



# GOING PARALLEL WITH EFFICIENCY ON THE FUTURE KNIGHTS FAMILY

CJ Newburn, SW/HW HPC Architect, Community Builder

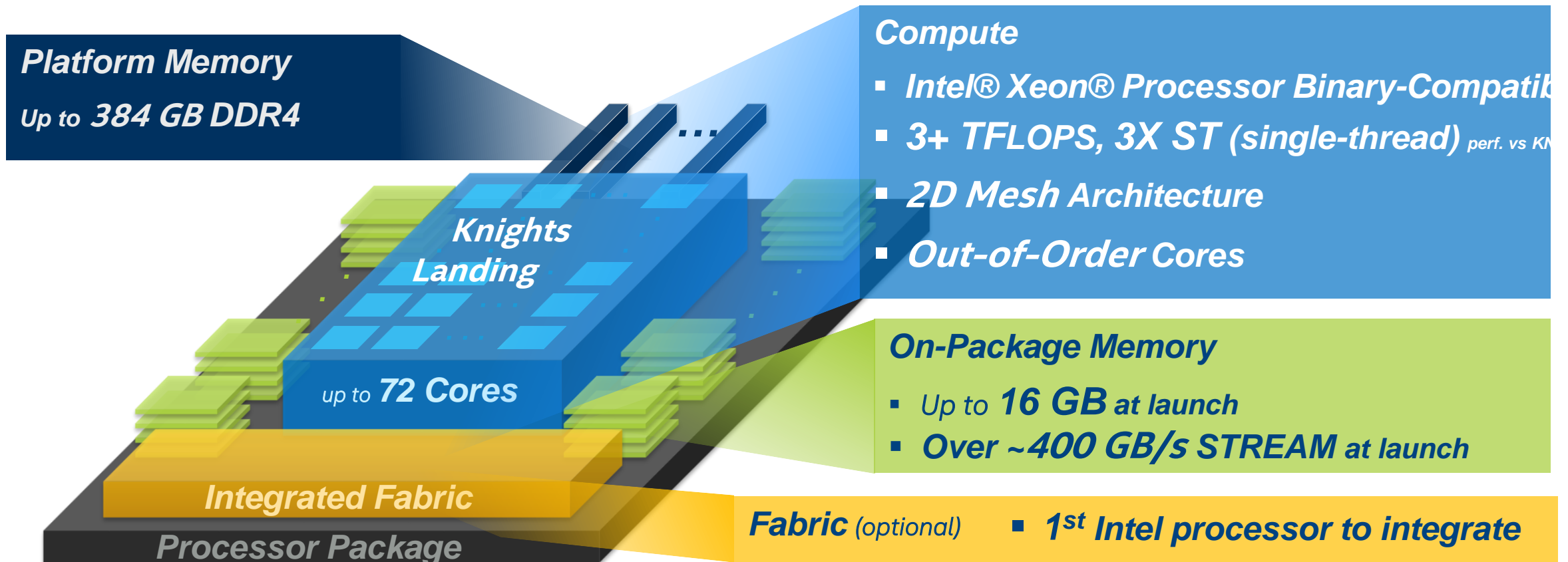
Nov 18, 2015 SC15 IXPUG BoF:

Paving the way for Performance on Intel® Knights Landing Processors and beyond:

Unleashing the Power of Next-Generation Many-Core Processors

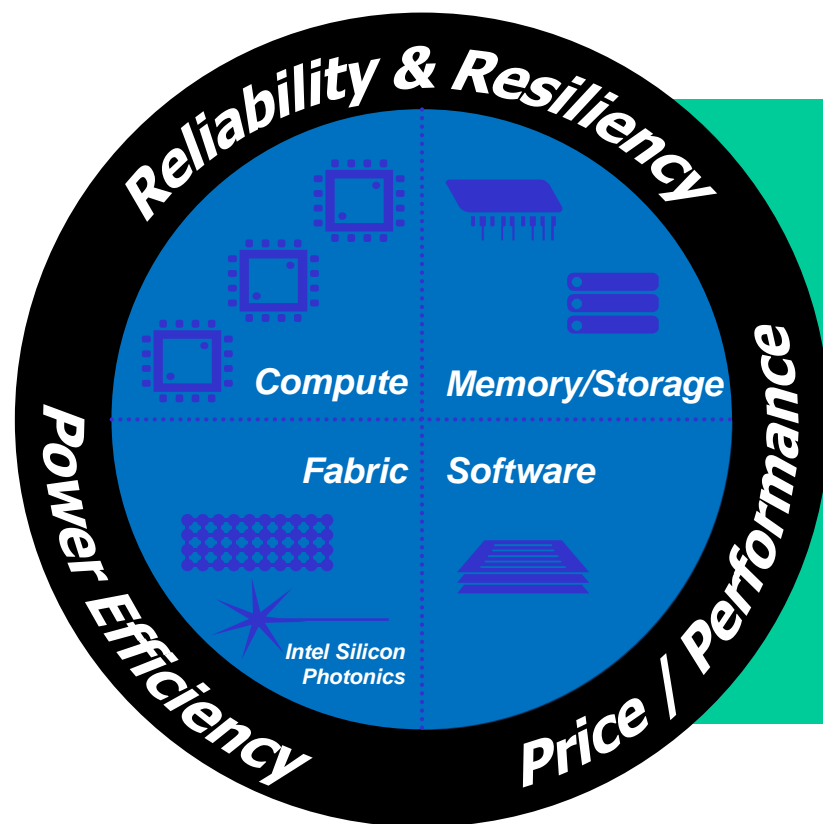
# Intel® Xeon Phi™ x200 Product Family

(codename Knights Landing)



