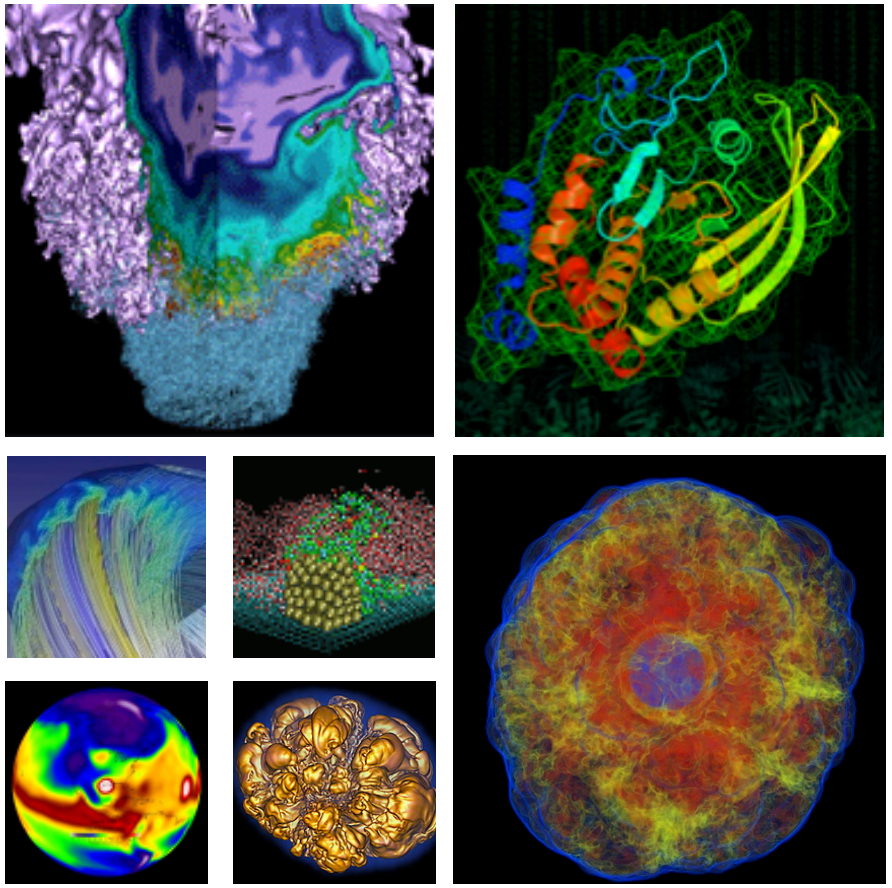
Estimating Performance Impact of MCDRAM on KNL Using Dual-Socket Ivy Bridge Nodes on Cray XC30



Acknowledgement:
Jeongnim Kim, Martyn Corden, Christopher Cantalupo, Sumedh Naik, other engineers at Intel, Inc and Jack Deslippe at NERSC who helped during the VASP "Dungeon session" in Oct, 2015 at Intel Offices in Hillsboro, OA.

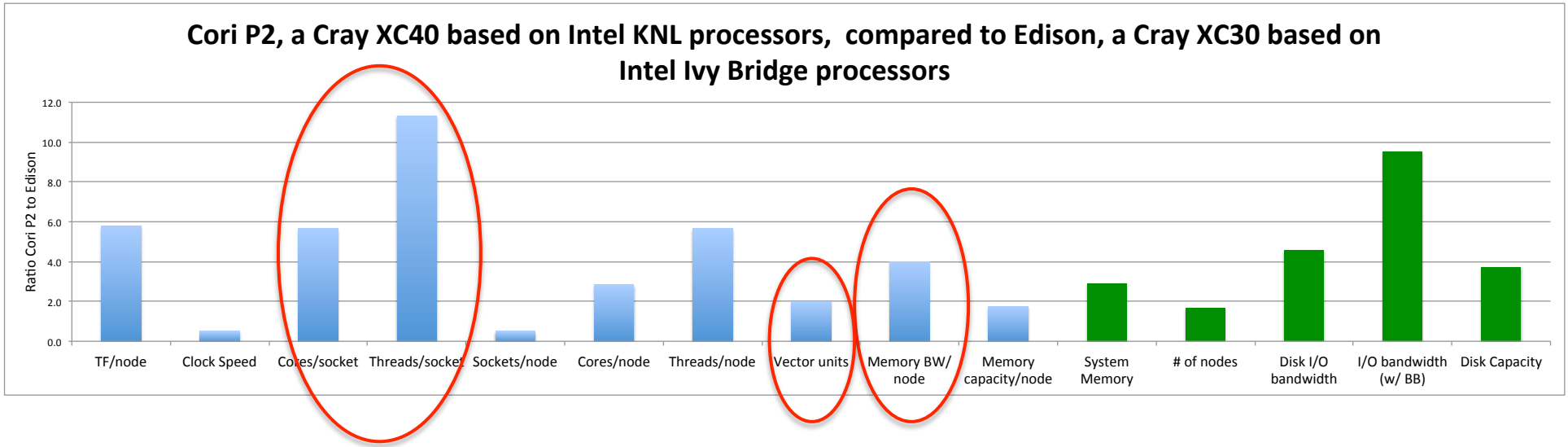
Zhengji Zhao[1] and Martijn Marsman[2]
[1] NERSC User Engagement Group
[2] University of Vienna

