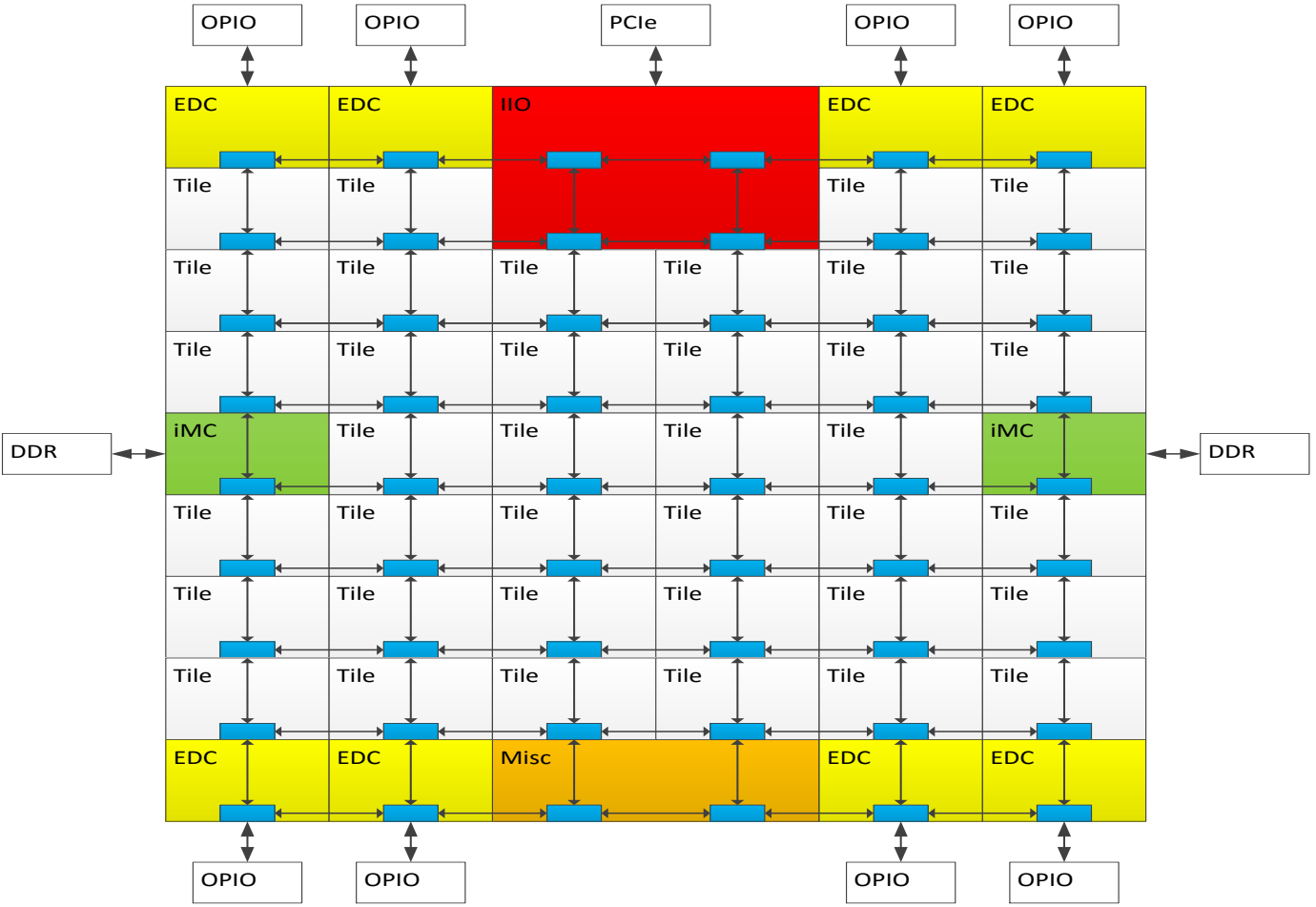
- Intel® Xeon Phi™ processor related documents, software tools, recipe
  - https://software.intel.com/xeon-phi/x200-processor