# Intel's HPC Scalable System Framework

*A design foundation enabling wide range of highly workload-optimized solutions*



*Small Clusters Through Supercomputers*  
*Compute and Data-Centric Computing*  
*Standards-Based Programmability*  
*On-Premise and Cloud-Based*

**Intel® Xeon® Processors**

**Intel® Xeon Phi™  
Coprocessors**

**Intel® Xeon Phi™  
Processors**

**Intel® True Scale Fabric**

**Intel® Omni-Path  
Architecture**

**Intel® Ethernet**

**Intel® SSDs**

**Intel® Lustre-based Solutions**

**Intel® Silicon Photonics  
Technology**

**Intel® Software Tools**

**HPC Scalable Software Stack**

**Intel® Cluster Ready  
Program**

# Got scale.

# Got efficiency?

## HW component

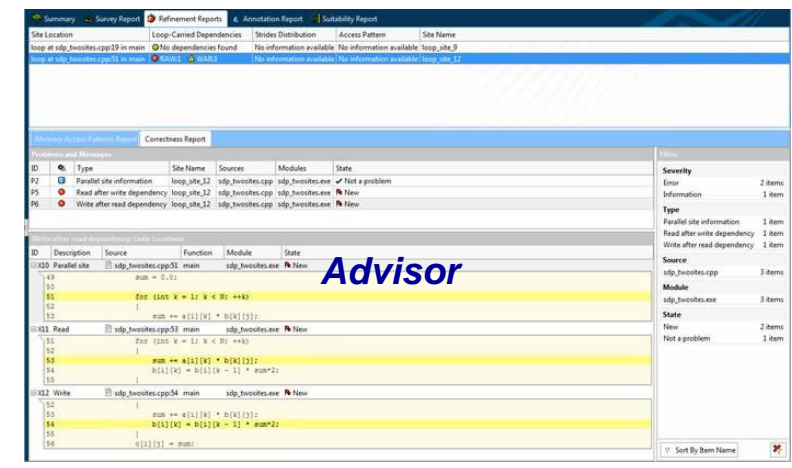
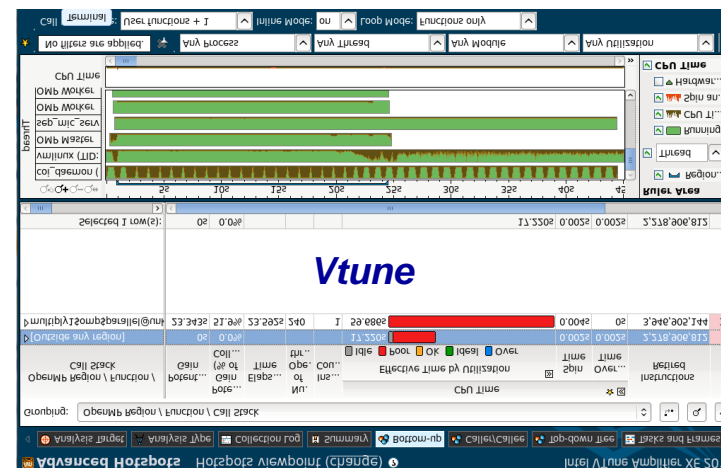
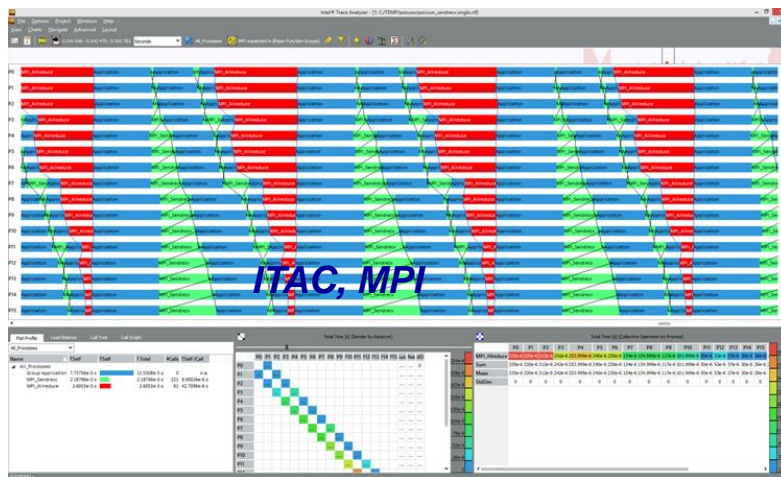
- Nodes
- HW threads
- SIMD lanes
- Mem bw, latency

## SW abstraction

- Ranks
- Tasks
- SW threads
- Vectors
- Tiles

## What you use

- MPI, omp target, hStreams
- omp task, TBB, hStreams
- omp parallel for, TBB for
- omp simd packing?
- Memory preconditioning for locality



*What about YOUR efficiency?*

## Go it alone

- Start from scratch
- Figure out what matters
- Reinvent
- Quit when it looks good enough
- Keep it to yourself

## Join the village

- Stand on others' shoulders
- See what didn't matter
- Enhance
- Gain deeper understanding
- Share your insights