Cray User Group Meeting,
March 8-12, 2016, London UK

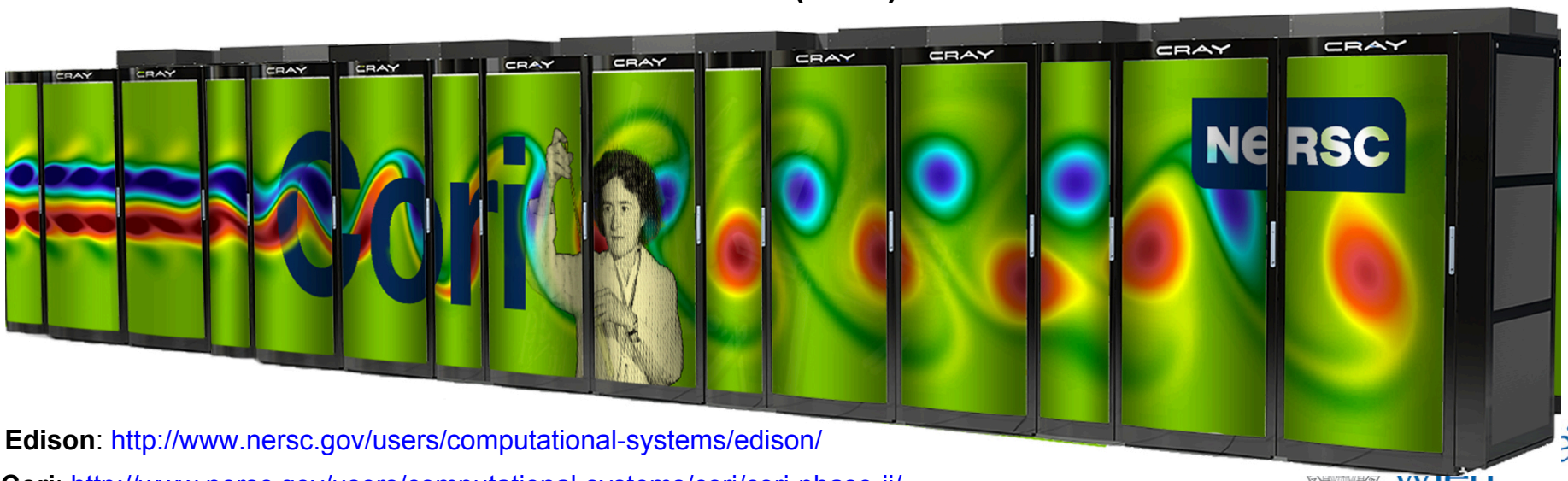
Outline

- Motivation
- How to estimate MCDRAM (or HBM) performance on existing Xeon nodes
- Application to VASP
- Conclusions

Efficient use of HBM is important to get optimal application performance on Cori P2



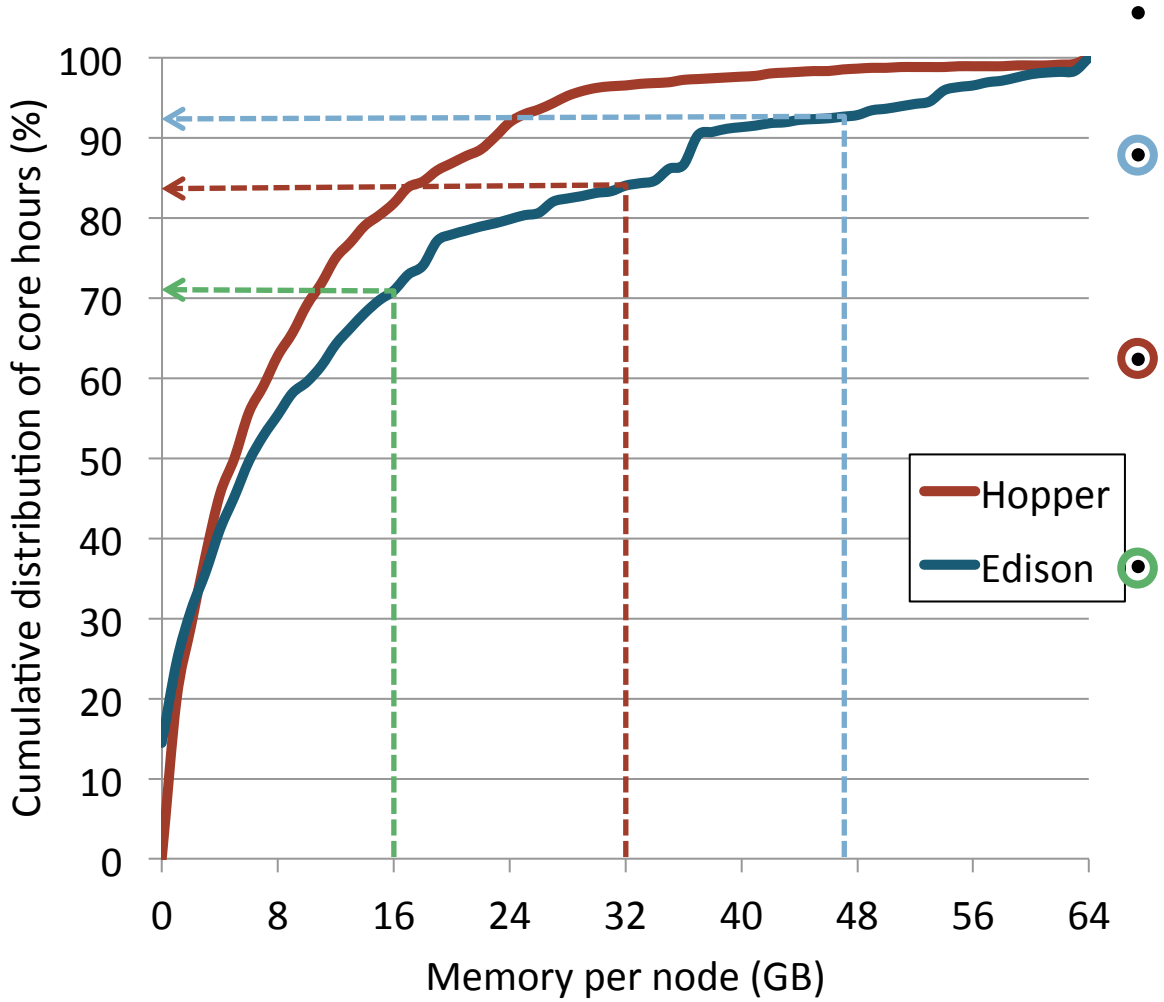
Cori's KNL nodes will have 16 GB MCDRAM (HBM) with ~ 5x DDR4 bandwidth



