MCDRAM on 2\textsuperscript{nd} Generation Intel\textsuperscript{®} Xeon Phi\textsuperscript{™} Processor (code-named Knights Landing): Analysis Methods and Tools

Chris Cantalupo, Karthik Raman, Ruchira Sasanka
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/绩效

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**: Intel® Xeon® processor E5-2697v3 Family (code-named Haswell)
- **KNL**: 2nd generation Intel® Xeon Phi™ processor (code-named Knights Landing)
- **MCDRAM**: Multi-Channel DRAM (High-bandwidth memory)
- **NUMA**: Non-Uniform Memory Access
- **RHEL**: Red Hat Enterprise Linux*

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

At the end of this tutorial, you will be able to:

- Explain different modes of MCDRAM in KNL and how to use them
  - Cache mode, flat mode, and hybrid mode

- Identify whether an app can benefit from MCDRAM
  - Collect memory BW profile and view it on a timeline using Intel® VTune™ Amplifier
  - Identify which functions/structures contribute most to memory BW

- Place data structures in MCDRAM
  - Using compiler directives (Fortran
  - Using hbw and memkind API
  - Using an interposer library without source modification

- Do functional testing & emulation of MCDRAM-enabled code on a NUMA machine
Pre-requisites for the Tutorial

To try the tutorial (hands-on) for your own application pre-requisites are:

- Intel® C++/Fortran Compiler 16.0 or above
- Intel® Vtune™ Amplifier 2016 (Update 2 or above)
- Haswell systems - for memory object analysis via Intel® Vtune™ Amplifier
- Memkind library (see slide 26 for details on how/where to obtain)
- Dual socket NUMA systems for functional testing of MCDRAM enabled code
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?
  - Functional testing and emulation on a NUMA system
**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+

- 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. Binary Compatible with Intel Xeon processors using Haswell Instruction Set (except TSX).

- Bandwidth 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.

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
  - Less addressable memory

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?
  - Functional testing and emulation on a NUMA 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)
  - 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
Instructions for Data Collection

- **Linux** command line (on Haswell):
  ```bash
  amplxe-cl -c memory-access -data-limit=0 -knob analyze-mem-objects=true -knob mem-object-size-min-thres=1024 -- mpirun -n 2 -env I_MPI_DEBUG 5 -env OMP_NUM_THREADS 28 -env KMP_AFFINITY compact,verbose ./gppkernel.hsw 512 2 5000 2 2
  ```

- **Using Intel® Vtune™ Amplifier GUI**:
  - In **Analysis Type** tab; under “Microarchitecture Analysis” menu:
    - **Memory Access**
      - **Analyze memory objects**
      - Minimal memory object size to track, in bytes: 1024

*Other names and brands may be claimed as the property of others.*
Summary View: Bandwidth Histogram/Top Memory Objects

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

**Memory Objects:**
Lists the most actively used memory objects (stack/heap) in the application. Shown for each MPI process.
Bottom-Up View

- Time-line view of BW utilization
- Memory allocation call stack
- 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
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

- **Performance metrics**
  - CPU time/memory bound metrics used to identify functions with memory issues
  - Loads, Stores, LLC Miss 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 systems

- Select Function/Memory Object Allocation Source/Allocation Stack Grouping
- 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 Loads/Stores/LLC 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 ALLOCS/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

- 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?
  - Functional testing and emulation on a NUMA 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(1), mbind(1)
  - Not the preferred method
    - Page-only granularity, OS serialization, no pool management

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

- 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”

- 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
    - `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

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

Allocate 1000 floats from MCDRAM

```c
#include <hbwmalloc.h>
float   *fv;
fv = (float *)hbw_malloc(sizeof(float) * 1000);
```

Allocate arrays from MCDRAM and DDR in Intel® Fortran Compiler

```fortran
REAL, ALLOCATABLE :: A(:), B(:), C(:)

!DEC$ ATTRIBUTES FASTMEM :: A

NSIZE=1024
ALLOCATE (A(1:NSIZE))

Allocate arrays that will come from DDR
ALLOCATE  (B(NSIZE), C(NSIZE))
```
hbw and memkind APIs

- See “man hbwmalloc”

```c
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 hbwposix_memalign(void **memptr, size_t alignment, size_t size);
int hbwposix_memalignpsize(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

