

Intel Xeon Phi 7200 series processor (Knights Landing) architecture and software

PRACE PATC Course: Intel MIC Programming Workshop
29 June 2016

Andrey Semin
Principal Engineer
HPC Technology Manager,
Europe, Middle-East and Africa

Legal Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. 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. Intel products are not intended for use in medical, life-saving, life-sustaining, critical control or safety systems, or in nuclear facility applications. Intel products 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.

Intel may make changes to dates, specifications, product descriptions, and plans referenced in this document at any time, without notice.

This document may contain information on products in the design phase of development. The information herein is subject to change without notice. Do not finalize a design with this information.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

Intel Corporation or its subsidiaries in the United States and other countries may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.

Wireless connectivity and some features may require you to purchase additional software, services or external hardware.

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.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, visit Intel Performance Benchmark Limitations

Intel, the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

Copyright © 2016 Intel Corporation. All rights reserved.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

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



Agenda

Morning session:

- Platform system and chip architectures
- Core details and AVX512 outline
- Memory modes
- Cluster modes
- Early performance

Afternoon session:

- Where KNL is the best choice
- Optimization directions
- AVX512 highlights
- Memory profiling





Intel® Xeon Phi™ Product Family

Highly-Parallel Roadmap



Available from 2013

Knights Corner

Intel® Xeon Phi™
x100 Product Family

- 22 nm process
- Coprocessor only
- >1 TF DP Peak
- Up to 61 Cores
- Up to 16GB GDDR5



Launched in 2016

Knights Landing

Intel® Xeon Phi™
x200 Product Family

- 14 nm process
- Host Processor & Coprocessor
- >3 TF DP Peak¹
- Up to 72 Cores
- Up to 16GB HBM
- Up to 384GB DDR4²
- ~465 GB/s STREAM
- Integrated Fabric²

Future

Knights Hill

3rd generation

- 10 nm process
- Integrated Fabric (2nd Generation)
- In Planning...

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 complete information visit <http://www.intel.com/performance>.

*Results will vary. This simplified test is the result of the distillation of the more in-depth programming guide found here: <https://software.intel.com/sites/default/files/article/383067/is-xeon-phi-right-for-me.pdf> All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice. 1 Over 3 Teraflops of peak theoretical double-precision performance is preliminary and based on current expectations of cores, clock frequency and floating point operations per cycle. FLOPS = cores x clock frequency x floating-point operations per second per cycle. 2 Host processor only

Copyright © 2016 Intel Corporation. All rights reserved.

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



Knights Landing: Architecture

Over 3 TF DP peak
Full Xeon ISA compatibility through AVX-512
~3x single-thread vs. compared to Knights Corner

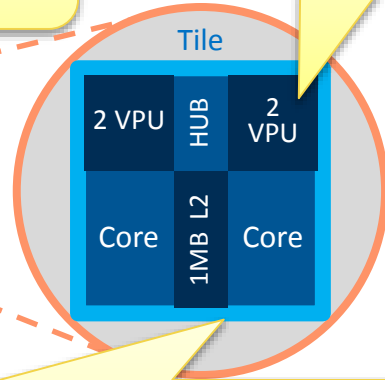
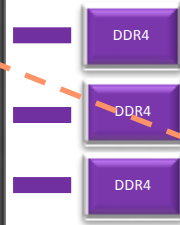
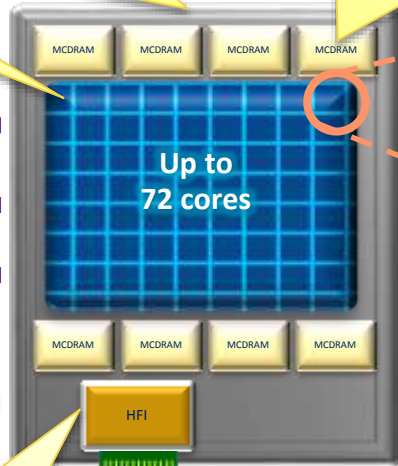
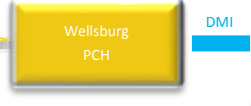
Up to 16GB high-bandwidth on-package memory (MCDRAM)
Exposed as NUMA node
>400 GB/s sustained BW

2x 512b VPU per core
(Vector Processing Units)

Up to 72 cores (36 tiles)
2D mesh architecture

6 channels DDR4
Up to 384GB
~90 GB/s

Common with Grantley PCH



Based on Intel® Atom Silvermont processor with many HPC enhancements
Deep out-of-order buffers
Gather/scatter in hardware
Improved branch prediction
4 threads/core
High cache bandwidth
& more

On-package 2 ports OPA Integrated Fabric



Diagram is for conceptual purposes only and only illustrates a CPU and memory – it is not to scale and does not include all functional areas of the CPU, nor does it represent actual component layout.

All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Copyright © 2016 Intel Corporation. All rights reserved.

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



Intel® Xeon Phi™ Processor



SKU Lineup

	CORES	GHZ	INTEGRATED MEMORY	FABRIC*	DDR4	POWER**	PRICE†
7290* Best Performance/Node	72 ↑	1.5 ↑	16GB 7.2 GT/s	Yes	384GB 2400 MHz	245W ↓	\$6,254 ↑
7250 Best Performance/Watt	68 ↑	1.4 ↑	16GB 7.2 GT/s	Yes	384GB 2400 MHz	215W	\$4,876 ↑
7230 Best Memory Bandwidth/Core	64	1.3	16GB 7.2 GT/s ↑	Yes	384GB 2400 MHz ↑	215W	\$3,710 ↑
7210 Best Value	64	1.3	16GB 6.4 GT/s	Yes	384GB 2133 MHz	215W	\$2,438

*Available beginning in September

**Add 15 watts for integrated fabric

†Recommended Customer Pricing (RCP); add \$287 for integrated fabric option

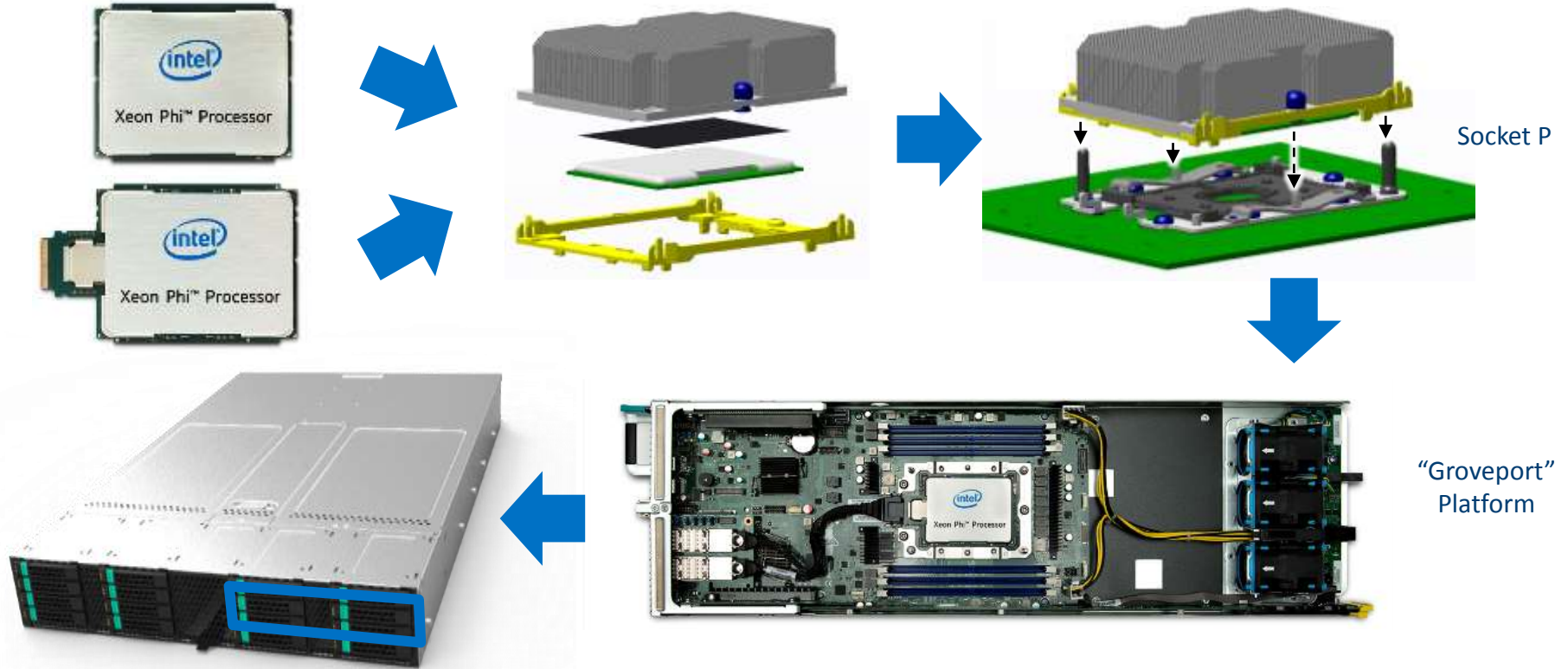
All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Copyright © 2016 Intel Corporation. All rights reserved.

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



Knights Landing Host Processor



All products, systems, dates and figures are preliminary based on current expectations, and are subject to change without notice.