Edison: http://www.nersc.gov/users/computational-systems/edison/

Cori: http://www.nersc.gov/users/computational-systems/cori/cori-phase-ii/

Memory utilization of NERSC workload in allocation year 2014

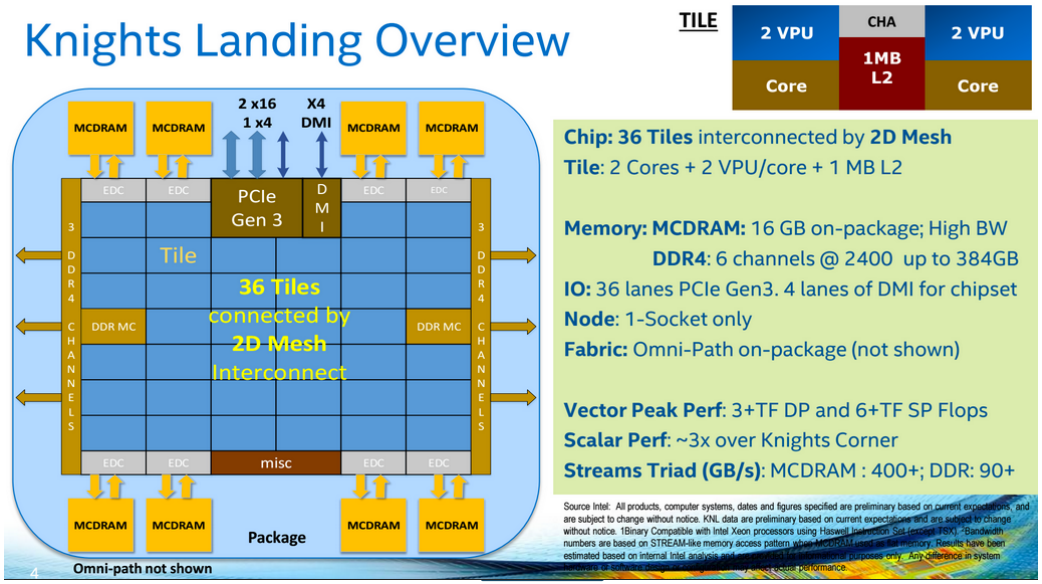


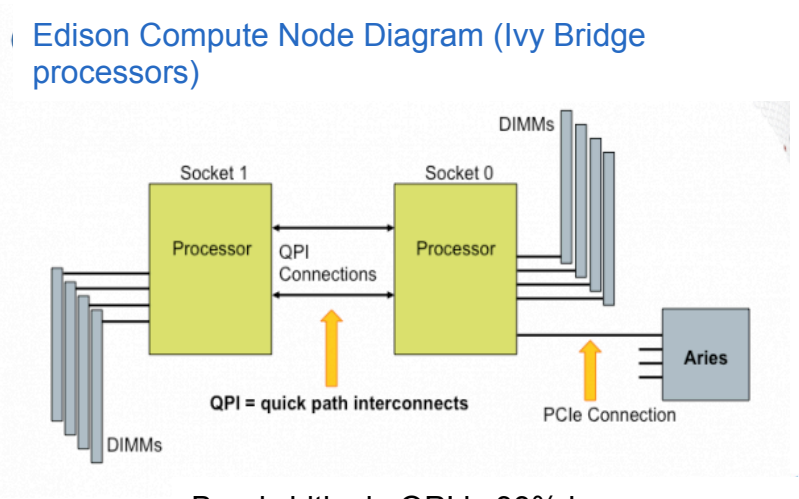
71% of Edison workload used less than 16GB memory per node. However, considering more cores on the KNL node, we estimate about half of the Edison workload will not fit into the 16GB HBM (if no changes).

Slide from http://portal.nersc.gov/project/mpccc/baustin/NERSC_2014_Workload_Analysis_30Oct2015.pdf

Dual Socket Xeon nodes can serve as proxies for HBM on KNL nodes

- **Accessing memory on the remote socket via the QPI bus is slower compared to accessing the near socket memory.**
- **The near socket memory can mimic the HBM on KNL; while the remote socket memory can mimic the DDR memory.**
- **This is not an accurate model of the bandwidth and latency characteristics of the KNL on package memory (MCDRAM, or HBM) but is a reasonable way to determine which data structures rely critically on bandwidth.**





Bandwidth via QPI is 33% lower

Memkind library is an Intel development tool to allocate arrays in HBM

- The Memkind library is a user extensible heap manager built on top of Jemalloc which enables control of memory characteristics and a partitioning of the heap between kinds of memory (including user defined kinds of memory).
- Using the Intel compiler directive, !DIR$ ATTRIBUTES FASTMEM, for Fortran codes, and/or using the malloc wrapper APIs, hbw_malloc, for C/C++ codes, one can allocate selected arrays to HBM on KNL by linking the application codes to the Memkind library.
- On today's Xeon nodes, Memkind can be used to simulate/ estimate the HBM performance impact on the application codes

http://memkind.github.io/memkind

Code changes are required to use Memkind to selectively allocate arrays in HBM

- **Add compiler directive !DIR$ ATTRIBUTES FASTMEM in the Fortran codes**
  - real, allocatable :: a(:,:), b(:,:), c(:)
  - !DIR$ ATTRIBUTES FASTMEM :: a, b, c
- **Use hbw_malloc, hbw_calloc to replace the malloc, calloc calls in the C/C++ codes**
  - #include <hbwmalloc.h>
  - malloc(size) -> hbw_malloc(size)
- **Link the codes to the memkind and jemalloc libraries using Intel compilers**
  - mpiifort –dynamic -O3 -openmp mycode.f90 -L<path-to-mekind-library> –lmemkind –ljemalloc

On today's Xeon nodes (proxies for KNL) an environment variable, MEMKIND_HBW_NODES, is needed to designate a socket for HBM

- **Run the codes with the numactl and env MEMKIND_HBW_NODES**
  - export MEMKIND_HBW_NODES=0
  - numactl --membind=1 --cpunodebind=0 ./a.out
- **Notes:**
  - Memkind can only allocate heap variables to the HBM. There is no way to allocate stack variables to HBM currently.
  - The FASTMEM directive may be supported by other compilers in the future, e.g., Cray, but may be slightly different from the Intel FASTMEM directive.
  - Srun can be used for parallel jobs

AutoHBW tool is another Intel development tool to allocate arrays in HBM without code changes

- **AutoHBW Tool – automatically allocate the arrays in certain size range to the HBM at run time.**
- **AutoHBW intercepts the standard heap allocations (e.g., malloc, calloc) and allocate them out of HBM using the memkind API.**
- **No code change is required**
- **Included in the memkind distribution:** http://memkind.github.io/memkind

  Reference: examples/autohbw_README

AutoHBW tool uses the environment variables to control its behavior

- **AUTO_HBW_SIZE=x[:y]**
  - Indicates that any allocation larger than x and smaller than y should be allocated in HBM. x,y (in K, M, or G)
  - AUTO_HBW_SIZE=1M:5M # allocations between 1M and 5M allocated in HBM
- **AUTO_HBW_LOG=level**
  - Sets the verbosity of the logging level:
    - 0 = no messages are printed for allocations
    - 1 = a log message is printed for each allocation (Default)
    - 2 = a log message is printed for each allocation with a backtrace.
  - Using the autohbw_get_src_lines.pl script to find source lines for each allocation.
  - Logging adds extra overhead. Use 0 for performance critical runs

Application codes need to link to the autohbw and memkind libraries

- **MEMKIND_HBW_NODES=<list of numa nodes>**
  - Sets a comma separated list of NUMA nodes as HBW nodes, e.g., MEMKIND_HBW_NODES=0
  - For non-KNL node this env must be set
- **AUTO_HBW_MEM_TYPE=<memory_type>**
  - Sets the type of memory that should be automatically allocated. Default: MEMKIND_HBW.
  - Examples:

    AUTO_HBW_MEM_TYPE=MEMKIND_HBW **(Default)**

    AUTO_HBW_MEM_TYPE=MEMKIND_HBW_HUGETLB

    AUTO_HBW_MEM_TYPE=MEMKIND_HUGETLB
- **Link the codes to the autohbw, memkind and jemalloc libraries**
  - mpiifort -O3 -openmp mycode.f90 mycode.f90 -L<path-to-memkind-library> -lautohbw –lmemkind –ljemalloc

The autohbw libraries may need to be preloaded to make sure the malloc commands in applications get intercepted by AutoHBW.