## TECHNIQUES

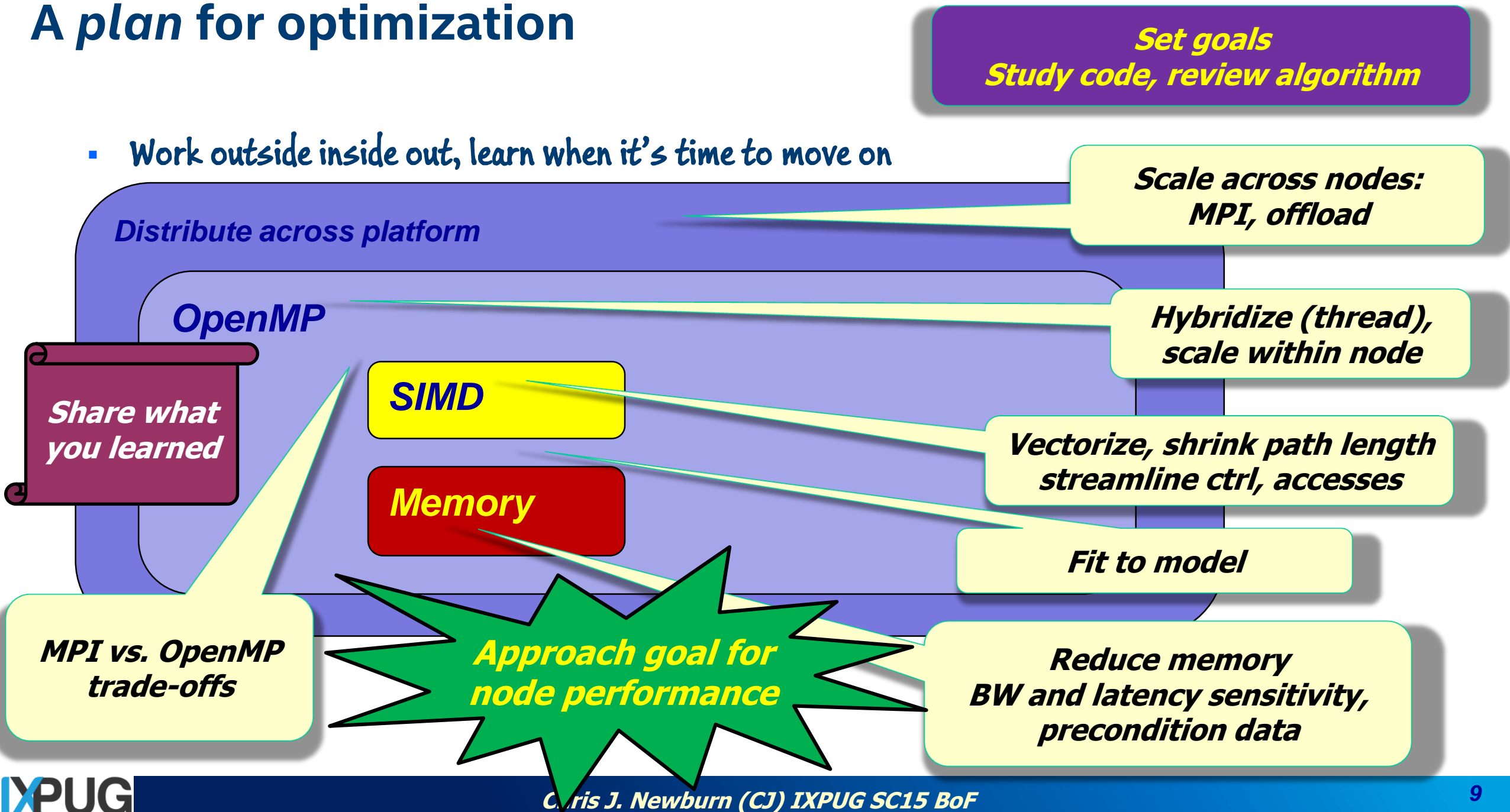
**Tools**

*Experiment design*

**Working groups**

# A plan for optimization

- Work outside inside out, learn when it's time to move on



# Trends that are here to stay

## Data parallelism

- Lots of threads, spent on MPI ranks or OpenMP/TBB/pthreads
- Improving support for both peak tput and modest/single thread
- *Can we keep this debuggable with a sequential semantic?*

## Bigger, better, faster, persistent memory

- High capacity, high bandwidth, lower latency, persistent DRAM
- Effective caching and paging
- Increasing support for irregular memory refs, modest tuning
- *Can we manage these with a declarative interface?*

## ISA innovation

- Increasing support for vectorization, new usages

*Give the tools more guidance, or create new language interfaces?*



# Fundamental or incremental code changes?

Incremental changes, significant gains 😊

Parallelization – **consistent** strategy

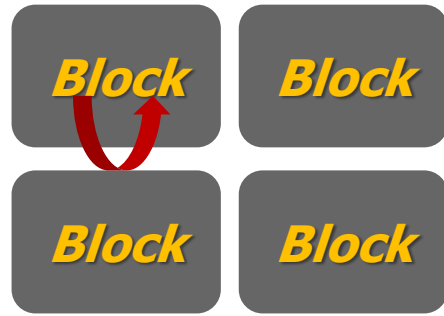
- MPI vs. OpenMP – already needed to tune and tweak
- Less thread-level parallelism required
- Vectorization – **more opportunity, more profitable**