Copyright © 2016 Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Andrey Semin | 29 June 2016 | Slide 7



KNL Architecture Overview

ISA

Intel® Xeon® Processor Binary-Compatible (w/Broadwell)

On-package memory

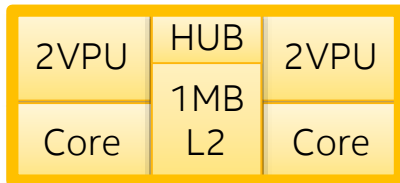
Up to 16GB, ~490 GB/s STREAM at launch

Platform Memory

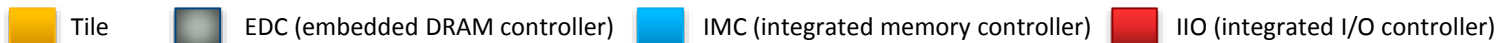
Up to 384GB (6ch DDR4-2400 MHz)

- Fixed Bottlenecks**
- ✓ 2D Mesh Architecture
 - ✓ Out-of-Order Cores
 - ✓ 3X single-thread vs. KNC

TILE:
(up to 36)

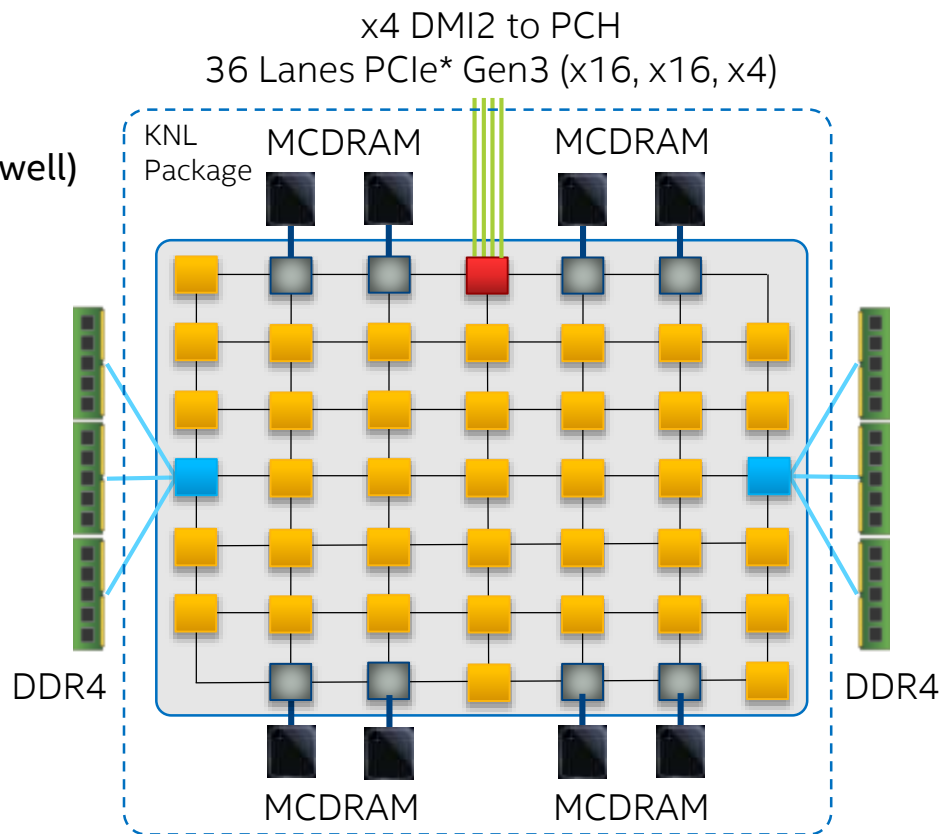


Enhanced Intel® Atom™ cores based on Silvermont Microarchitecture



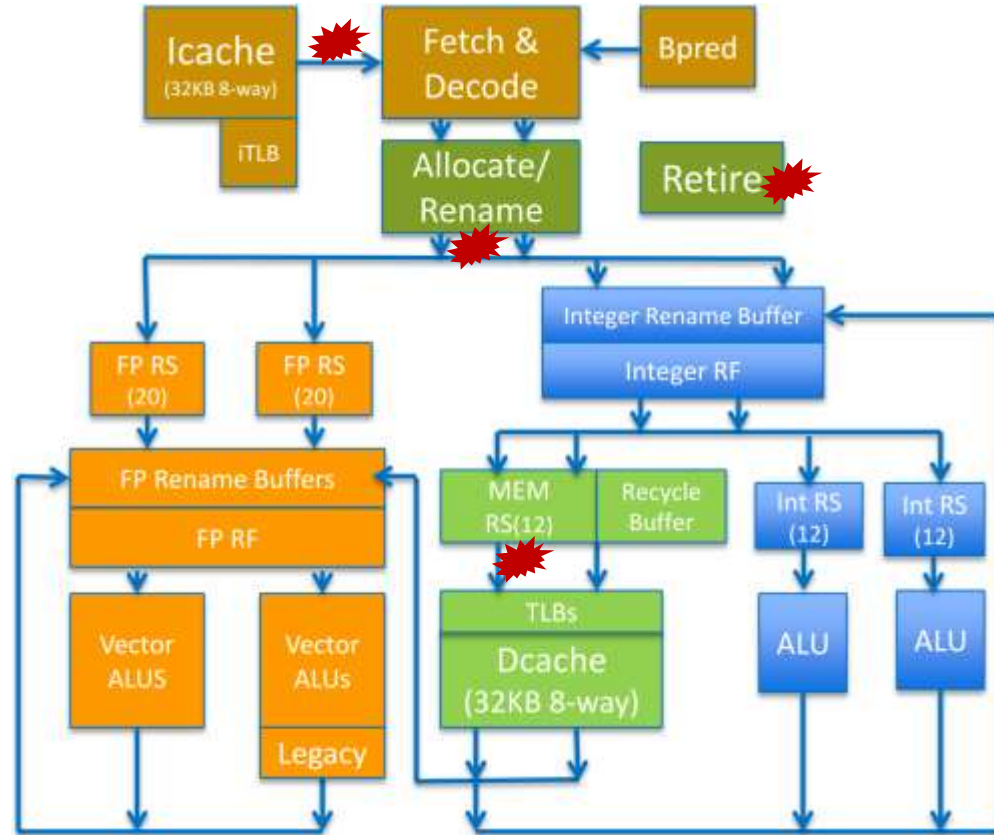
Copyright © 2016 Intel Corporation. All rights reserved.


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



KNL Hardware Threading

- 4 threads per core SMT
- Resources dynamically partitioned
- Re-order Buffer
- Rename buffers
- Reservation station
- Resources shared
- Caches
- TLB



 Thread selection point

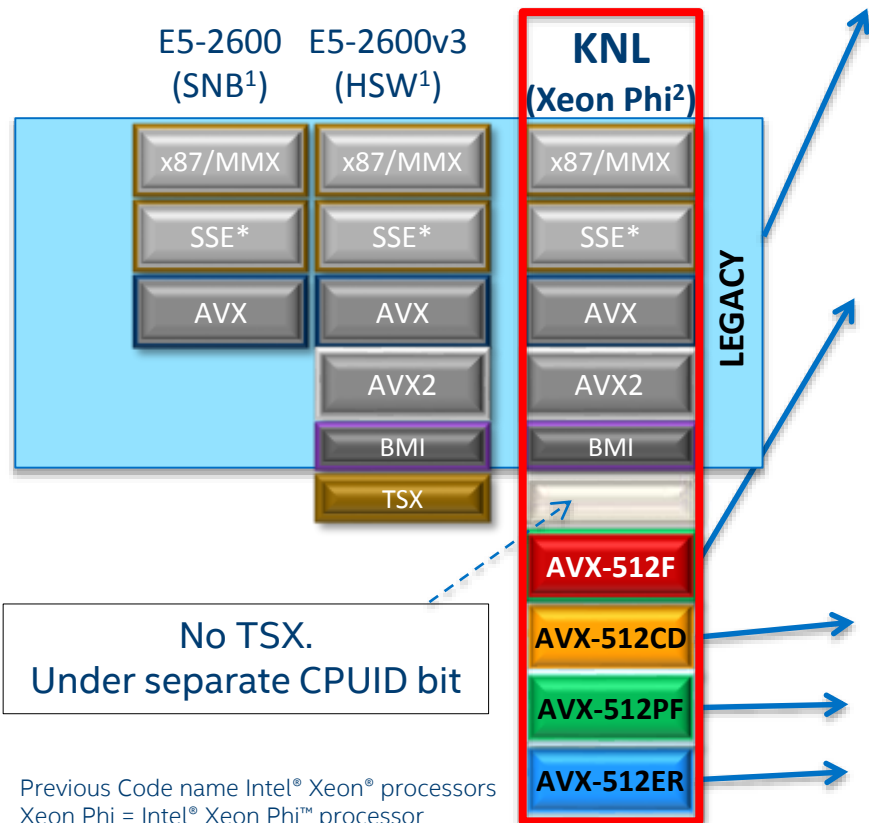
All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Copyright © 2016 Intel Corporation. All rights reserved.

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



KNL Supported ISA



KNL implements all legacy instructions

- Legacy binary runs w/o recompilation
- KNC binary requires recompilation

KNL introduces AVX-512 Extensions

- 512-bit FP/Integer Vectors
- 32 registers, & 8 mask registers
- Gather/Scatter

Conflict Detection: Improves Vectorization

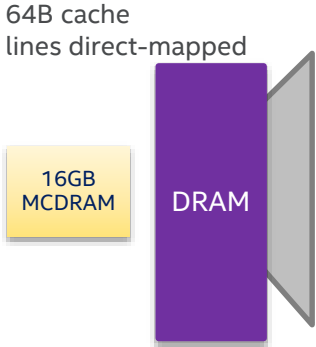
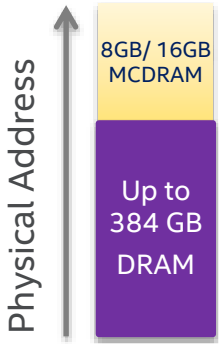
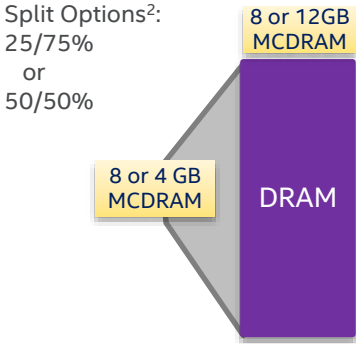
Prefetch: Gather and Scatter Prefetch

Exponential and Reciprocal Instructions

1. Previous Code name Intel® Xeon® processors
2. Xeon Phi = Intel® Xeon Phi™ processor

Integrated On-Package Memory Usage Models

Model configurable at boot time and software exposed through NUMA¹

	Cache Model	Flat Model	Hybrid Model
	 <p>64B cache lines direct-mapped</p> <p>16GB MCDRAM</p> <p>DRAM</p>	 <p>Physical Address</p> <p>8GB/ 16GB MCDRAM</p> <p>Up to 384 GB DRAM</p>	 <p>Split Options²: 25/75% or 50/50%</p> <p>8 or 12GB MCDRAM</p> <p>8 or 4 GB MCDRAM</p> <p>DRAM</p>
Description	Hardware automatically manages the MCDRAM as a “L3 cache” between CPU and external DDR memory	Manually manage how the app uses the integrated on-package memory and external DDR for peak perf	Harness the benefits of both Cache and Flat models by segmenting the integrated on-package memory
Usage Model	<ul style="list-style-type: none"> App and/or data set is very large and will not fit into MCDRAM Unknown or unstructured memory access behavior 	<ul style="list-style-type: none"> App or portion of an app or data set that can be, or is needed to be “locked” into MCDRAM so it doesn’t get flushed out 	<ul style="list-style-type: none"> Need to “lock” in a relatively small portion of an app or data set via the Flat model Remaining MCDRAM can then be configured as Cache

1. NUMA = non-uniform memory access;

2. As projected based on early product definition; Platform Memory (DDR4) only available for bootable KNL host processor

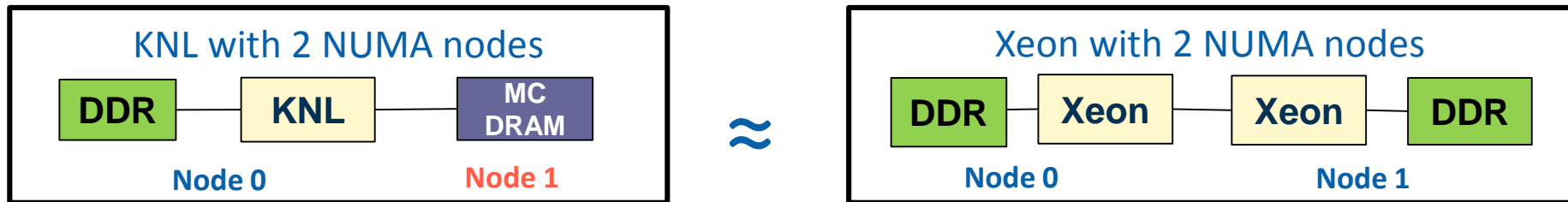
Copyright © 2016 Intel Corporation. All rights reserved.