- **Run the codes with the numactl and proper environment variables**
  - export MEMKIND_HBW_NODES=0
  - export AUTO_HBW_LOG=0
  - export AUTO_HBW_MEM_TYPE=MEMKIND_HBW
  - export AUTO_HBW_SIZE=1K:5K # all allocations between sizes 1K and 5K allocated in HBM
  - Export LD_LIBRARY_PATH=<path-to-memkind-library>:${LD_LIBRARY_PATH}
  - numactl --membind=1 --cpunodebind=0 ./a.out
- **Make sure the malloc calls intercepted by the autohbw library APIs:**
  - For dynamic builds, using **LD_PRELOAD** or **LD_LIBRARY_PATH** to allow libautohbw.so:libmemkind.so in front of the system default path.
  - For static builds, make the autohbw and memkind libraries in front of the link line. May use -Wl,-ymalloc to confirm
- **Use the numastat –p <pid> command to displays the memory allocations among sockets for a running process**

Memkind and AutoHBW applications in VASP

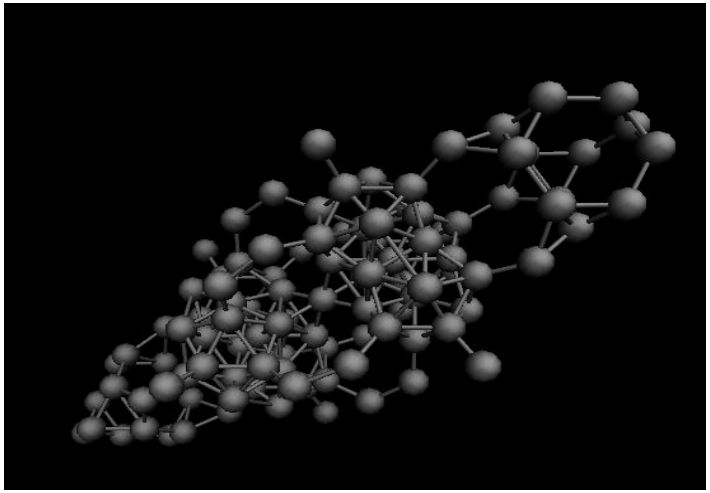
- **The Vienna Ab-initio Simulation Package (VASP) is a widely used materials science application.**
  - Rank #1 code at NERSC, uses about 10-12% of total computing cycles at NERSC each year
- **The fundamental mathematical problem that VASP solves is a non-linear eigenvalue problem that has to be solved iteratively via self-consistent iteration cycles until a desired accuracy is achieved. FFTs and Linear Algebra libraries (BLAS/LAPACK/ScaLAPACK) are heavily depended on.**
- **Fortran code with MPI or MPI + OpenMP**

VASP: http://www.vasp.at/
VASP: G. Kresse and J. Furthm_ller. Efficiency of ab-initio total energy calculations for metals and semiconductors using a plane-wave basis set. Comput. Mat. Sci., 6:15, 1996

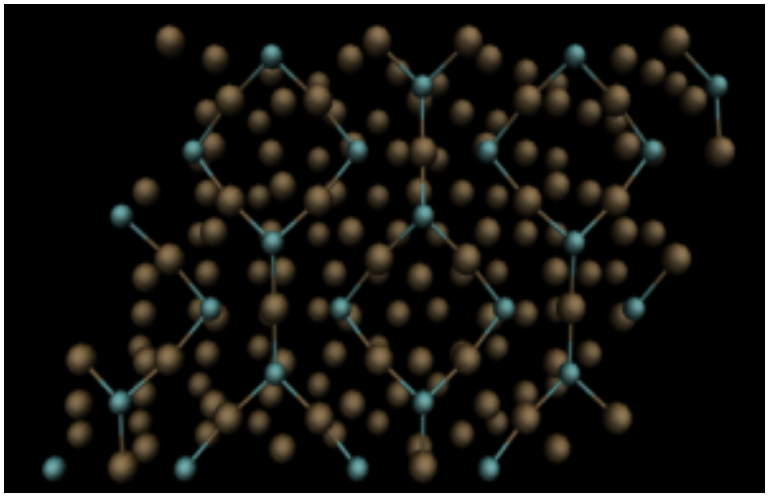
VASP versions used and test cases

- **Two different versions of VASP were used in this study**
  - Currently released, version 5.3.5, a pure MPI code
  - A development version (as of 9/29/2015), a MPI + OpenMP hybrid
  - Compiled with Intel compilers and used MKL / ELPA libraries