SNC4 NUMA Distances

- Nodes 0-3 have both CPUs and DDR. Nodes 4-7 have only MCDRAM (no CPUs).
- node distances:

node	0	1	2	3	4	5	6	7
0:	10	21	21	21	31	41	41	41
1:	21	10	21	21	41	31	41	41
2:	21	21	10	21	41	41	31	41
3:	21	21	21	10	41	41	41	31
4:	41	31	41	41	10	41	41	41
5:	41	41	31	41	41	10	41	41
6:	41	41	41	31	41	41	10	41
7:	31	41	41	41	41	41	41	10

KNL Mesh Interconnect



Mesh of Rings

- Every row and column is a (half) ring
- YX routing: Go in Y → Turn → Go in X
- Messages arbitrate at injection and on turn

Cache Coherent Interconnect

- MESIF protocol (F = Forward)
- Distributed directory to filter snoops

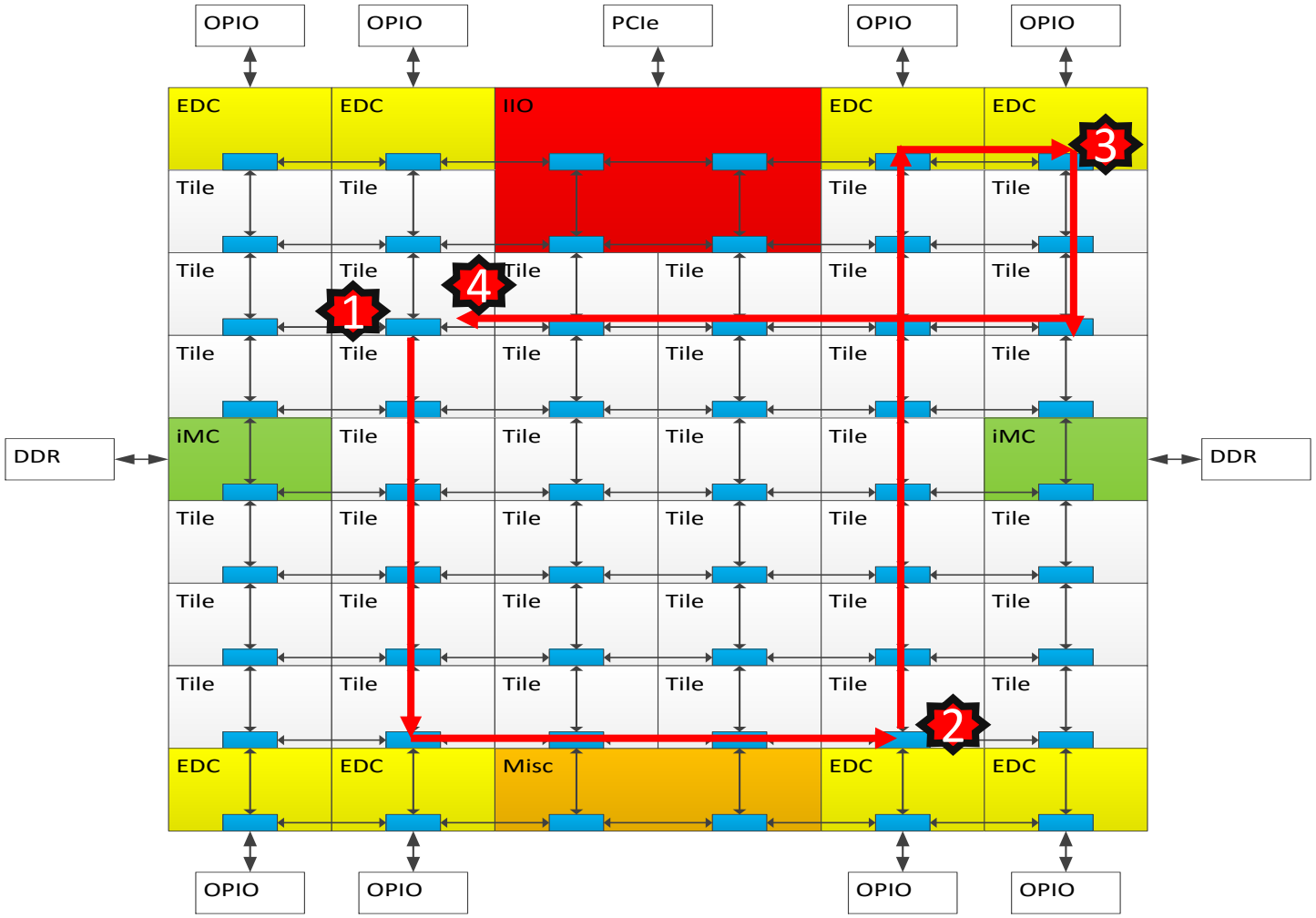
Three Cluster Modes

(1) All-to-All

(2) Quadrant

(3) Sub-NUMA Clustering (SNC)