**Enable new features** with memory tuning

- Access **MCDRAM** with special allocation
- Blocking for MCDRAM vs. just cache

# Tolerating a lack of locality, before well tuned

*We strongly encourage tuning!*



*Lower bandwidth  
Less sensitive to latency*



*But until you get there, we help tolerate latency*



*High bandwidth demand  
Outstanding misses enqueued  
Sensitive to latency*

**$\leq 16\text{GB}$  MCDRAM**

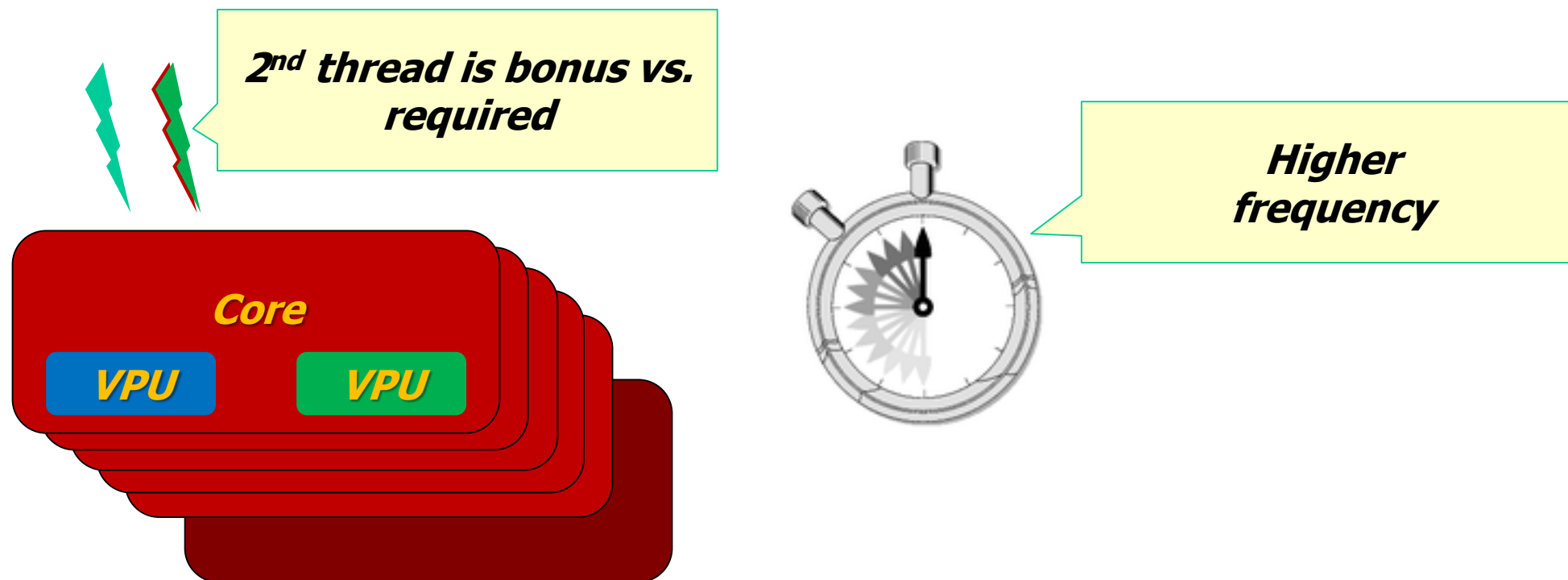
**KNL DDR – up to 384GB**

**Tuning helps locality: lower BW demand, shorter latency**

**Until better tuning is achieved, we make the on-ramp gentler:**

**→ higher BW support, shorter latency to DRAM, more outstanding misses**

# Enhanced core



**3x higher peak, up to 3x better single-thread performance**  
**2 VPUs, higher frequency, more cores, fewer issue restrictions**  
**Gather/scatter with lower overhead**

<http://www.wpclipart.com/time/stopwatch/>

# KNL-specific SW enabling

## Recompilation, with `-xMIC-AVX512`

- Unaligned support
- Access to faster svml routines than earlier Xeon

## Threading

- Now possible to support many more MPI ranks, given larger memory capacity
- 1 thread per core is now often sufficient

## Vectorization

- New cases become efficient: compress, expand, index conflicts

## MCDRAM and memory tuning

- Selectively allocate memory, in non-cached mode
- Tile, if working set doesn't fit in MCDRAM
- Consider using 1GB pages

## Offload

- Continues to be supported, for compiler, hStreams, MKL AO

# How tools help provide insight

## Development tools

- Several tools for MCDRAM allocations: memkind, numactl, libhugetlbfs, ...
- Intel® Software Development Emulator, v 7.5+
- hStreams supports HBM allocation, task concurrency, offload

## Analysis tools

- VTune “memory” analysis for high bandwidth memory analysis

*Focus on insight: Don't forget to correlate performance results with PerfMon data*

# Manual steps

## Building

- Change compiler switches in make files

## Coding

- Parallelization: vectorization, offload
- Memory management: MCDRAM enumeration and memory allocation

## Tuning

- Potentially fewer threads: more cores but less need for SMT
- More memory → more MPI ranks

# Take-aways

**Keep doing what you were doing for KNC and Xeon**

**Some goodness comes for free with a recompile**

**With some extra enabling, use new MCDRAM feature**

*Going parallel: how do you scale?*

*With efficiency: your code, you, community*

*Surpass the mundane: leverage the tribe*



# IXPUG working groups

- MIC Tuning – umbrella
- Data preconditioning for locality
- Life sciences – starting Dec
- New memory types – starting Dec
- General vectorization
- Vector packing and scheduling
- Floating point precision
- Nested parallelism
- Performance portability
- New MPI features

# Some examples of thinking bigger

- How can you share the good judgment you developed from experience?
- Does your work give rise to a collaborative research agenda?
- Who could you approach to get traction with?
  