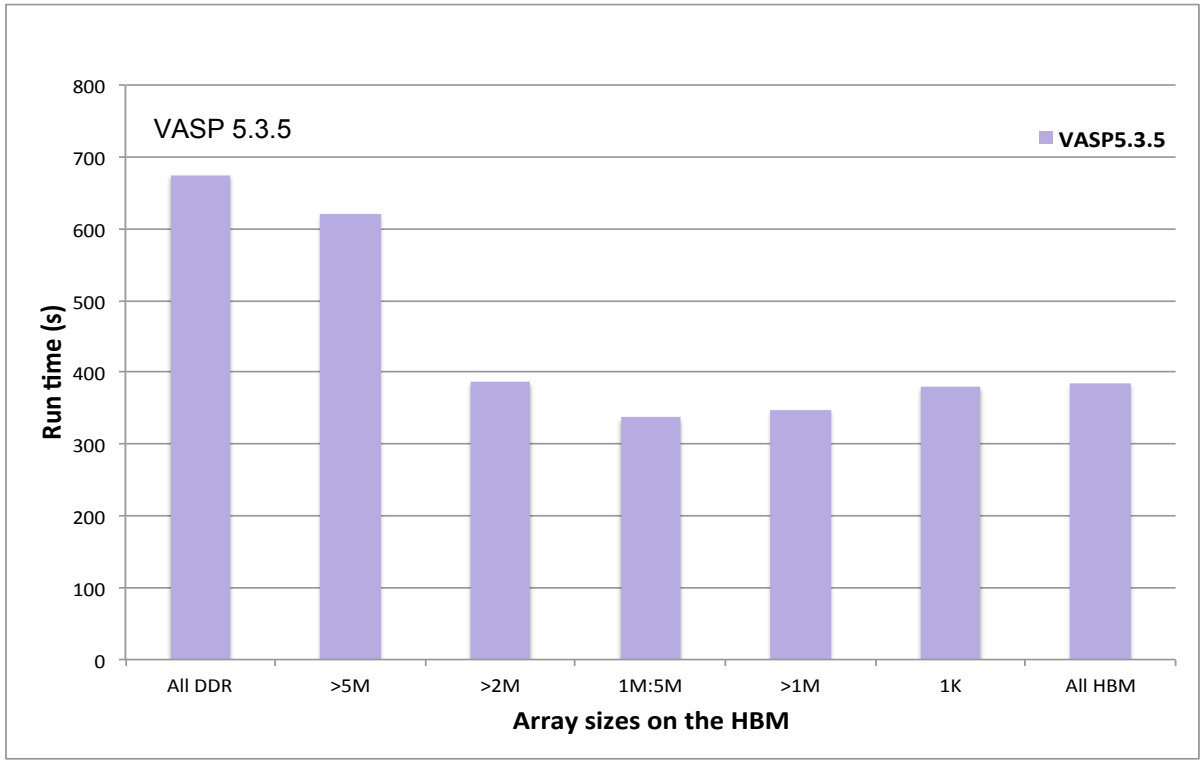
Test case 1: B.hR105-s



Test case 2: PdO@Pd-slab



Estimating the performance impact of HBM on VASP code using AutoHBW tool on Edison



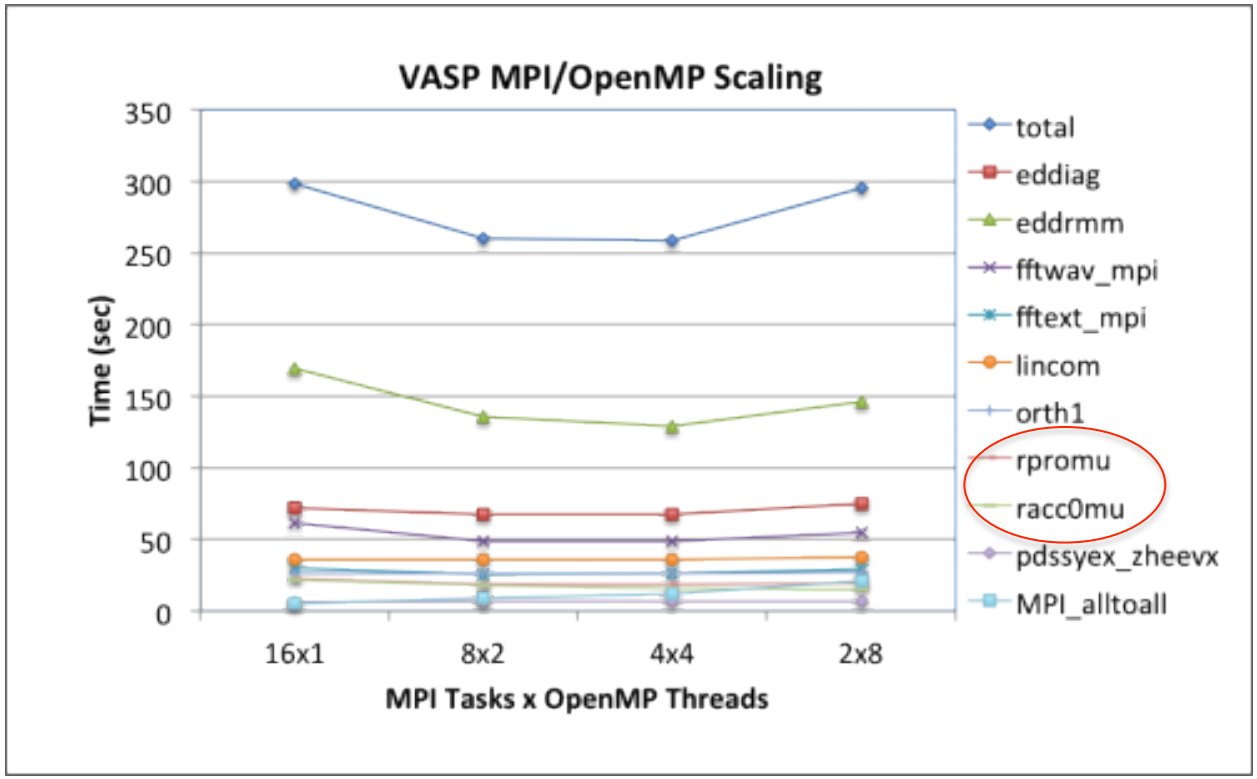
VASP run time for the hybrid functional calculation (HSE06) when arrays in different size range were allocated to the HBM.
The bandwidths of the near socket memory (simulating MCDRAM) and the far socket memory via QPI (simulating DDR) differ by 33%

- **VASP performs significantly better (40%) when arrays within 1M to 5M were allocated to HBM.**
- **Expect to have a larger performance benefit from HBM on KNL**
- **Application End users may use HBM more efficiently using AutoHBW.**

Using Memkind to selectively allocate arrays to HBM

- Profiling
- Identify the candidate routines and memory objects that may get benefit from using the HBM
- Add the Intel compiler directive, FASTMEM, to the code
- Linking the code with the memkind library
- Run with numactl on the KNL proxy, the Ivy Bridge Edison compute node