Cluster Mode: All-to-All



Address uniformly hashed across all distributed directories

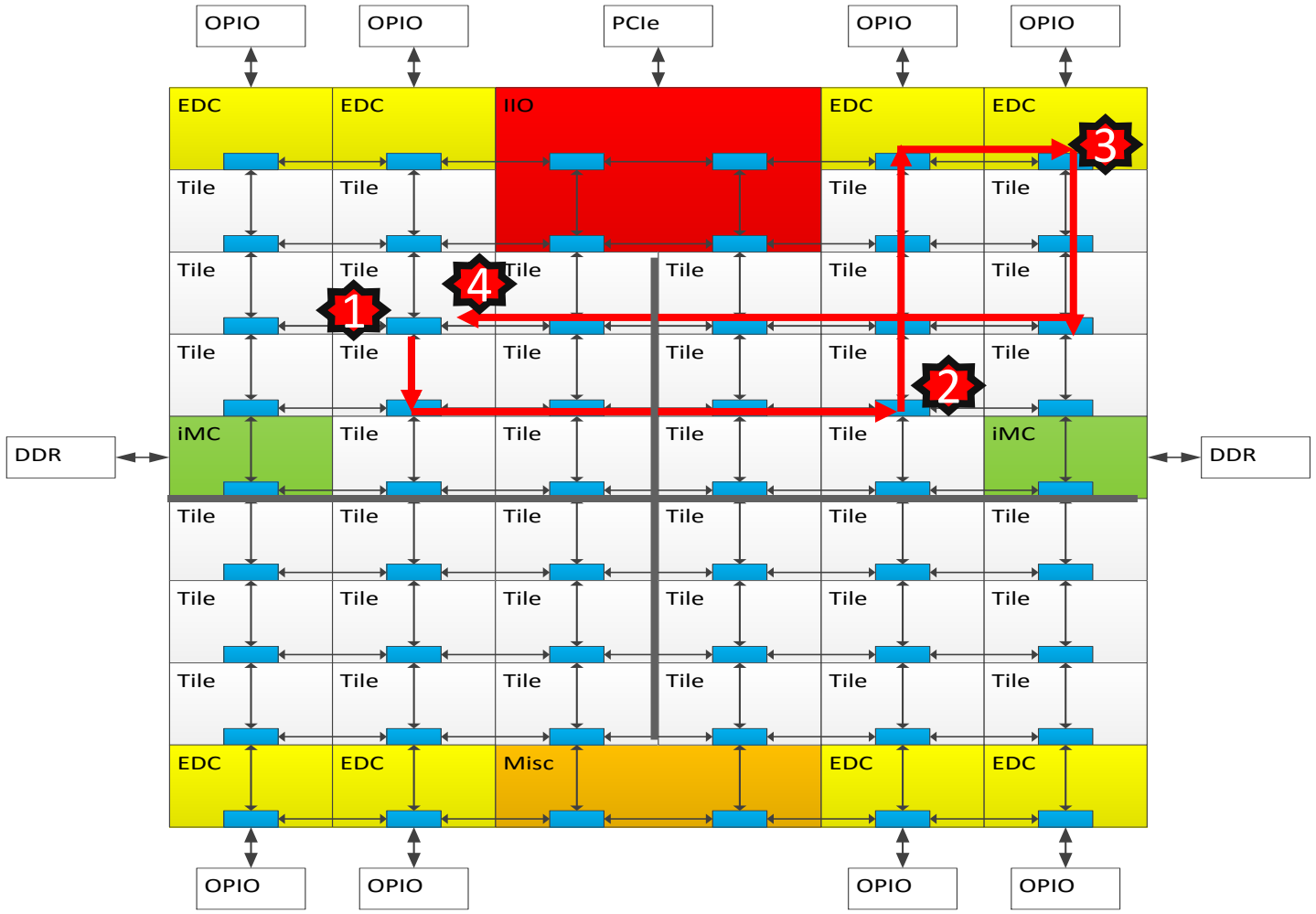
No affinity between Tile, Directory and Memory