- Filing bugs and feature requests is an investment in the future
- Business-motivated test cases and reproducers are key

# Backup

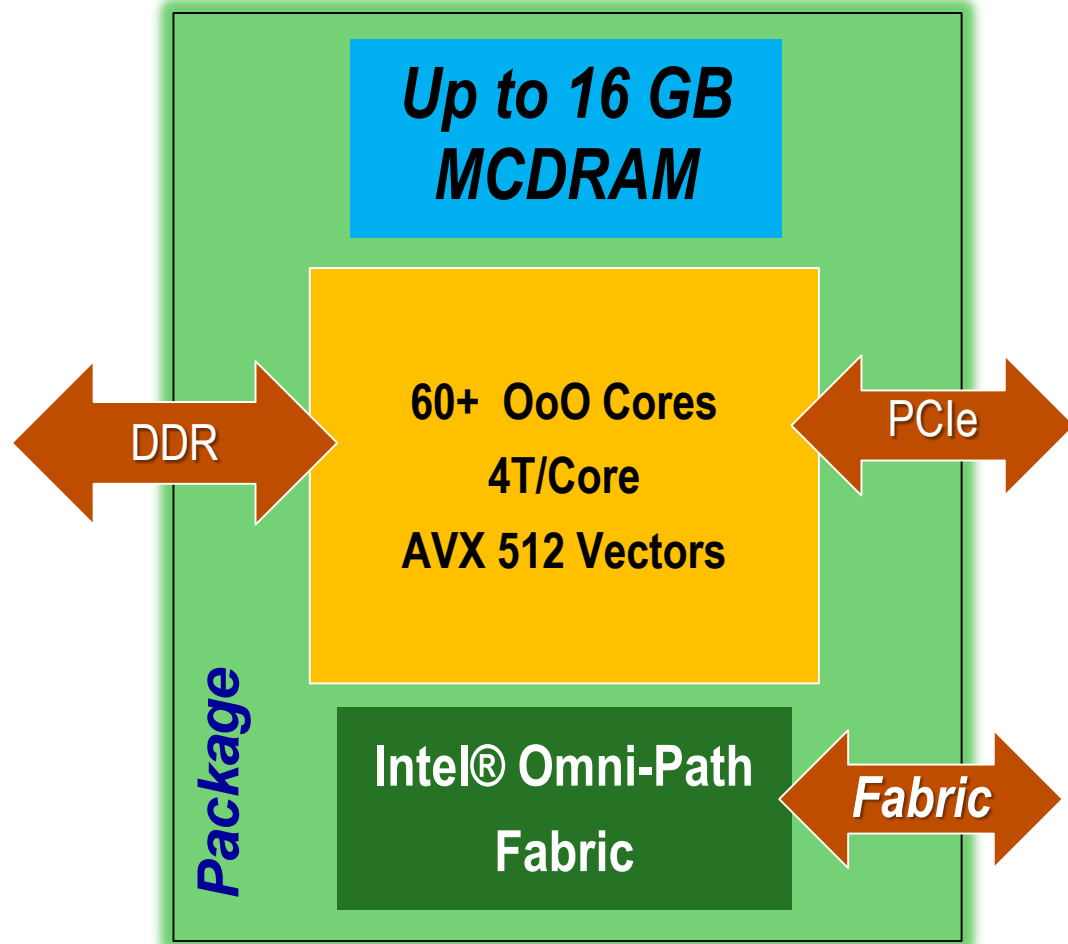
- Resources
- What's public
- MCDRAM
- Expand and compress
- Conflict detection details
- Broadcast details
- Overview of tools



# References

- [Porting guide](#)
- [Intrinsics for Intel® AVX-512 instructions](#) : the online document of the User and Reference Guide for the Intel® C++ Compiler
- [The Intel® Intrinsics Guide](#) : the interactive reference tool for Intel intrinsic instructions
- [Intel® Architecture Instruction Set Extensions Programming Reference](#)
- **The Intel Instruction Set Architecture Extensions page:**  
<https://software.intel.com/en-us/intel-isa-extensions>

# Knights Landing Overview



- Stand-alone, Self-boot CPU
- 60+ new Silvermont-based cores
- 4 Threads per core
- AVX 512 vector units
- Binary Compatible<sup>1</sup> with Intel® Xeon® processor
- 2-dimensional Mesh on-die interconnect