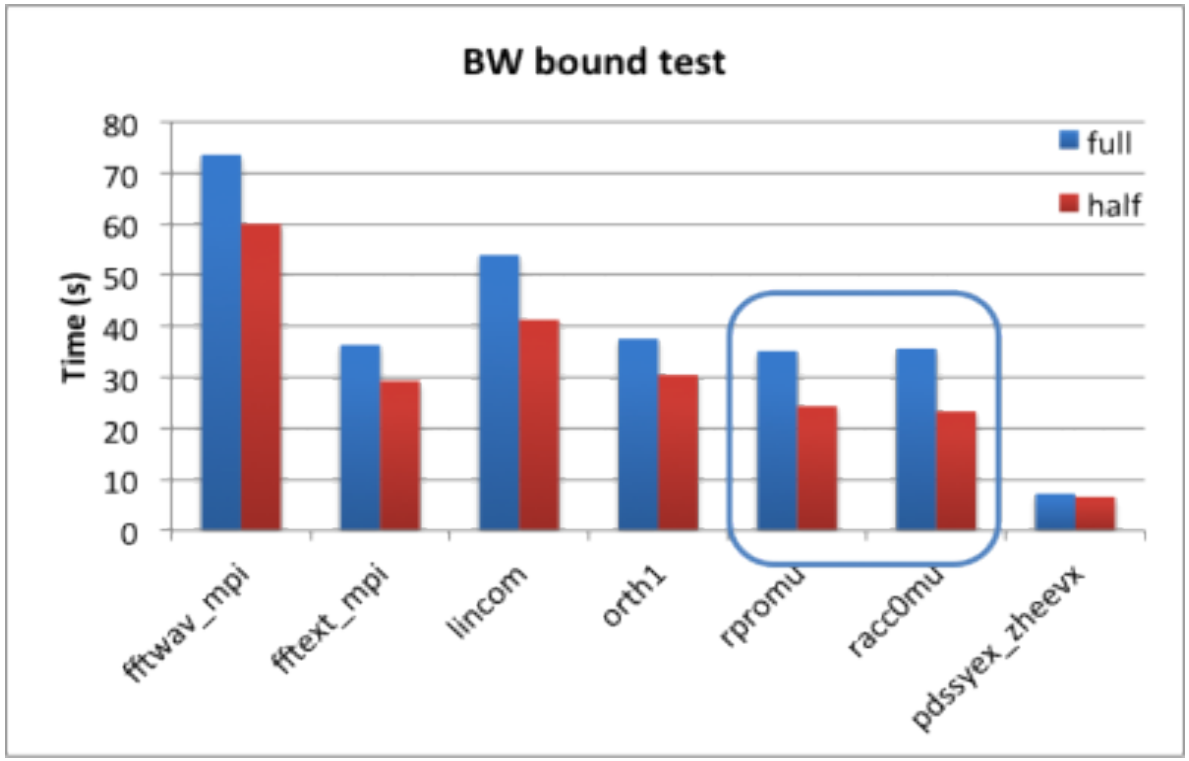
VASP heavily depends on the math libraries



The thread scaling of the MPI/OpenMP VASP code with the test case PdO@Pd-slab. These are the fixed core (16 cores) tests running on one of the sockets on an Intel dual-socket Haswell node (Xeon E5-2697 v3 2.6 GHz) at the University of Vienna.

- **Top routines, eddiag, eddrmm, fft*, lincom and orht1, all depend heavily on the math libraries used.**
- **The real space projection routines, rpromu and racc0mu, could be good candidate hotspots for HBM optimization**

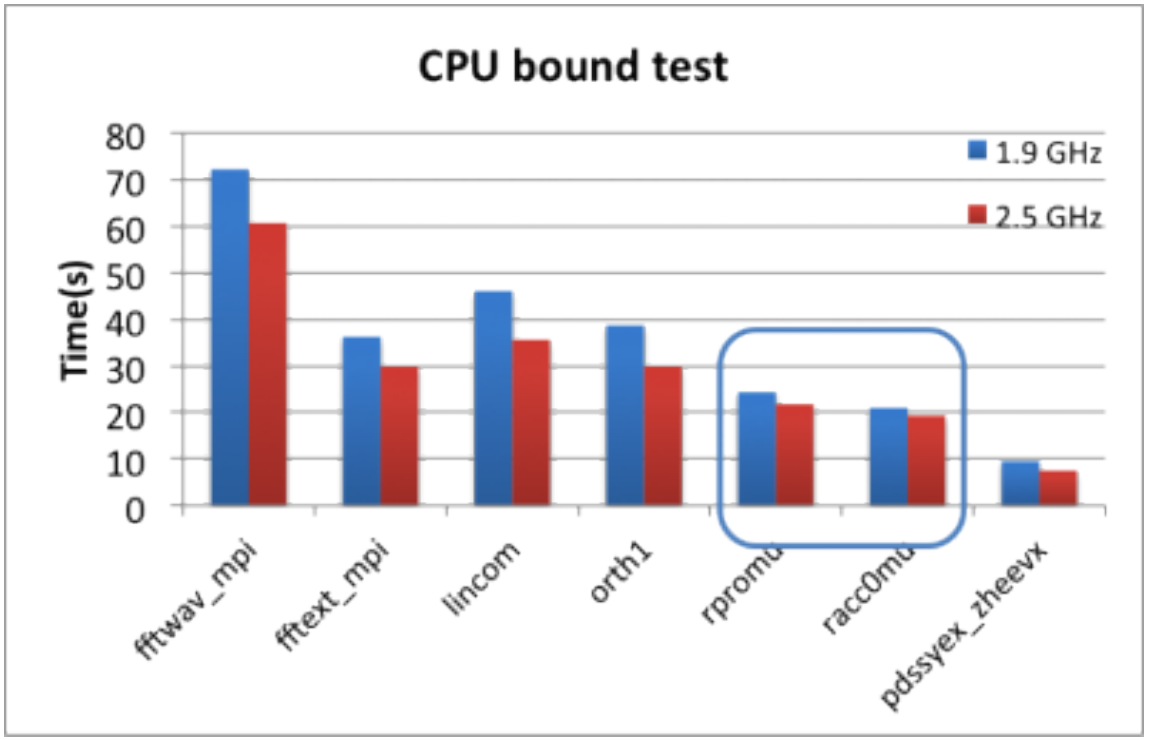
Is VASP memory bandwidth bound?



The VASP run time breakdown over the top subroutines when running on the fully packed (blue) and half-packed (red) nodes. The tests were done on an Intel Xeon E5-2697 v3 node and used the test case PdO@Pd-slab. When running on half packed nodes, the memory bandwidth available for each task is twice as much as it is when running on fully packed nodes.

- **Several top subroutines ran faster when running on the unpacked nodes, especially rpromu, and racc0mu, indicating they might be bandwidth bound.**

Is VASP compute bound?



The VASP run time breakdown over the top subroutines when running at the clock speeds of 2.5 GHz, and 1.9 GHz on an Intel Xeon E5-2697 v3 node. The test case used was PdO@Pd-slab, and this experiment was designed to test if the code is CPU bound.

- **Several top subroutines ran slower when the clock speed was decreased, indicating they were likely CPU bound. Relatively rpromu and racc0mu were affected less by the clock speed change.**
- **Two real space projection routines, rpromu and racc0mu were likely memory bandwidth bound.**

Allocating arrays in subroutine racc0mu in HBM using the Memkind library and the FASTMEM Intel compiler directive

Simple cases

```
 SUBROUTINE RACC0MU(NONLR_S, WDES1, CPROJ_LOC, CRACC, LD, NSIM, LDO)
...
    REAL(qn),ALLOCATABLE:: WORK(:),TMP(:,:)
    GDEF,ALLOCATABLE    :: CPROJ(:,:)

   !DIR$ ATTRIBUTES FASTMEM :: WORK,TMP,CPROJ
...
    ALLOCATE(WORK(ndata*NSIM*NONLR_S
%IRMAX),TMP(NLM,ndata*2*NSIM),CPROJ(WDES1%NPRO_TOT,NSIM))
...
  END SUBROUTINE RACC0MU
```

The only change needed to allocate the arrays WORK, TMP, and CPROJ to the HBM was adding !DIR$ ATTRIBUTES FASTMEM :: WORK,TMP,CPROJ

The FASTMEM directive does not work directly with the Fortran array pointers.