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



Flat MCDRAM: SW Architecture

MCDRAM exposed as a separate NUMA node



Memory allocated in DDR by default → keeps non-critical data out of MCDRAM §

Flat MCDRAM with existing NUMA support in Legacy OS

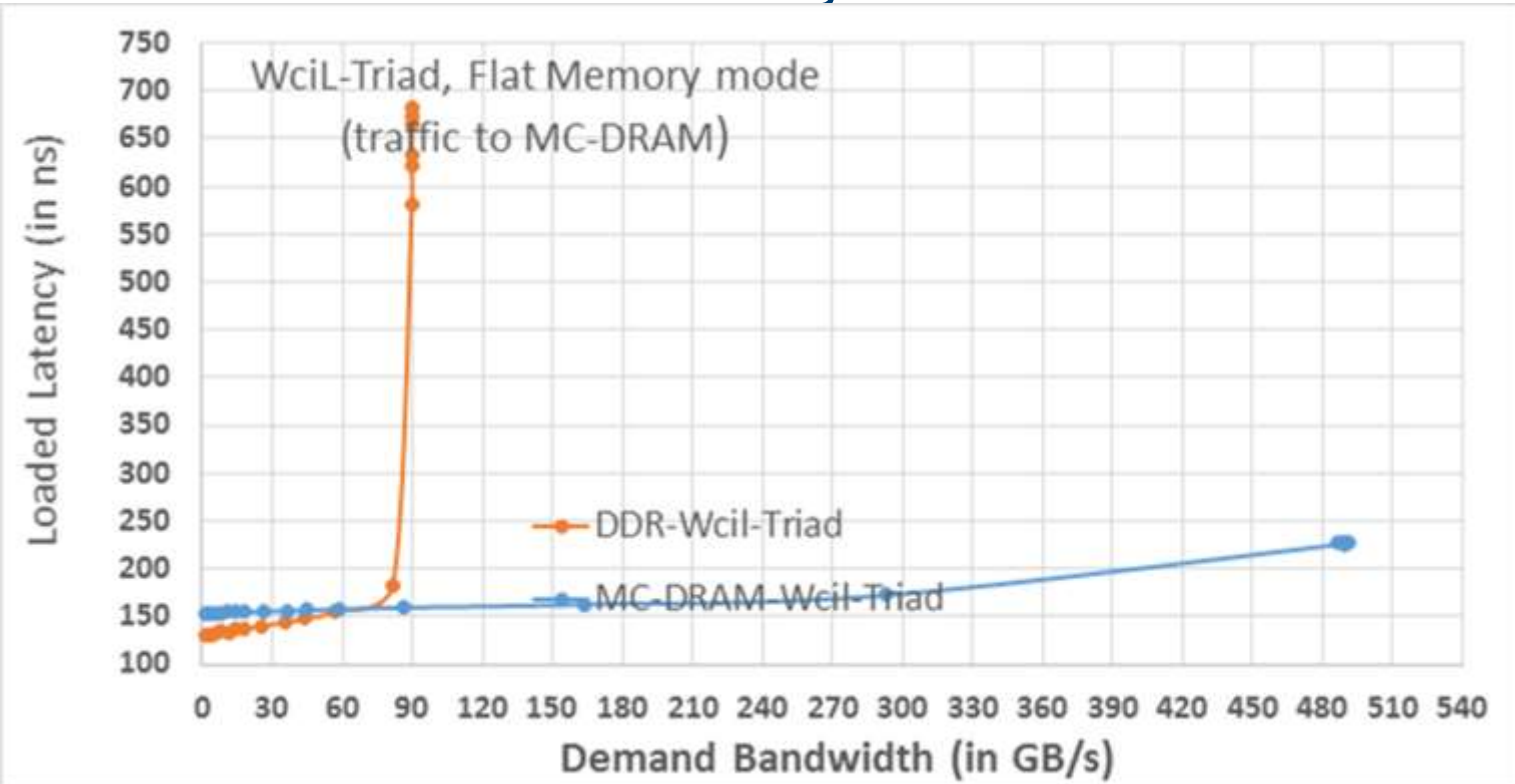
Copyright © 2016 Intel Corporation. All rights reserved.

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

Andrey Semin | 29 June 2016 | Slide 13



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>

All products, systems, dates and figures are preliminary based on current expectations, and are subject to change without notice.

Copyright © 2016 Intel Corporation. All rights reserved.

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

Andrey Semin | 29 June 2016 | Slide 14



Software visible memory configuration (numactl --hardware)

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
```

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

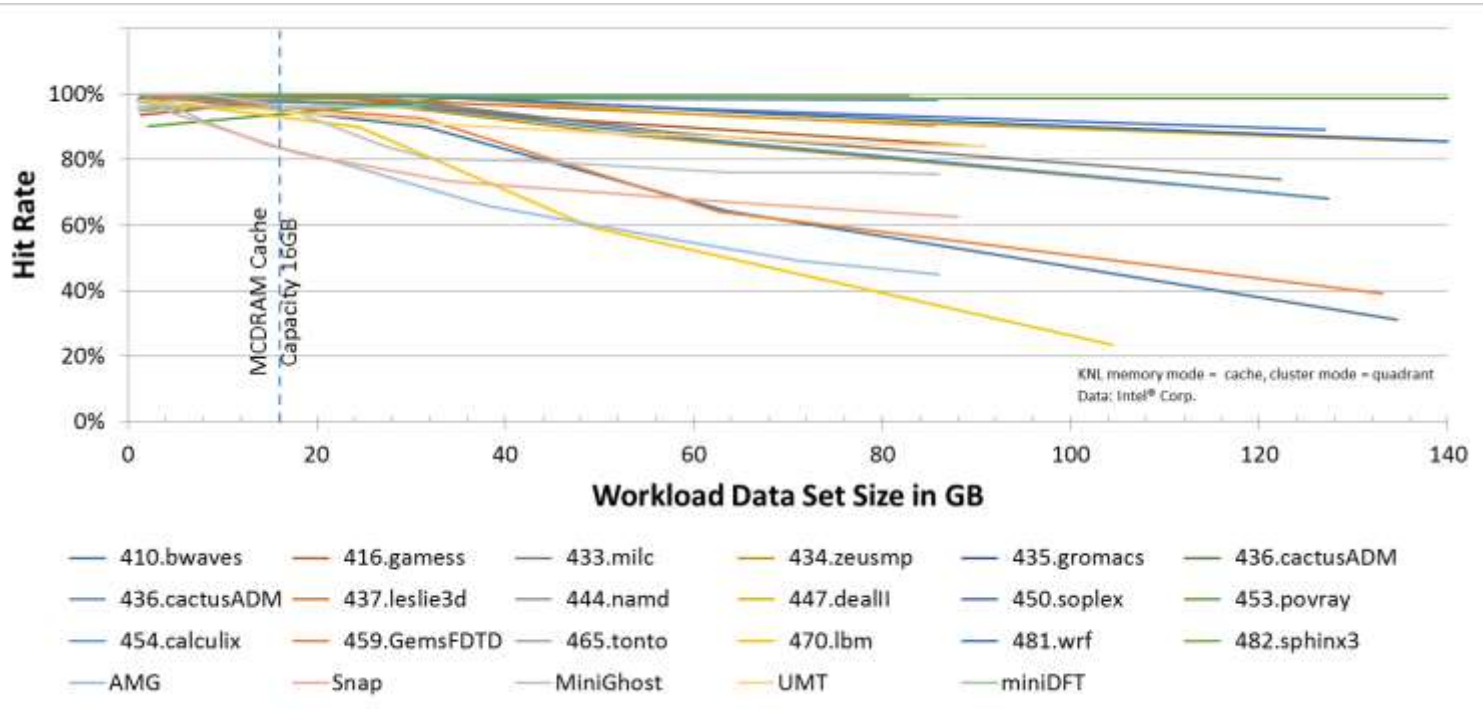
$ mpirun -np 68 numactl ./app                 # DDR4
```

In the latter case the app should explicitly allocate critical data in MCDRAM, using two methods:

- “**Fast Malloc**” functions in High BW library <https://github.com/memkind>)
- “**FASTMEM**” Compiler Annotation for Intel Fortran



MCDRAM Cache Hit Rate



MCDRAM performs well as cache for many workloads
Enables good out-of-box performance without memory tuning

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>

Copyright © 2016 Intel Corporation. All rights reserved.