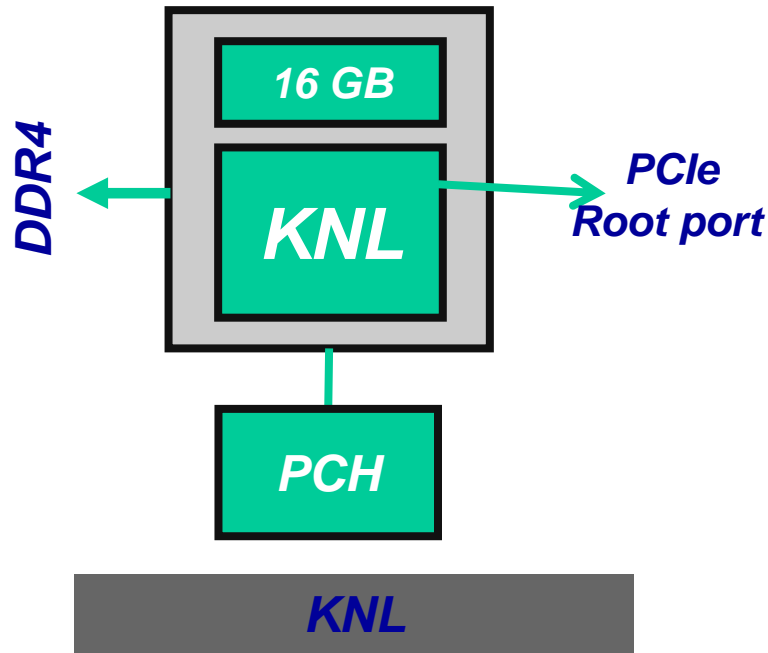
MCDRAM: On-Package memory: 400+ GB/s of BW<sup>2</sup>

- DDR memory
  - Intel® Omni-path Fabric
  - 3+ TFLOps (DP) peak per package
- ~3x ST performance over KNC

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. <sup>1</sup>Binary Compatible with Intel Xeon processors using Haswell Instruction Set (except TSX). <sup>2</sup>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.

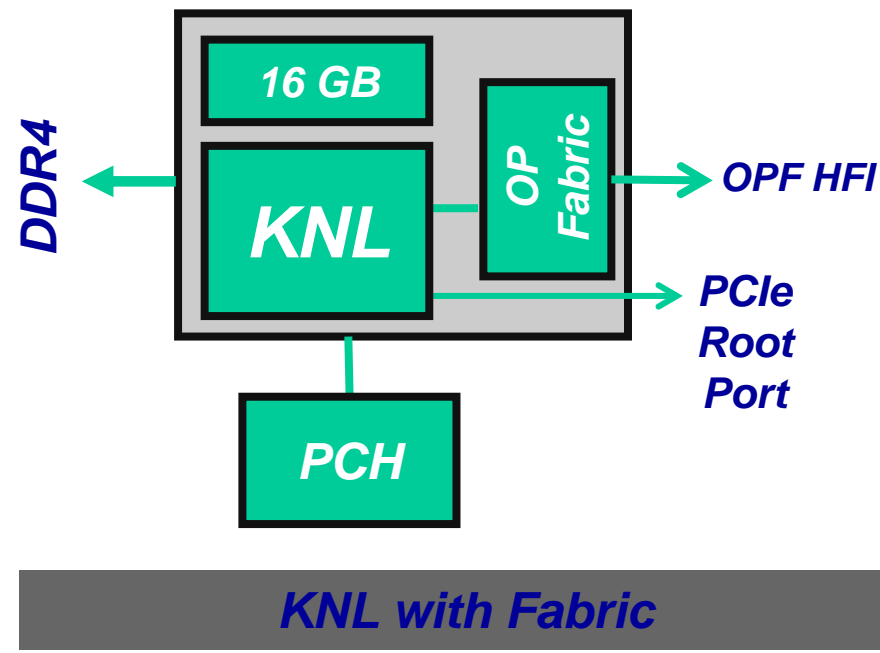
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may 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>

# Knights Landing Products

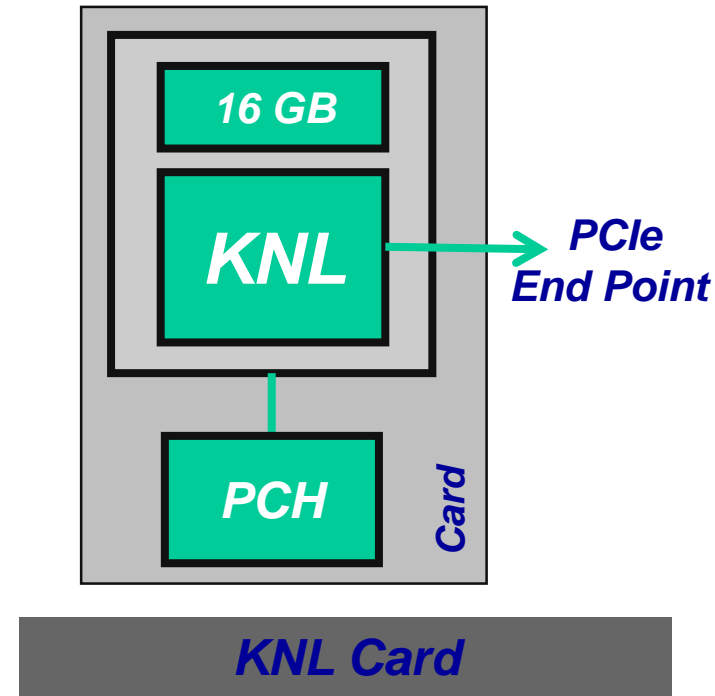


**DDR**  
**MCDRAM: up to 16 GB**  
**Gen3 PCIe (Root port)**

**Self Boot Socket**



**DDR**  
**MCDRAM: up to 16 GB**  
**Gen3 PCIe (Root port)**  
**Omni-Path Fabric**



**No DDR Channels**  
**MCDRAM: up to 16 GB**  
**Gen3 PCIe (End point)**

**PCIe Card**

Potential future options subject to change without notice. Codenames.