Lower performance mode, compared to other modes. Mainly for fall-back

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

Cluster Mode: Quadrant



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

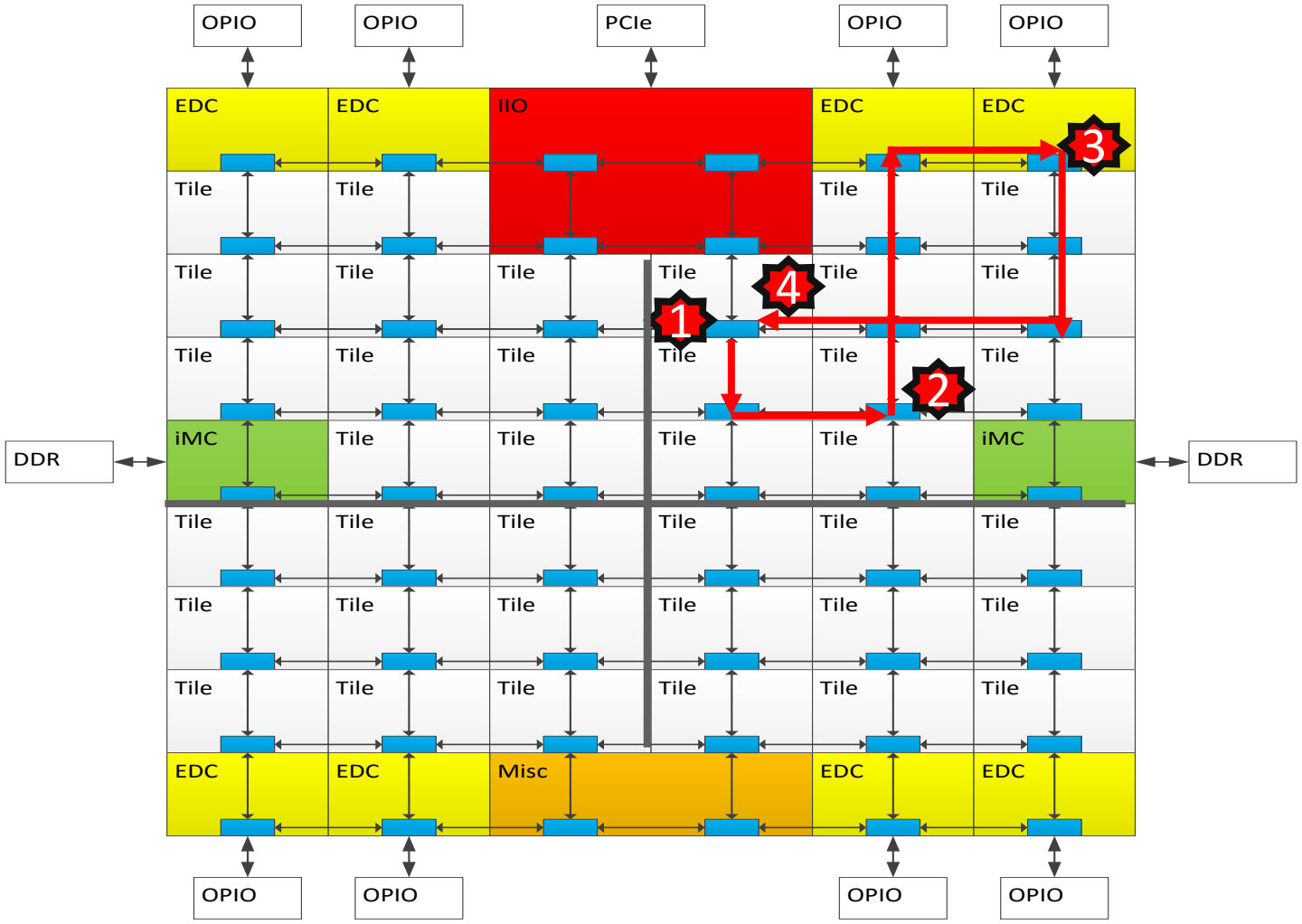
Chip divided into four virtual Quadrants

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. Software transparent.