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

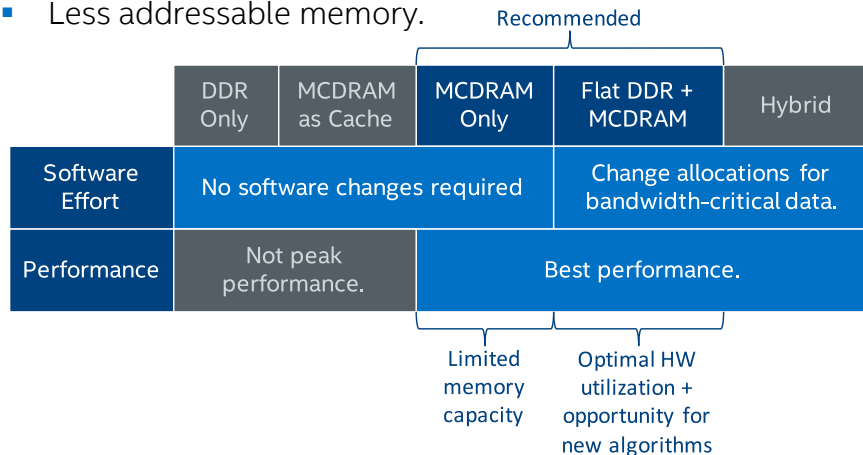
Andrey Semin | 29 June 2016 | Slide 16



Memory Modes

■ MCDRAM as Cache

- Upside:
 - No software modifications required.
 - Bandwidth benefit.
- Downside:
 - Latency hit to DDR.
 - Limited sustained bandwidth.
 - All memory is transferred DDR -> MCDRAM -> L2.
 - Less addressable memory.



■ Flat Mode

- Upside:
 - Maximum bandwidth and latency performance.
 - Maximum addressable memory.
 - Isolate MCDRAM for HPC application use only.
- Downside:
 - Software modifications required to use DDR and MCDRAM in the same application.
 - Which data structures should go where?
 - MCDRAM is a limited resource and tracking it adds complexity



Flat MCDRAM SW Usage: Code Snippets

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

Allocate into DDR

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



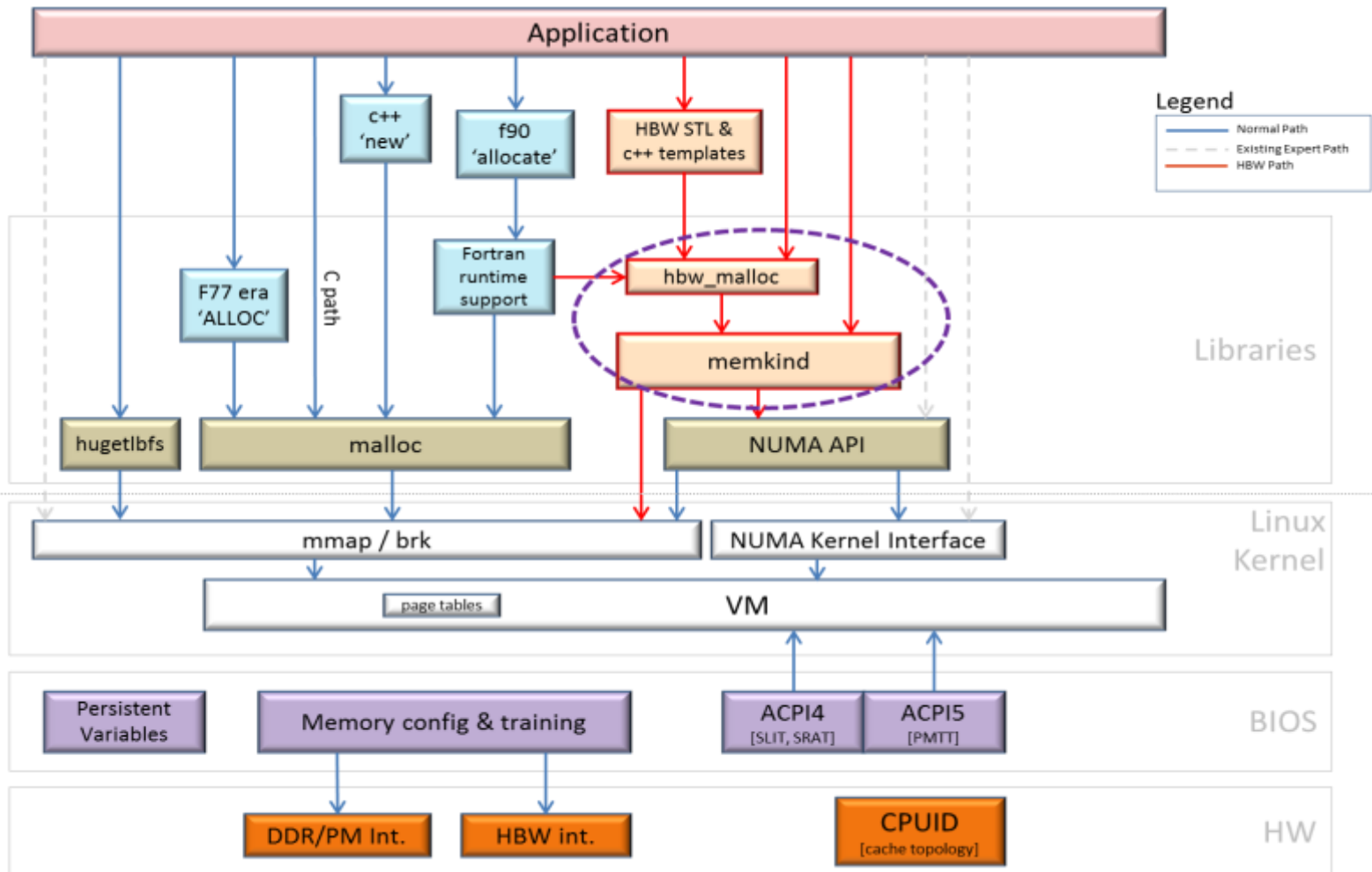
Allocate into MCDRAM

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

Intel Fortran

Allocate into MCDRAM

```
c    Declare arrays to be dynamic  
    REAL, ALLOCATABLE :: A(:)  
  
!DEC$ ATTRIBUTES, FASTMEM :: A  
  
    NSIZE=1024  
c    allocate array 'A' from MCDRAM  
c  
    ALLOCATE (A(1:NSIZE))
```



Copyright © 2016 Intel Corporation. All rights reserved.

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



High Bandwidth (HBW) malloc API

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 MANAGEMENT:

```
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);
```



memkind - “Kinds” of Memory

- Many “kinds” of memory supported by 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.

High Bandwidth (HBW) malloc API

```
HBWMALLOC(3)                HBWMALLOC                HBWMALLOC(3)

NAME
    hbwmalloc - The high bandwidth memory interface

SYNOPSIS
    #include <hbwmalloc.h>

    Link with -ljemalloc -lnuma -lmemkind -lpthread

    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);
```

Released Publicity: <https://github.com/memkind>

Copyright © 2016 Intel Corporation. All rights reserved.

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

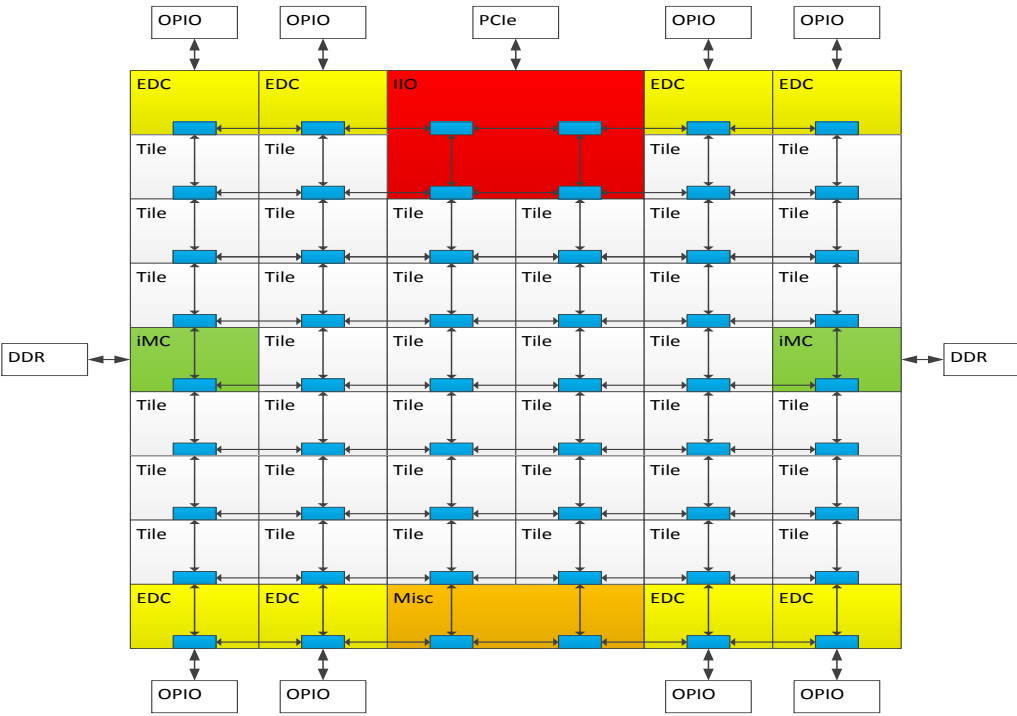
Andrey Semin | 29 June 2016 | Slide 22



AutoHBW Library

- Simplest way to experiment with HBW memory is with AutoHBW library:
 - `LD_PRELOAD=libautohbw.so ./application`
- Run-time configuration options are passed through environment variables:
 - `AUTO_HBW_SIZE=x[:y]`
Any allocation larger than x and smaller than y should be allocated in HBW memory.
 - `AUTO_HBW_MEM_TYPE`
Sets the “kind” of HBW memory that should be allocated (e.g. `MEMKIND_HBW`)
 - `AUTO_HBW_LOG` and `AUTO_HBW_DEBUG` for extra information.
- Easy to integrate similar functionality into other libraries, C++ allocators, etc.