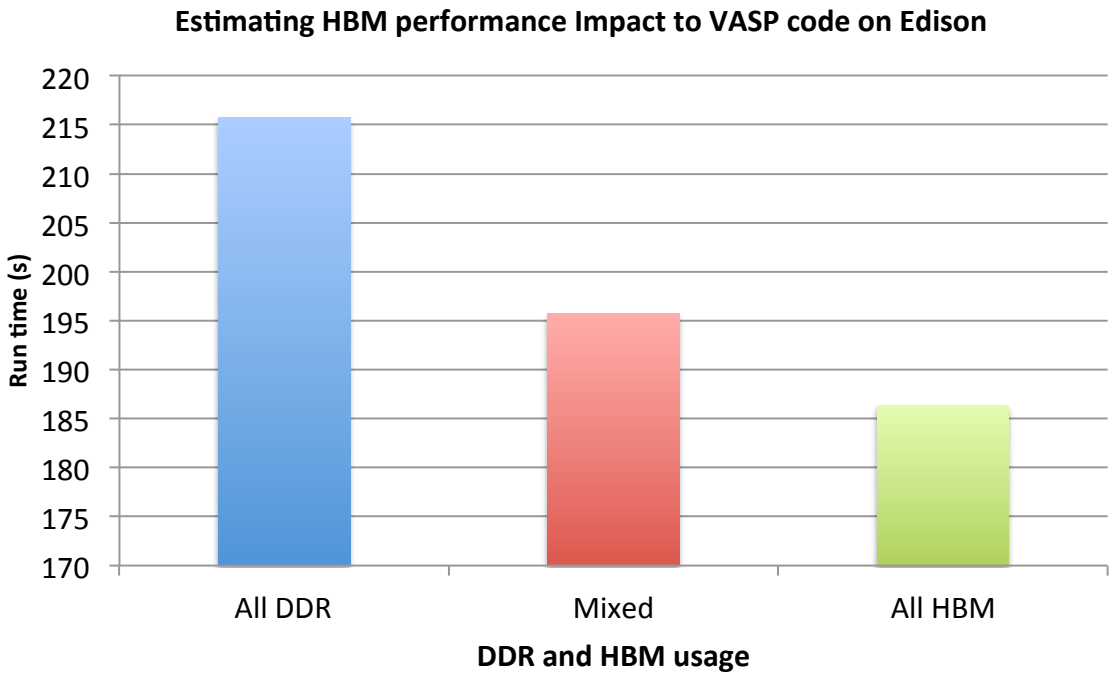
```
INTEGER, POINTER :: NLI (:,:) ! index for gridpoints
REAL(qn),POINTER, CONTIGUOUS :: RPROJ (:) ! projectors on real space grid
COMPLEX(q),POINTER, CONTIGUOUS::CRREXP(:,:,:) ! phase factor exp (i k (R(ion)-
r(grid)))

...

ALLOCATE(NONLR_S%NLI(IRMAX,NIONS),NONLR_S%RPROJ(NONLR_S%IRALLOC))
ALLOCATE(NONLR_S%CRREXP(IRMAX,NIONS,1))
```

- **To allocate array pointers NLI, RPROJ and CRREXP to HBM,**
  - Need to introduce intermediate allocatable arrays.
  - Allocate the intermediate arrays in HBM
  - Associate the array pointers to these intermediate arrays in HBM

FASTMEM: Working with Fortran array pointers

```
INTEGER, POINTER :: NLI (:,:) ! index for gridpoints
REAL(qn),POINTER, CONTIGUOUS :: RPROJ (:) ! projectors on real space grid
COMPLEX(q),POINTER, CONTIGUOUS::CRREXP(:,:,:) ! phase factor exp (i k (R(ion)-
r(grid)))

! To exploit fastmem
INTEGER, ALLOCATABLE :: fm_NLI (:,:) ! index for gridpoints
REAL(qn), ALLOCATABLE :: fm_RPROJ (:) ! projectors on real space grid
COMPLEX(q), ALLOCATABLE :: fm_CRREXP(:,:,:) ! phase factor exp (i k (R(ion)-
r(grid)))

!DIR$ ATTRIBUTES FASTMEM :: fm_RPROJ,fm_CRREXP,fm_NL

...

! put NLI and RPROJ into fastmem
ALLOCATE(NONLR_S%fm_NLI(IRMAX,NIONS),NONLR_S%fm_RPROJ(NONLR_S
%IRALLOC))
NONLR_S%NLI =>NONLR_S%fm_NLI
NONLR_S%RPROJ=>NONLR_S%fm_RPROJ

ALLOCATE(NONLR_S%fm_CRREXP(IRMAX,NIONS,1))
NONLR_S%CRREXP=>NONLR_S%fm_CRREXP
```

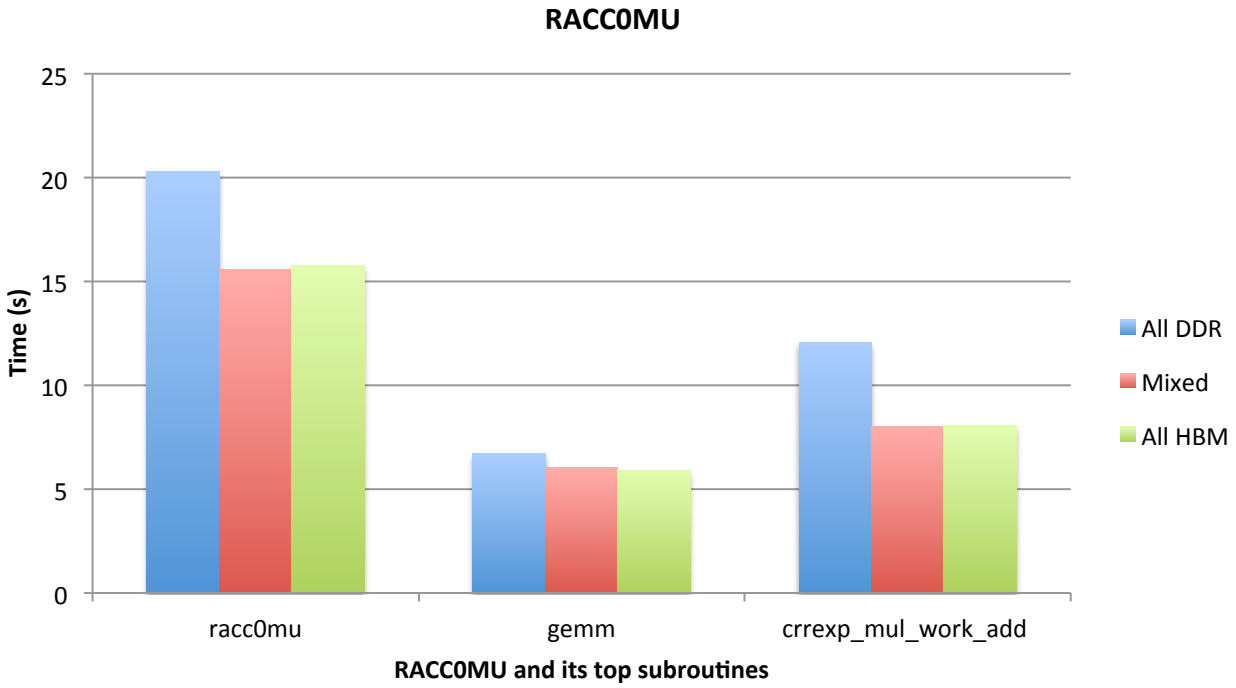
VASP performance when a few selected arrays were allocated in HBM using FASTMEM+Memkind