```c
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 memkindposix_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
Obtaining Memkind Library

- Homepage: [http://memkind.github.io/memkind](http://memkind.github.io/memkind)

- Download package
  - On Fedora* 21 and above: `yum install memkind`
  - On RHEL* 7: `yum install epel-release; yum install memkind`
  - For other distros: install from [http://download.opensuse.org/repositories/home:/cmcantalupo/](http://download.opensuse.org/repositories/home:/cmcantalupo/)

- Alternatively, you can build from source
  - `git clone https://github.com/memkind.git`
  - See CONTRIBUTING file for build instructions
  - Must use this option to get AutoHBW library

- To create RPM which will install the memkind service and include AutoHBW:
  ```
git clone https://github.com/cmcantalupo/memkind.git
cd memkind
git checkout sc15-mcdram-tutorial
./autogen.sh
./configure
make rpm
  ```

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

- **Fortran* example (gppkernel.f90)**
  - Inspect MCDRAM directive
  - Compile using ifort
    - `mpiifort -g -o gppkernel.hbm gppkernel.f90 -openmp -lmemkind`
  - Do functional testing on DDR machine
    - `export LD_LIBRARY_PATH, if needed`
    - `mpirun -n 2 ./gppkernel.hbm 512 2 5000 2 2`

- **C example (hello_hbw_example.c)**
  - Inspect hbw_malloc calls
  - Compile using icc
    - `icc -g -o hello_hbw hello_hbw_example.c -lmemkind`
  - Do functional testing
    - `export LD_LIBRARY_PATH, if needed`
    - `./hello_hbw`

*Other names and brands may be claimed as the property of others.*
Memkind Policies and Memory Types

- How do we make sure we get memory only from MCDRAM?
  - This depends on POLICY
    - See man page and `hbw_set_policy()` / `hbw_get_policy()`
    - **BIND**: Will cause app to die when it runs out of MCDRAM
    - **PREFERRED**: Will allocate from DDR if MCDRAM not sufficient (default)

- Allocating 2 MB and 1 GB pages
  - Use `hbw_posix_memalign_psize()`
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

- **Demo**
  - see /examples/autohbw_test.sh
  - Run gpp with AutoHBW

- **Environment variables** (see autohbw_README)
  - AUTO_HBW_SIZE=x[:y]
  - AUTO_HBW_LOG=level
  - AUTO_HBW_MEM_TYPE=memory_type       # useful for interleaving

- **Finding source locations of arrays**
  - export AUTO_HBW_LOG=2
  - ./app_name > log.txt
  - autohbw_get_src_lines.pl log.txt app_name
Advanced Topic: MCDRAM in SNC4 Mode

- SNC4: Sub-NUMA Clustering
  - KNL die is divided into 4 clusters (similar to a 4-Socket Haswell)
    - 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

- 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 --membind=4,5,6,7`
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?
  - Functional testing and emulation on a NUMA 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 -p numastat exec_name`
  - `cat /sys/devices/system/node/node*/meminfo`
    - Info about each NUMA node
  - `cat /proc/meminfo`
    - Aggregate info for system

- Utilities that provide MCDRAM node info
  - `<memkind_install_dir>/bin/memkind-hbw-nodes`
  - `numactl --hardware`
  - `lscpu`
MCDRAM Emulation Demo

- Use cores on socket A and DDR on socket B (remote memory)
  - `export MEMKIND_HBW_NODES=0` # HBW allocation go to DDR on Socket A
  - `numactl --membind=1 --cpunodebind=0 <your_command>`

- Any HBW allocations will allocate DDR on socket A
  - Accesses to DDR on socket A (local memory) has higher BW
    - Also has lower latency (which is an inaccuracy)
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
Keep in touch

- Join IXPUG (The Intel Xeon Phi User's Group)
  - **New Memory Types WG**
    - *Describe usage models, review requirements, establish best known methods for new memory types*
  - Sign up form available at:
    - [https://docs.google.com/forms/d/1FoRHI6NDn7u0ALnGRtMF5q2X3R1h1MeCDLxJJfOy55rs/viewform](https://docs.google.com/forms/d/1FoRHI6NDn7u0ALnGRtMF5q2X3R1h1MeCDLxJJfOy55rs/viewform)