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)

All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

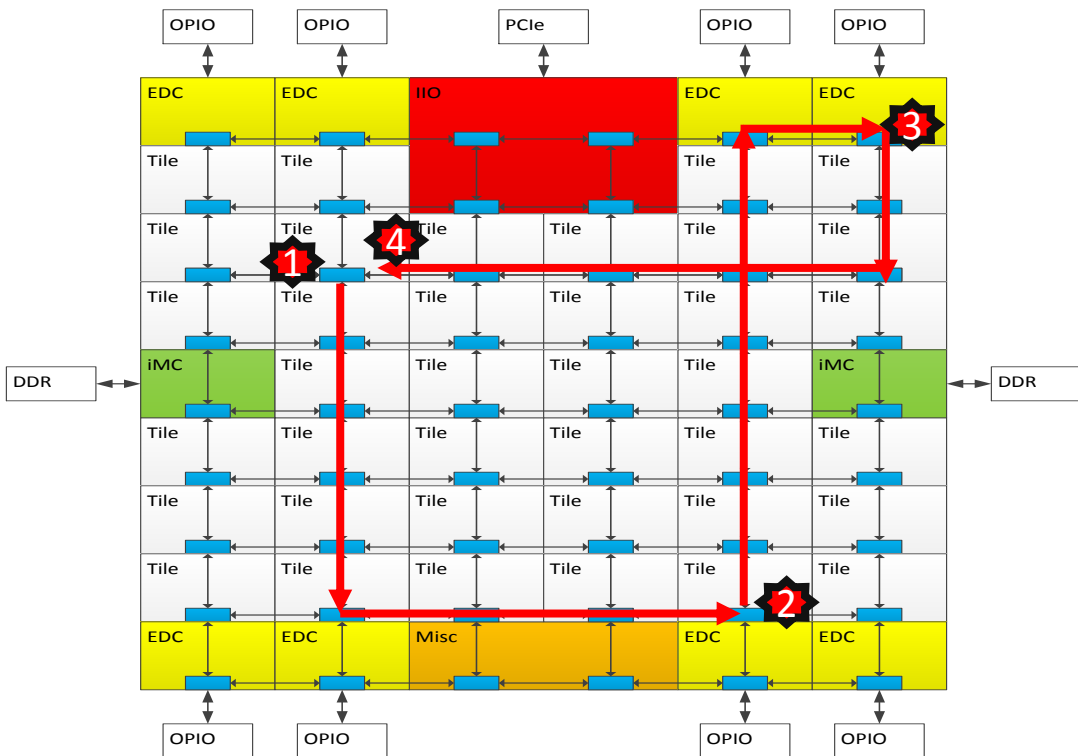
Copyright © 2016 Intel Corporation. All rights reserved.

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

Andrey Semin | 29 June 2016 | Slide 24



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

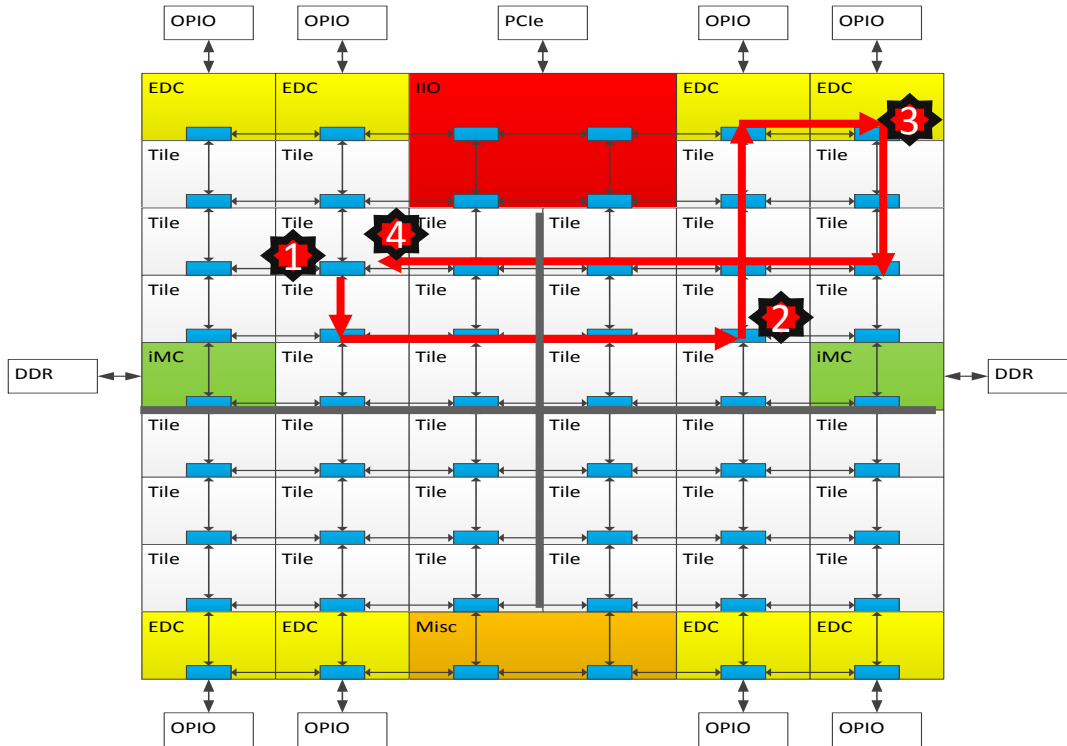
All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Copyright © 2016 Intel Corporation. All rights reserved.

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



Cluster Mode: Quadrant



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.

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



Cluster Mode: Sub-NUMA Clustering (SNC)

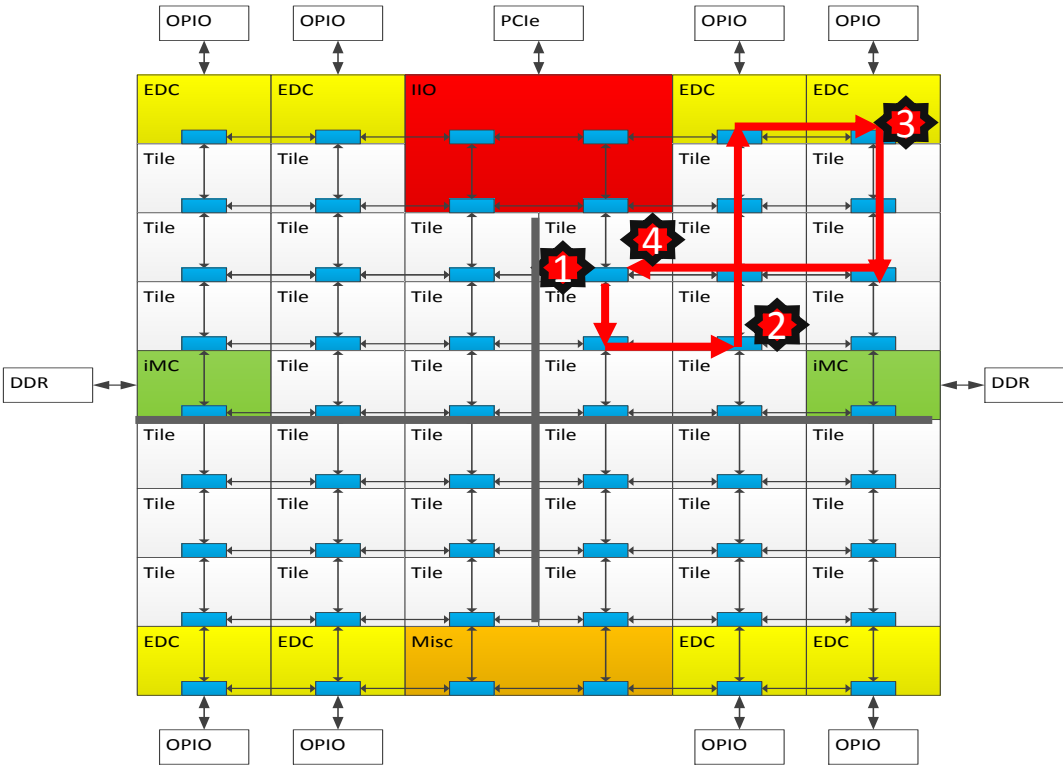
Each Quadrant (Cluster) exposed as a separate NUMA domain to OS

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



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

Copyright © 2016 Intel Corporation. All rights reserved.

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



Software visible memory configuration (numactl -- hardware)

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:
```

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
```

```
$ mpirun numactl -m 4,5,6,7 ./app
```

```
$ mpirun -perhost 16 numactl --preferred 5 ./app : numactl --preferred 6 ./app : ...
```

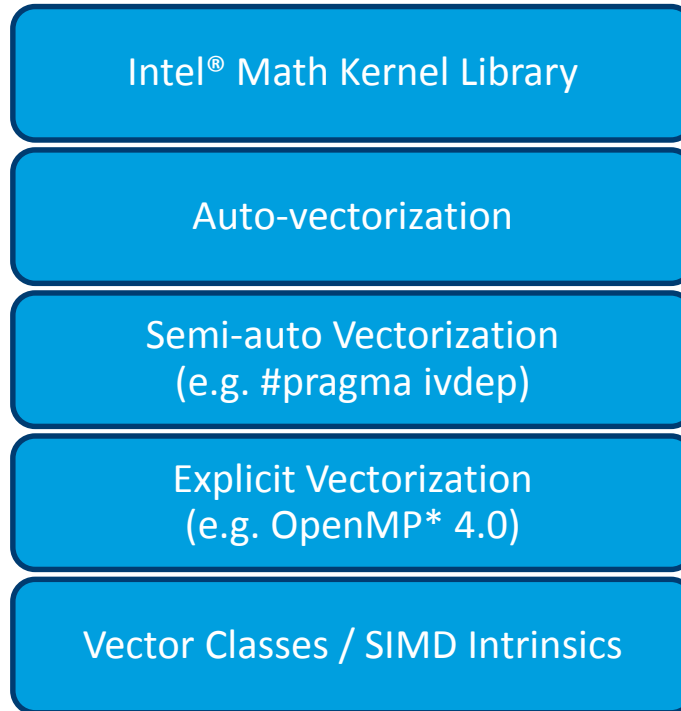
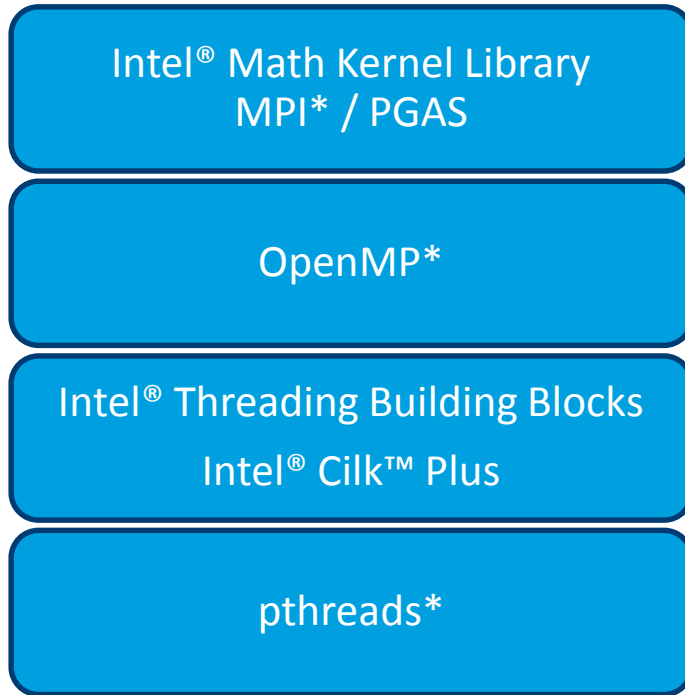
[I_MPI_HBW_POLICY](#) will be available later to simplify the command line (to omit node numbers)

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

Wide Range of Development Options

Thread/Task Parallelism

Vector Parallelism



Copyright © 2016 Intel Corporation. All rights reserved.