All timeframes, features, products and dates are preliminary forecasts and subject to change without further notification.

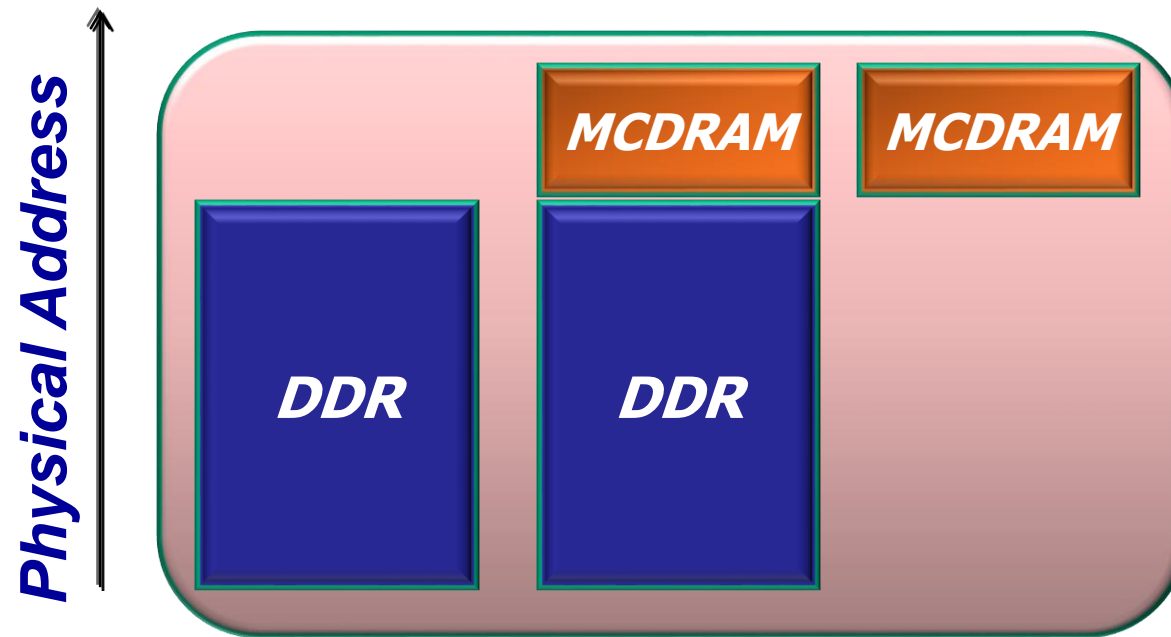
# Beyond AVX-512 Foundation

- Intel AVX-512 Prefetch Instructions (PFI)
- Intel AVX-512 Exponential and Reciprocal Instructions (ERI)
- Intel AVX-512 Conflict Detection Instructions (CDI)

CPUID	Instructions	Description
AVX512PF	PREFETCHWT1	Prefetch cache line into the L2 cache with intent to write
	VGATHERPF{D,Q}{0,1}PS	Prefetch vector of D/Qword indexes into the L1/L2 cache
	VSCATTERPF{D,Q}{0,1}PS	Prefetch vector of D/Qword indexes into the L1/L2 cache with intent to write
AVX512ER	VEXP2{PS,PD}	Computes approximation of $2^x$ with maximum relative error of $2^{-23}$
	VRCP28{PS,PD}	Computes approximation of reciprocal with max relative error of $2^{-28}$ before rounding
	VRSQRT28{PS,PD}	Computes approximation of reciprocal square root with max relative error of $2^{-28}$ before rounding
AVX512CD	VPCONFLICT{D,Q}	Detect duplicate values within a vector and create conflict-free subsets
	VPLZCNT{D,Q}	Count the number of leading zero bits in each element
	VPBROADCASTM{B2Q,W2D}	Broadcast vector mask into vector elements