VASP performance comparison between runs when everything was allocated in the DDR memory (blue/ slow), when only a few selected arrays were allocated to HBM (red/ mixed), and when everything was allocated to HBM (green/fast). The test case PdO@Pd-slab was used, and the tests were run on a single Edison node.

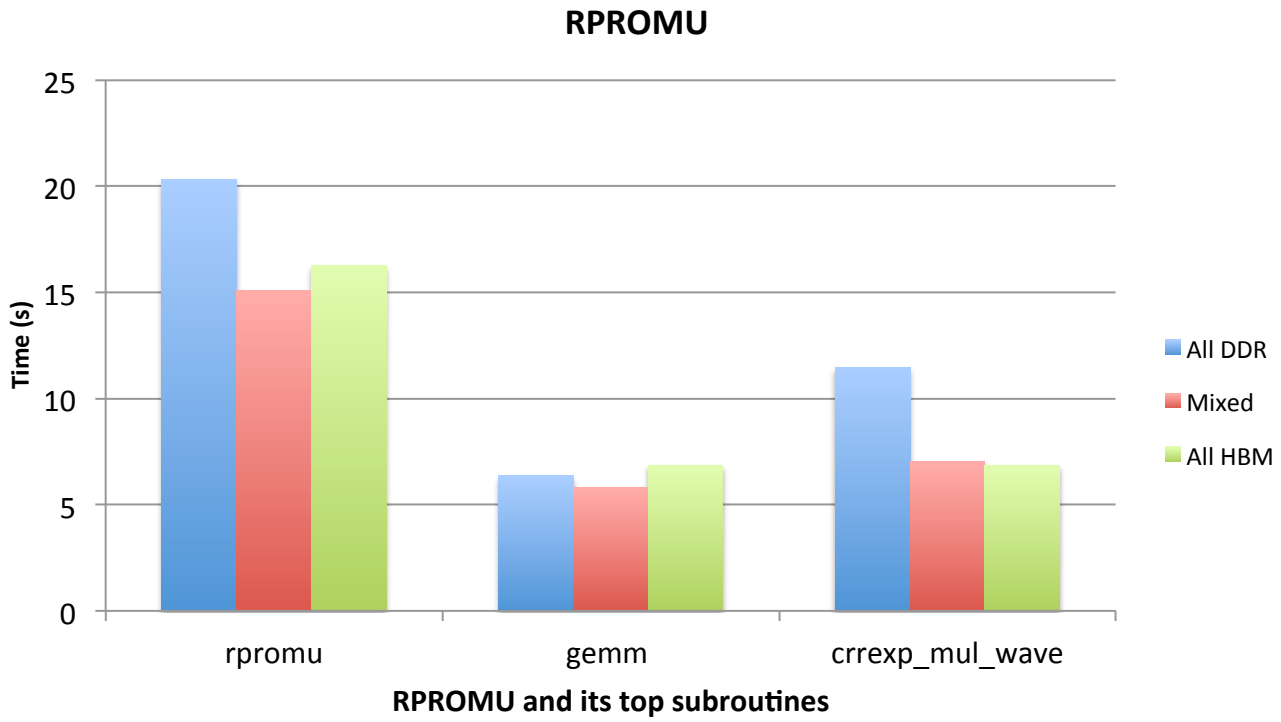
- **About 9% of speedup of total run time was achieved when a few selected arrays were allocated to HBM in comparison to when everything was allocated in the DDR memory.**
- **14% of speedup when everything was allocate on the HBM**
- **This test case had a modest memory bandwidth usage.**

Allocating a few arrays in RACC0MU to the HBM achieved the same level of speedup as allocating everything in the HBM



- **Allocating a few arrays in the real-space projection routine, RACC0MU, in the HBM (red:Mixed) showed up to 23% of speedup compared to allocating everything in the DDR memory.**

Allocating a few arrays in RPROMU to the HBM achieved the same level of speedup as allocating everything in HBM



- **Allocating a few arrays in the real-space projection routine, RPROMU, in the HBM (red:Mixed) showed up to 26% of speedup compared to allocating everything in the DDR memory.**

Conclusions

- **HBM may have a significant performance benefit to applications.**
  - Selectively allocating arrays to HBM is a key optimization tactic to use the small amount of available HBM efficiently on KNL nodes.
- **Intel development tools like Memkind and AutoHBW are very helpful for users and developers to do HBM optimizations for KNL on today's architectures. Early adoption of these tools is key to produce the KNL ready codes.**
- **The available tools such as Memkind and AutoHBW are only capable of allocating heap arrays in HBM, and there is no good way of allocating stack arrays to HBM so far.**
  - This prevents OpenMP private arrays from using HBM.
  - It is also necessary to change stack arrays (where applicable) to allocatable arrays to use HBM

Conclusions -- continued

- **The Intel compiler directive !DIR$ ATTRIBUTES FASTME does not work with array pointers directly. Allocatable work arrays must be introduced and allocated to the HBM first before associating the array pointers to them.**
- **The FASTMEM directive is an Intel compiler specific directive, and is not supported by other compilers.**
  - This may cause the HBM optimization portability issue.
  - Cray compilers will support this directive (with slight modification) in the future.
- **The estimation may not be exact analogy to the HBM on KNL, however, the approach used here will be applicable for KNL without modification.**