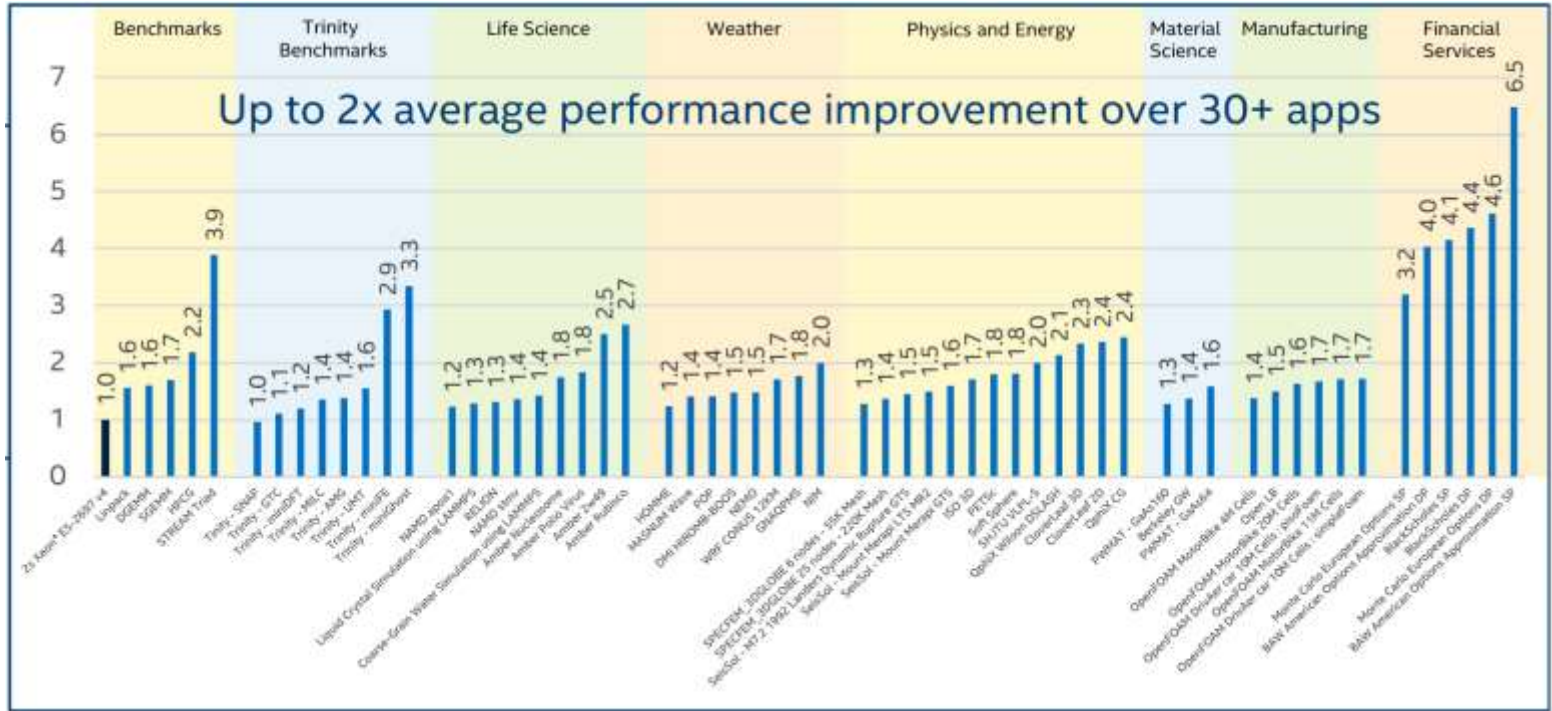
*Other names and brands may be claimed as the property of others.

Andrey Semin | 29 June 2016 | Slide 29



Greater Cores, Vectors, and Memory Bandwidth

Intel® Xeon Phi™ Processor 7250
 Relative Performance (Higher is better)
 (Normalized to 1.0 baseline of a 2 Socket
 Intel® Xeon® processor E5-2697 v4)



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 complete information visit: <http://www.intel.com/performance>
 Source: Intel measured or estimated as of May 2016.

Copyright © 2016 Intel Corporation. All rights reserved.

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



Summary

- Intel Xeon Phi 7200 series processor is the next big step for HPC:
 - 3+ TFLOPS (DP) of peak performance via new AVX-512 ISA
 - up to 16GB integrated MCDRAM to with up to 490 GB/s on STREAM TRIAD
 - the first Intel processor with integrated Intel Omni-Path HFI
- New memory and die architecture exposed to the software developers to enable greater flexibility in ways of getting performance
 - MCDRAM can be configured as cache or flat memory region
 - The cores/tiles on die can be exposed in all-to-all, quadrant or SNC modes
- Intel Xeon Phi 7200 series processor delivers great performance improvements over latest Xeon E5-2600 v4 for highly parallel applications



Intel Xeon Phi (Knights Landing) optimization best known methods

PRACE PATC Course: Intel MIC Programming Workshop
29 June 2016

Andrey Semin
Principal Engineer
HPC Technology Manager,
Europe, Middle-East and Africa

Agenda

- Where KNL is the best choice
- Optimization directions
- AVX512 highlights
- Memory profiling



What are Target Usages?

Workload Alignment Overview



Business Processing	Analytics	Scientific	Cloud Services	Visualization & Audio	Comms	Storage
OLTP	Data Analysis & Mining	Simulation/CAE & CFD	Front End Web	Media Delivery and Transcode	Wired Networking	Analytics
File & Print	Big Data Analytics	CAD	Data Caching	Remote Visualization	Packet Processing	Business Processing
Email	Machine/Deep Learning - Training	Life Sciences – Genomics/ Sequencing	Search	Remote Gaming	Virtual Switching	Cloud Storage Object Storage Active-Archive
ERP	Machine/Deep Learning - Evaluation	Life Science - Molecular Dynamics		VDI (Clients)	Network Security	Archive/ Regulatory Compliance
CRM		Financial - Trading		Image & Video Analytics	Wireless Access	Backup/ Recovery
Application Servers		Financial - Risk		Speech & Audio	Wireless Core	Disaster Recovery
		Energy – Seismic/Reservoir		Scientific Visualization		
		Weather		Professional Rendering		
		Defense/Security				

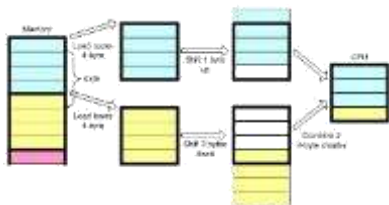
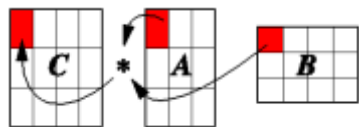
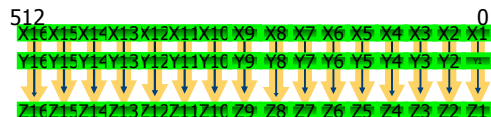
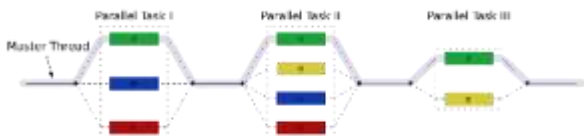
Very Applicable
Applicable
Less Common

Copyright © 2016 Intel Corporation. All rights reserved.

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



What is “Modernized” Code?



What

Defined

Tools of the trade

1 Thread Scaling

Increase concurrent thread use across coherent shared memory

OpenMP, TBB, Cilk Plus

2 Vector Scaling

Use many wide-vector (512-bit) instructions

Vector loops, vector functions, array notations

3 Cache Blocking

Use algorithms to reduce memory bandwidth pressure and improve cache hit rate

Blocking algorithms

4 Fabric Scaling

Tune workload to increased node count

MPI

5 Data Layout

Optimize data layout for unconstrained performance

AoS → SoA, directives for alignment

MPI needs help

- Many codes are already parallel (MPI)
 - May scale well, but...
 - What is single-node efficiency?
 - MPI isn't vectorising your code...
 - It has trouble scaling on large shared-memory chips.
 - ✓ Process overheads
 - ✓ Handling of IPC
 - ✓ Lack of communication aggregation off-die
- Threads are most effective for many cores on a chip
 - Adopt a hybrid thread-MPI model for clusters of many-core



OpenMP 4+

- OpenMP helps express thread- and vector-level parallelism via directives
 - (like `#pragma omp parallel`, `#pragma omp simd`)
- Portable, and powerful
- Don't let simplicity fool you!
 - It doesn't make parallel programming easy
 - There is no silver bullet
- Developer still must expose parallelism, optimize & test performance



AVX512 - Greatly increased register file

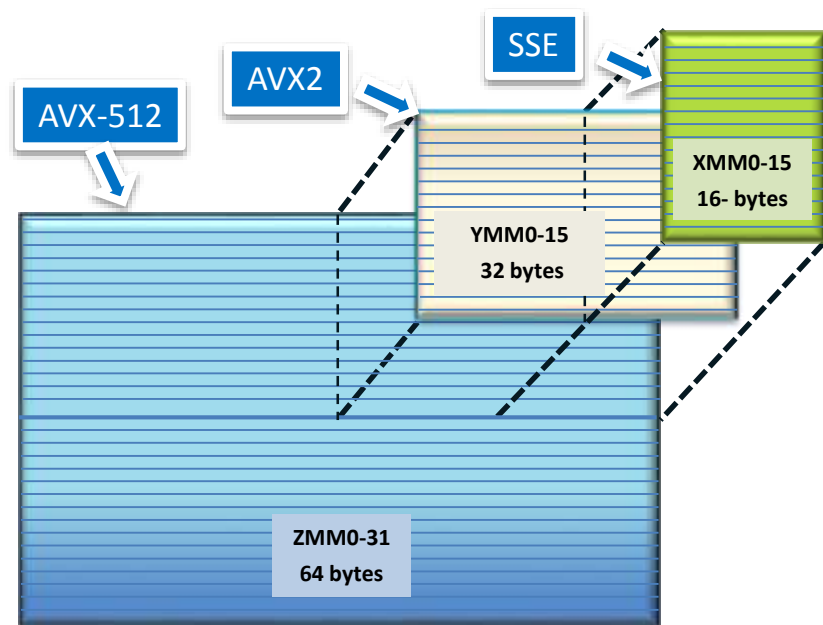
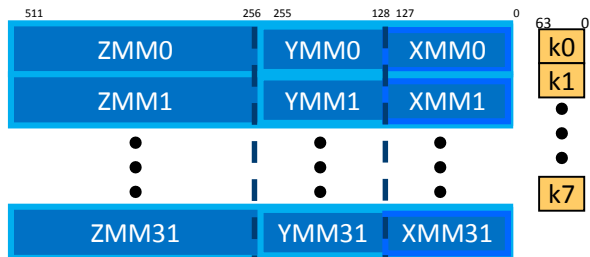
- Higher throughput
- Greatly improved unrolling and inlining opportunities