Cluster Mode: Sub-NUMA Clustering (SNC)



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

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

Looks analogous to 4-Socket Xeon

Affinity between Tile, Directory and Memory

Local communication. Lowest latency of all modes

Software needs to be NUMA-aware to get benefit

Software visible memory configuration (numactl –-hardware) w/ different clustering modes

1. **Cache mode** / Quadrant

```
available: 1 nodes (0)
node 0 cpus: 0 1 ... 286 287
node 0 size: 98200 MB
node 0 free: 91900 MB
node distances:
node   0
  0:  10
```

2. **Flat mode** / Quadrant

```
available: 2 nodes (0-1)
node 0 cpus: 0 1 ... 270 271
node 0 size: 98200 MB
node 0 free: 91631 MB
node 1 cpus:
node 1 size: 16384 MB
node 1 free: 15927 MB
node distances:
node   0   1
  0:  10  31
  1:  31  10
```

3. Cache mode / **SNC-4**

```
available: 4 nodes (0-3)
node 0 cpus: 0 1 .. 220 221
node 0 size: 23921 MB
node 1 cpus: 18 19 .. 238 239
node 1 size: 24231 MB
node 2 cpus: 36 37 .. 254 255
node 2 size: 24232 MB
node 3 cpus: 52 53 .. 270 271
node 3 size: 24229 MB
node distances:
node   0   1   2   3
  0:  10  21  21  21
  1:  21  10  21  21
  2:  21  21  10  21
  3:  21  21  21  10
```

4. Flat mode with **sub-NUMA clustering (SNC-4)**

```
available: 8 nodes (0-7)
node 0 cpus: 0 1 .. 220 221
node 0 size: 23922 MB
node 1 cpus: 18 19 .. 238 239
node 1 size: 24231 MB
node 2 cpus: 36 37 .. 254 255
node 2 size: 24232 MB
node 3 cpus: 52 53 .. 270 271
node 3 size: 24232 MB
node 4 cpus:
node 4 size: 4039 MB
node 5 cpus:
node 5 size: 4039 MB
node 6 cpus:
node 6 size: 4039 MB
node 7 cpus:
node 7 size: 4036 MB
node distances:
node   0   1   2   3   4   5   6   7
  0:  10  21  21  21  31  41  41  41
  1:  21  10  21  21  41  31  41  41
  2:  21  21  10  21  41  41  31  41
  3:  21  21  21  10  41  41  41  31
  4:  31  41  41  41  10  41  41  41
  5:  41  31  41  41  41  10  41  41
  6:  41  41  31  41  41  41  10  41
  7:  41  41  41  31  41  41  41  10
```

memkind Interface

```
MEMKIND(3) -- 2014-09-22 -- Intel Corporation -- MEMKIND

NAME
     memkind -  Heap manager  that enables allocations  to memory
     with different properties.

SYNOPSIS
     #include <memkind.h>

     Link with -ljemalloc -lnuma -lpthread -lmemkind

     void memkind_error_message(int err, char *msg, size_t size);

     HEAP MANAGEMENT:
     void *memkind_malloc(memkind_t kind, size_t size);
     void *memkind_calloc(memkind_t kind, size_t num, size_t size);
     void *memkind_realloc(memkind_t kind, void *ptr, size_t size);
     int memkind_posix_memalign(memkind_t kind, void **memptr, size_t alignment, size_t size);
     void memkind_free(memkind_t kind, void *ptr);
     int memkind_get_kind_for_free(void *ptr, memkind_t *kind);

     ALLOCATOR CALLBACK FUNCTIONS:
     int memkind_partition_check_available(int partition);
     int memkind_partition_get_mmap_flags(int partition, int *flags);
     int memkind_partition_mbind(int partition, void *addr, size_t len);

     KIND MANAGMENT:
     int memkind_create(const struct memkind_ops *ops, const char *name , memkind_t *kind);
     int memkind_finalize(void);
     int memkind_get_num_kind(int *num_kind);
     int memkind_get_kind_by_partition(int partition, memkind_t *kind);
     int memkind_get_kind_by_name(const char *name, memkind_t *kind);
     int memkind_get_size(memkind_t kind, size_t *total, size_t *free);
     int memkind_check_available(memkind_t kind);
```

Allocating Fortran Pointers to MCDRAM

```
integer, parameter :: N=600

Real(8), allocatable, target, dimension(:,:) :: A, B, C
Real(8), pointer,             dimension(:,:) :: p1, p2, p3

!DIR$ ATTRIBUTES FASTMEM :: A, B, C

Allocate (A(N,N), B(N,N), C(N,N))

p1 => A
p2 => B
p3 => C
```