32 vector registers, 512b wide: zmm0 through zmm31

- Overlaid on top of existing YMM arch state
- Writing to xmm zeroes bits [511:128]
- writing to ymm zeroes bits [511:256]

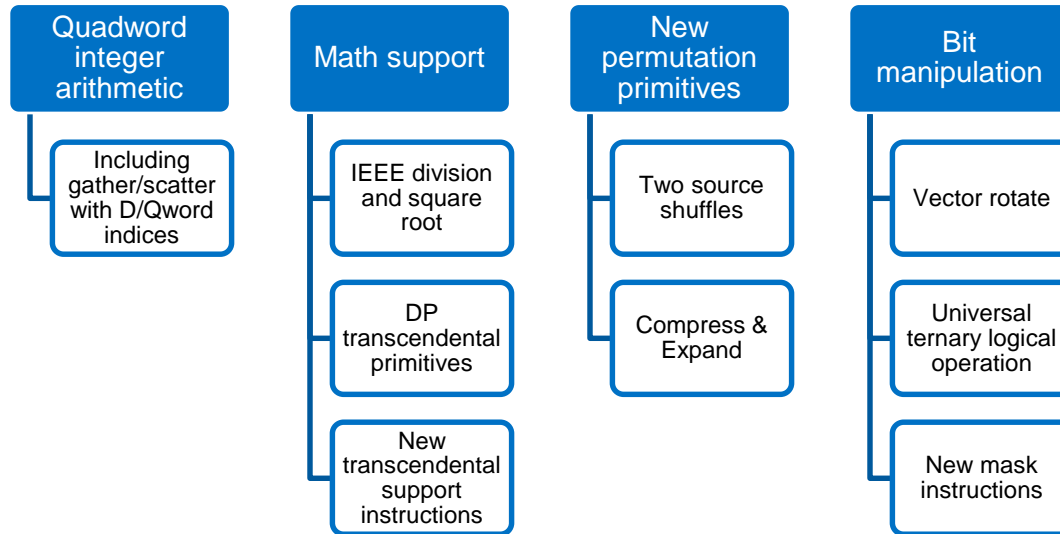
8 mask registers, 64b wide: k0 through k7

- KNL only uses bits [15:0] though (PS,PD,D,Q)
- EVEX.aaa=000 is an indicator of "no mask"
 - {k0} is illegal



AVX-512 F Designed for HPC

- Promotions of many AVX and AVX2 instructions to AVX-512
 - 32-bit and 64-bit floating-point instructions from AVX
 - Scalar and 512-bit
 - 32-bit and 64-bit integer instructions from AVX2
- Many new instructions to speedup HPC workloads



Copyright © 2016 Intel Corporation. All rights reserved.

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



Masking - new feature in AVX

8 new mask registers k0-k7

Create mask:

VCMPPS k1, zmm1, zmm2, lmm
k1 = ..0101100111 /* 16 bits */

VCMPPD k1, zmm1, zmm2, lmm
k1 = ..01011001 /* 8 bits */

Use mask:

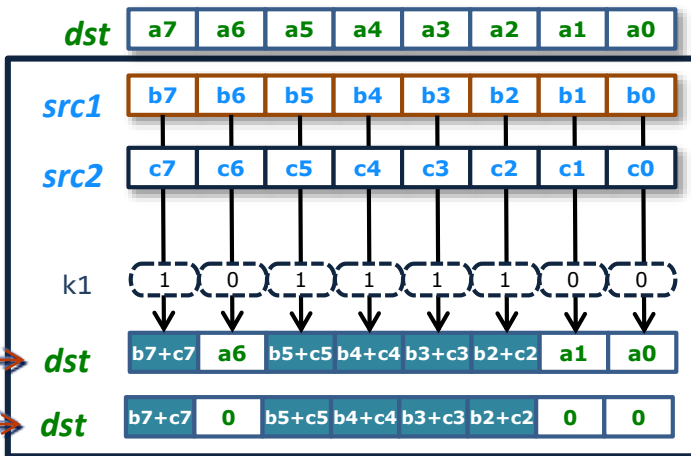
VADDPD dst {k1}, src1, src2

Unmasked elements remain unchanged:

VADDPD zmm1 {k1}, zmm2, zmm3

Or zeroed:

VADDPD zmm1 {k1} {z}, zmm2, zmm3



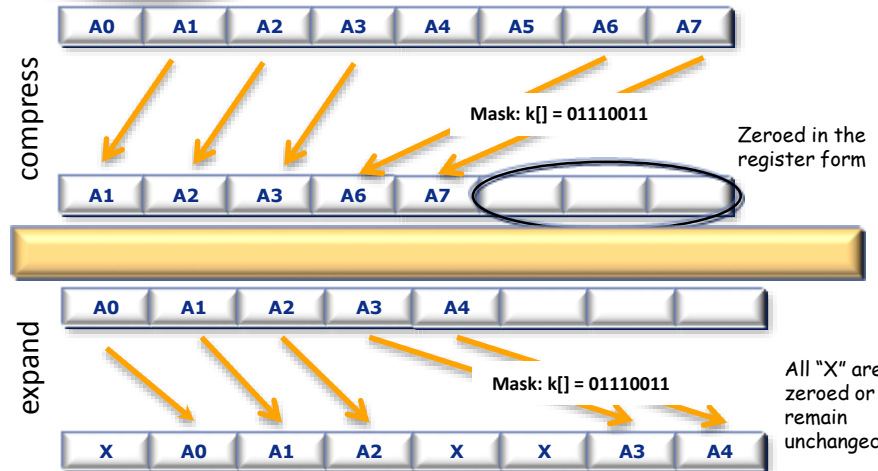
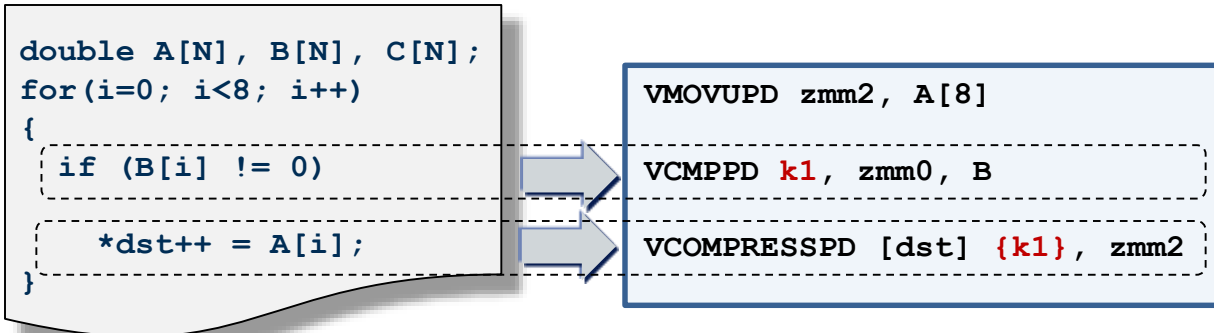
Why masking?

- Memory fault suppression
 - Vectorize code using masked load/store
 - Typical examples are if-conditional statements or loop remainders
- Avoid spurious floating-point exceptions
- Zeroing/merging
 - Use zeroing to avoid false dependencies `VADDPD zmm1 {k1} {z}, zmm2, zmm3`
 - Use merging to preserve unmasked values `VADDPD zmm1 {k1}, zmm2, zmm3`

```
float A[N], B[N], C[N];
for(i=0; i<16; i++)
{
    if (B[i] != 0)
        A[i] = A[i] / B[i];
    else
        A[i] = A[i] / C[i];
}
```

```
VMOVUPS zmm2, A[16]
VCMPPS k1, zmm0, B
VDIVPS zmm1 {k1}{z}, zmm2, B
KNOT k2, k1
VDIVPS zmm1 {k2}, zmm2, C
VMOVUPS A[16], zmm1
```

Expand & Compress



Motivation for Conflict Detection



- Sparse computations are common in HPC, but hard to vectorize due to race conditions
- Consider the “histogram” problem:

```
for(i=0; i<16; i++) { A[B[i]]++; }
```

```
index = vload &B[i]           // Load 16 B[i]
old_val = vgather A, index     // Grab A[B[i]]
new_val = vadd old_val, +1.0   // Compute new values
vscatter A, index, new_val     // Update A[B[i]]
```

- Code above is wrong if any values within B[i] are duplicated
 - Only one update from the repeated index would be registered!
- A solution to the problem would be to avoid executing the sequence gather-op-scatter with vector of indexes that contain conflicts

Conflict Detection - how does it work?

Iteration 1	mask	1	1	1	1	1	1	1	
	indices	9	3	2	2	2	7	8	7
	conflict-free mask	1	1	1	0	0	1	1	0
Iteration 2	mask	0	0	0	1	1	0	0	1
	indices	9	3	2	2	2	7	8	7
	conflict-free mask	0	0	0	1	0	0	0	1
Iteration 3	mask	0	0	0	0	1	0	0	0
	indices	9	3	2	2	2	7	8	7
	conflict-free mask	0	0	0	0	1	0	0	0

Copyright © 2016 Intel Corporation. All rights reserved.

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



Conflict Free Code

```
for(i=0; i<16; i++)  
{  
    j = B[i];  
    A[j]++;  
}
```

```
j = vload &B[i]  
pending_elts = 0xFFFF;  
do {  
    mask = conflict_free(j, pending_elts)  
    val_A = vgather {mask} A, j           // Grab A[j]  
    val_A++                               // Compute new values  
    vscatter A {mask}, j, val_A          // Update A[j]  
    pending_elts ^= mask                 // remove done idx  
} while (pending_elts)
```

CDI instr.