This is **not** allowed:

```
!!DIR$ ATTRIBUTES FASTMEM :: p1          !  this isn't allowed
!!  Allocate (p1(N,N))
```

Allocating C++ STL to MCDRAM

```
#include "hbwmalloc.h"

…

int main()

{

size_t size = 10;

#ifdef MALLOC
     std::allocator<int> a1;
     std::vector<int> array(size);

#elif HBWMALLOC
     hbwmalloc::hbwmalloc_allocator<int> a1
     std::vector<int,hbwmalloc::hbwmalloc_allocator<int> > array(size);

#endif

….

}
```

*Other names and brands may be claimed as the property of others.

Check for MCDRAM Availability in Fortran

```
interface
  function hbw_check_available() result(avail) &
           bind(C,name='hbw_check_available')
    use iso_c_binding
    implicit none
    integer(C_INT) :: avail
  end function hbw_check_available
end interface

res = hbw_check_available()
if (res == 0) then
   write (*,'(A)') 'MCDRAM available'
else
   write (*,'(A,I0)') &
   'ERROR, MCDRAM not available, return code=', res
end if
```

Check for MCDRAM Size in Fortran/C

"hbw_get_size.c"

```
#include <memkind.h>
int hbw_get_size(int partition, size_t * total, size_t * free) {
  memkind_t kind;
  int stat;

  stat = memkind_get_kind_by_partition(partition, &kind);
  if(stat==0) stat = memkind_get_size(kind, total, free);
  return stat;
}
```

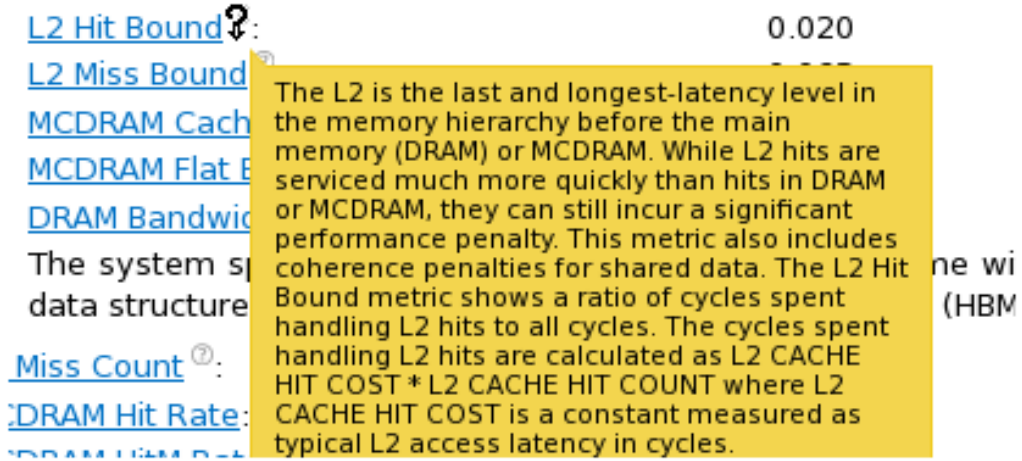
"Fortran Code" (Link the "hbw_get_size" code from above)

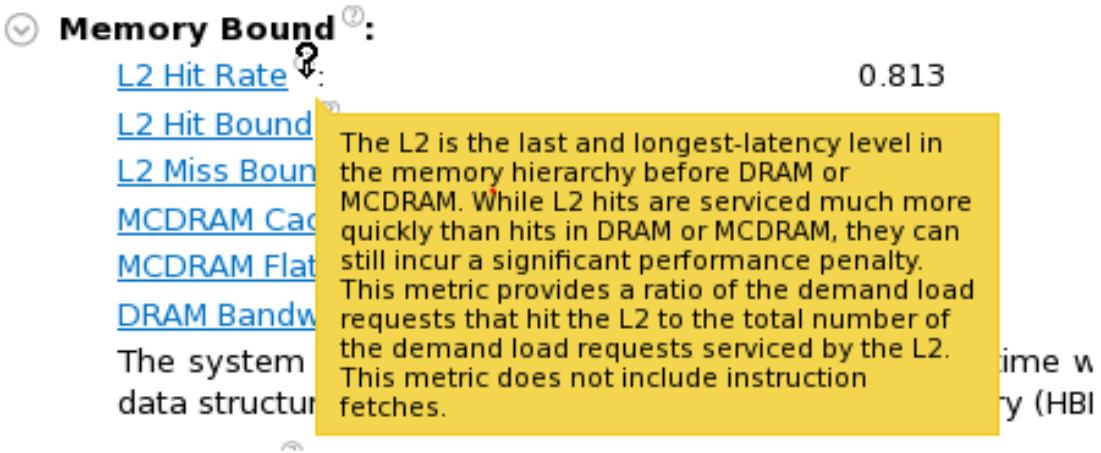
```
  integer(C_INT)     :: istat
  integer(C_SIZE_T)  :: total
  integer(C_SIZE_T)  :: free

interface
    function hbw_get_size(partition, total, free) result(istat) bind(C,name='hbw_get_size')
      use iso_c_binding
      implicit none
      integer(C_INT)        :: istat
      integer(C_INT), value :: partition
      integer(C_SIZE_T)     :: total, free
    end function hbw_get_size
  end interface

istat = hbw_get_size(partition, total, free)
print '("status, total, free, used =",I5,3I12)', &
            istat, total, free, total-free
.....
```

Available KNL SKU's

Xeon Phi Knights Landing	KNL 7290	KNL 7250	KNL 7230	KNL 7210
Process	14nm	14nm	14nm	14nm
Architecture	Silvermont	Silvermont	Silvermont	Silvermont
Cores/Threads	72 / 288	68 / 272	64 / 256	64 / 256
Clock (GHz)	1.5	1.4	1.3	1.3
HBM / Speed (GT/s)	16 GB / 7.2	16 GB / 7.2	16 GB / 7.2	16 GB / 6.4
DDR4 / Speed (MHz)	384 GB / 2400	384 GB / 2400	384 GB / 2400	384 GB / 2133
TDP	245W	215W	215W	215W
Omni-Path Fabric	Yes	Yes	Yes	Yes

VTune Metrics





Memkind Usage

- C example (BlackScholes.cpp)
  - Inspect hbw_posix_memalign calls
  - Compile using icc
    - `icc -g –o BlackScholes.hbm BlackScholesSP.cpp -lmemkind`
  - Do functional testing
    - export LD_LIBRARY_PATH, if needed
    - `./BlackScholes.hbm`
- Fortran* example (gppkernel.f90)
  - Inspect MCDRAM directive
  - Compile using ifort
    - `mpiifort -g -o gppkernel.hbm gppkernel.f90 -openmp -lmemkind`
  - Do functional testing
    - export LD_LIBRARY_PATH, if needed
    - `mpirun -n 2 ./gppkernel.hbm 512 2 5000 2 2`

Black-Scholes: Workload Detail

$$d_1 = \frac{\ln(S/X) + (r + v^2/2)T}{v\sqrt{T}}$$

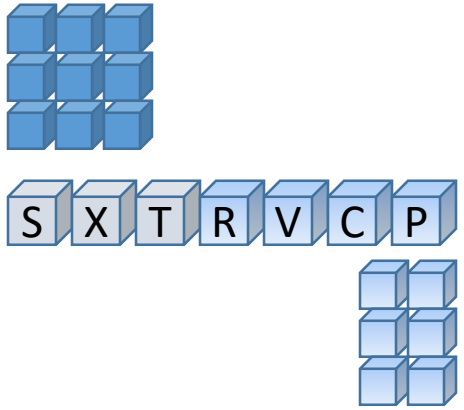
$$d_2 = \frac{\ln(S/X) + (r - v^2/2)T}{v\sqrt{T}} = d_1 - v\sqrt{T}$$

$$c = SCND(d_1) - X\ e^{-rT} CND(d_2)$$

$$p = X\ e^{-rT} CND(-d_2) - SCND(-d_1)$$

$$p + S\ e^{-rT} = c + X\ e^{-rT}$$

$$CND(x) = \frac{1}{2} + \frac{1}{2} ERF\left(\frac{1}{\sqrt{2}} x\right)$$



S: Current Stock price
X: Option strike price
T: Time to Expiry
R: Risk free interest rate
V: Volatility

c: European call
p: European put