VPCONFLICT{D,Q}	zmm1{k1}, zmm2/mem
VPBROADCASTM{W2D,B2Q}	zmm1, k2
VPTESTNM{D,Q}	k2{k1}, zmm2, zmm3/mem
VPLZCNT{D,Q}	zmm1 {k1}, zmm2/mem

VPCONFLICT instruction detects elements with previous conflicts in a vector of indexes

Allows to generate a mask with a subset of elements that are guaranteed to be conflict free

AVX-512 ERI

Provide building blocks for accelerating certain transcendental math ops

- `Vexp2[pd|ps]`
 - Approximation of the exponential 2^x (maskable)
 - 2^{-23} of max relative error
- `Vrcp28[pd|sd|ps|ss]`
 - Computes the approximate reciprocal (maskable)
 - $<2^{-28}$ relative error
- `Vrsqrt28[pd|sd|ps|ss]`
 - Computes the approximate reciprocal square root (maskable)
 - $<2^{-28}$ relative error



Prefetch Instructions (PFI)

Useful for reducing memory operation latency due to gather/scatter instructions

- `Vgatherpf0[dps|qps|dpd|qpd]`, `Vgatherpf1[dps|qps|dpd|qpd]`
 - Sparse conditional prefetch of up to 16x32 bit, or 8x64bit memory locations
 - Instruction is a hint (T0 or T1), prefetches may not occur
 - T0, T1 specify different cache levels
 - Maskable
- `Vscatterpf0[dps|qps|dpd|qpd]`, `Vscatterpf1[dps|qps|dpd|qpd]`
 - Sparse conditional prefetch of up to 16x32 bit, or 8x64 bit memory locations
 - Cache lines will be brought into exclusive state (RFO)
 - T0, T1 specify different cache levels
 - Maskable



AVX-512 - Summary

- Performance impact of AVX-512:
 - 2x increase in vector width (vs. AVX2)
 - Improved [gather/scatter efficiency](#) (vs. mov/insertps/extractps sequence and IMCI)
 - Improved control divergence management via [masking](#) (vs. AVX2)
- Programmability impact of AVX-512:
 - Auto-vectorization of [loops with potential dependencies](#) (via vconflict)
 - Improved vectorization of outer [loops with complicated flow](#) (via masking)



KNL Resources: AVX-512

- <https://software.intel.com/en-us/blogs/2013/avx-512-instructions>
- <https://software.intel.com/en-us/blogs/additional-avx-512-instructions>
- Intel® Architecture Instruction Set Extensions Programming Reference
- Intel® 64 and IA-32 Architectures Software Developer Manuals



Utilizing SIMD - Intel® Intrinsic Guide



The Intel Intrinsic Guide is an interactive reference tool for Intel Intrinsic instructions, which are C style functions that provide access to many Intel instructions - including Intel® SSE, AVX, and more - without the need to write assembly code.

`__m512d __mm512_abs_pd (__m512d v2)` vpandq

Synopsis

```
__m512d __mm512_abs_pd (__m512d v2)
#include "zmmintrin.h"
Instruction: vpandq zmm {k}, zmm, m512
CPUID Flag : KNCNI
```

Description

Finds the absolute value of each packed double-precision (54-bit) floating-point element in v2, storing the results in dst.

Operation

```
FOR j := 0 to 7
  i := j*64
  dst[i+63:i] := ABS(v2[i+63:i])
ENDFOR
dst[MAX:512] := 0
```

Expand any intrinsic for a detailed description.

Filter by ISA.

Technologies

- MMX
- SSE
- SSE2
- SSE3
- SSSE3
- SSE4.1
- SSE4.2
- AVX
- AVX2
- FMA
- AVX-512
- KNC
- SVM
- Other

Categories

- Application-Targeted
- Arithmetic
- Bit Manipulation
- Cast
- Compare
- Convert
- Cryptography
- Elementary Math Functions
- General Support

Filter by functionality.

Available at: <http://software.intel.com/sites/landingpage/IntrinsicGuide/>

Copyright © 2016 Intel Corporation. All rights reserved.

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

Andrey Semin | 29 June 2016 | Slide 50



Lessons from Previous Architectures

- Vectorization:
 - Avoid **cache-line splits**; align data structures to **64 bytes**.
 - Avoid **gathers/scatters**; replace with shuffles/permutates for known sequences.
 - Avoid **mixing** SSE, AVX and AVX512 instructions.
- Threading:
 - Ensure that thread **affinities** are set.
 - Understand affinity and how it affects your application (i.e. which threads share data?).
 - Understand how threads share core resources.



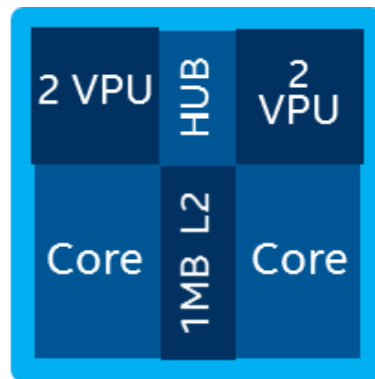
Lessons from Previous Architectures

- Memory:
 - Tile your algorithm to take advantage of data reuse
 - Layout data to avoid TLB misses and assist vectorisation (AOS vs. SOA, large pages)
 - Consider the need for software prefetches
 - Understand how threading affects cache and TLB pressure



Data Locality: Nested Parallelism

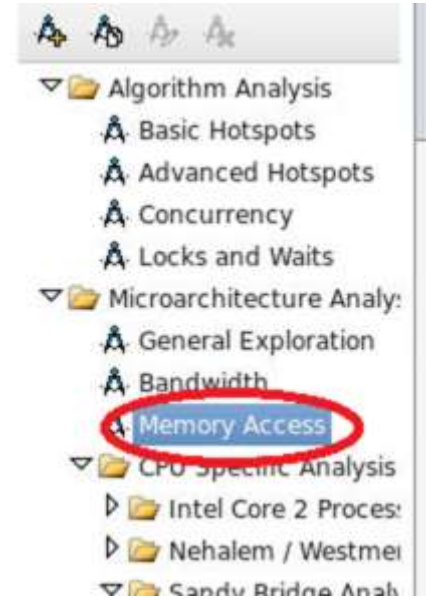
- Recall that KNL cores are grouped into tiles, with two cores sharing a 1MB L2.
- Effective capacity depends on locality:
 - 2 cores sharing no data => 2 x 512 KB
 - 2 cores sharing all data => 1 x 1 MB
- Ensuring good locality (e.g. through blocking or nested parallelism) is likely to improve performance.



```
#pragma omp parallel for num_threads(ntiles)
for (int i = 0; i < N; ++i)
{
    #pragma omp parallel for num_threads(8)
    for (int j = 0; j < M; ++j)
    {
        ...
    }
}
```

Memory Profiling

- Intel® VTune™ Amplifier 2016 introduces a “Memory Access” analysis type for tracking down various memory-related issues:
 - NUMA problems: applicable to MCDRAM in KNL
 - Bandwidth analysis
- On Linux, it instruments memory allocations/deallocations to find “memory objects”
 - and correlates these objects with performance events.
- Command-line usage:
`amplxe-cl -c memory-access -data-limit=0 -knob analyze-mem-objects=true -knob mem-object-size-min-thres=1024 -- <app>`

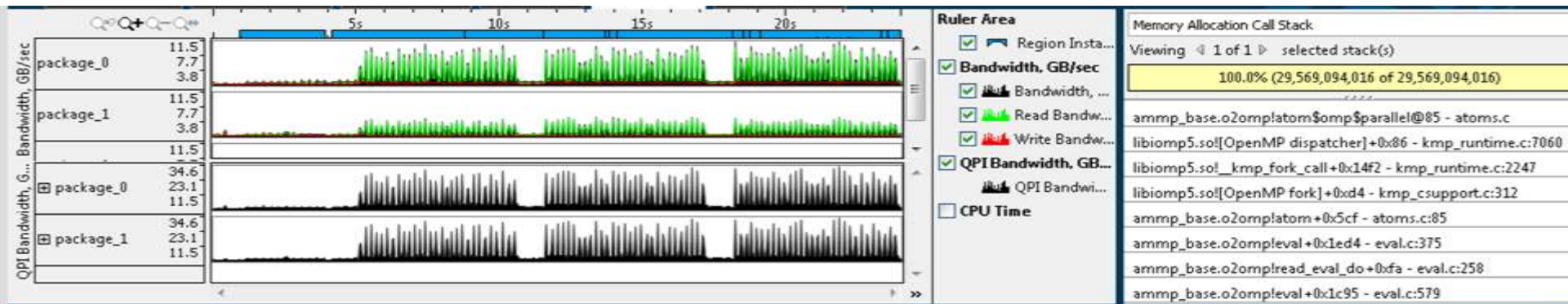


Typical workflow for KNL HBM analysis

- Select new grouping: “Function / Memory Object / Allocation stack”
- Sort by “loads”
 - But can use other metrics: “LLC Miss”, “Stores” etc
- Expand functions with high bandwidth estimates and examine memory objects accessed by it:
 - It is highly likely that the most referenced memory objects in the high-bandwidth function are also bandwidth limited.
- HW prefetching can hide significant amounts of misses, consider temporarily disabling prefetching during the analysis



Memory Profiling - Memory Objects View



Grouping: Function / Memory Object / Function Stack

Function / Memory Object / Function Stack	CPU Time	Loads	Stores	LLC Misses	Remote DRAM	Remote Cache	Memory Bound	Average Late ...	KNL Bandwidth ...	Start A...
mm_fv_update_nonbon\$omp\$parallel_for@593	97.449s	61,550,277,276	11,565,173,475	1,037,731,131	0	1,048,831,464	0.579	23	3.494	0x4153f7
atoms.c:88	0.002s	29,569,094,016	261,003,915	1,811,454,342	0	1,799,453,982	0.000	69	0.000	0x4153f7
atom\$omp\$parallel@85 - [OpenMP dispatcher]	0.002s	14,784,547,008	258,603,879	905,727,171	0	899,726,991	0.000	69	0.000	0x408fb8
rectmm.c:217	0.005s	20,910,773,670	0	114,603,438	0	141,004,230	0.000	11	0.000	0x4153f7
mm_fv_update_nonbon - fv_update_nonbon	0.002s	10,455,386,835	0	57,301,719	0	70,502,115	0.000	11	0.000	0x4147d0
rectmm.c:160	0.002s	1,648,062,826	0	97,202,916	0	103,203,946	0.000	23	0.000	0x4153f7





Conclusion:



Common code, tools
and developers



“[...] porting to the Intel Xeon Phi processor only requires a simple recompile, and it took us less than a week to hand-tune our kernels for AVX512. For the first time, this will enable us to have a single set of kernels that work both on many core Xeon Phi and future multi-core Xeon processors.”

– Erik Lindahl of KTH and Stockholm University*,
GROMACS* Project Leader*

Extreme parallel performance with general purpose programming