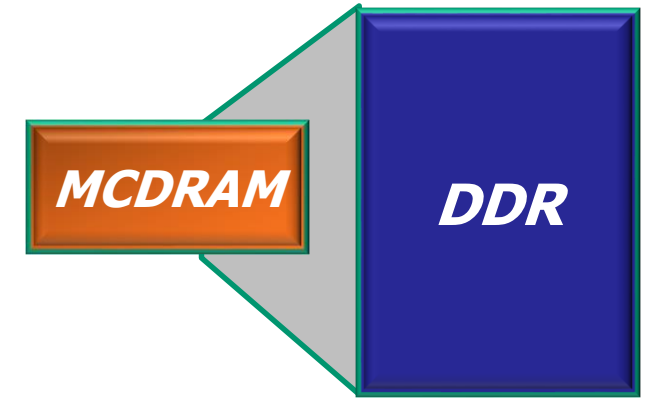
### 3 Memory Modes

- *Mode selected at boot*
- *MCDRAM-Cache covers all DDR*

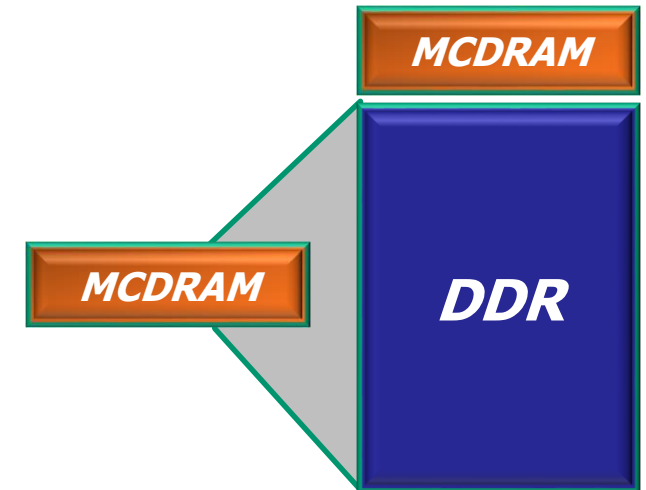


*Flat Models*

### *Cache Model*



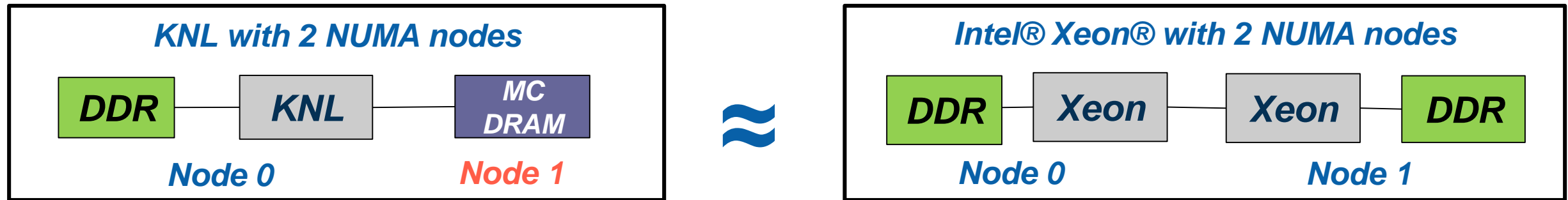
### *Hybrid Model*





# Flat MCDRAM: SW Architecture

*MCDRAM exposed as a separate NUMA node*



Memory allocated in DDR by default

- Keeps low bandwidth data out of MCDRAM.

Apps explicitly allocate important data in MCDRAM

- “**Fast Malloc**” functions: Built using NUMA allocations functions
- “**Fast Memory**” Compiler Annotation: For use in Fortran.

***Flat MCDRAM using existing NUMA support in Legacy OS***

# Flat MCDRAM SW Usage: Code Snippets

## Allocate 1000 floats from DDR

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

## Allocate 1000 floats from MCDRAM

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

## Allocate arrays from MCDRAM & DDR in Intel FORTRAN

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

***Keeping the App Effort Level Low***

# 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);
```

**Publicly released at <https://github.com/memkind>**

# Expand & Compress

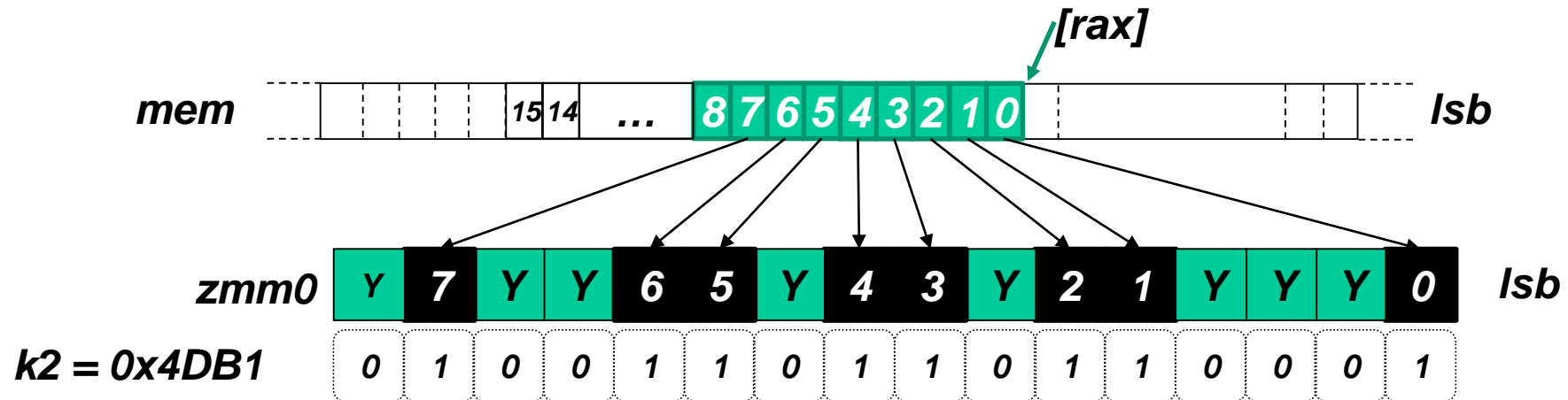
*Allows vectorization of conditional loops*

- *Opposite operation (compress) in AVX512F*
- *Similar to FORTRAN pack/unpack intrinsics*
- *Provides memory fault suppression*
- *Faster than alternative gather/scatter*

```
for(j=0, i=0; i<N; i++) {  
    if(C[i] != 0.0) {  
        B[i] = A[i] * C[j++];  
    }  
}
```

VEXPANDPS zmm0 {k2}, [rax]


Moves compressed (consecutive) elements in register or memory to sparse elements in register (controlled by mask), with merging or zeroing



# 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 Instructions in AVX-512

- **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
  - The computation loop can be re-executed with the remaining elements until all the indexes have been processed

## CDI instr.

8

VPCONFLICT{D,Q} zmm1{k1}, zmm2/mem

VPBROADCASTM{W2D,B2Q} zmm1, k2

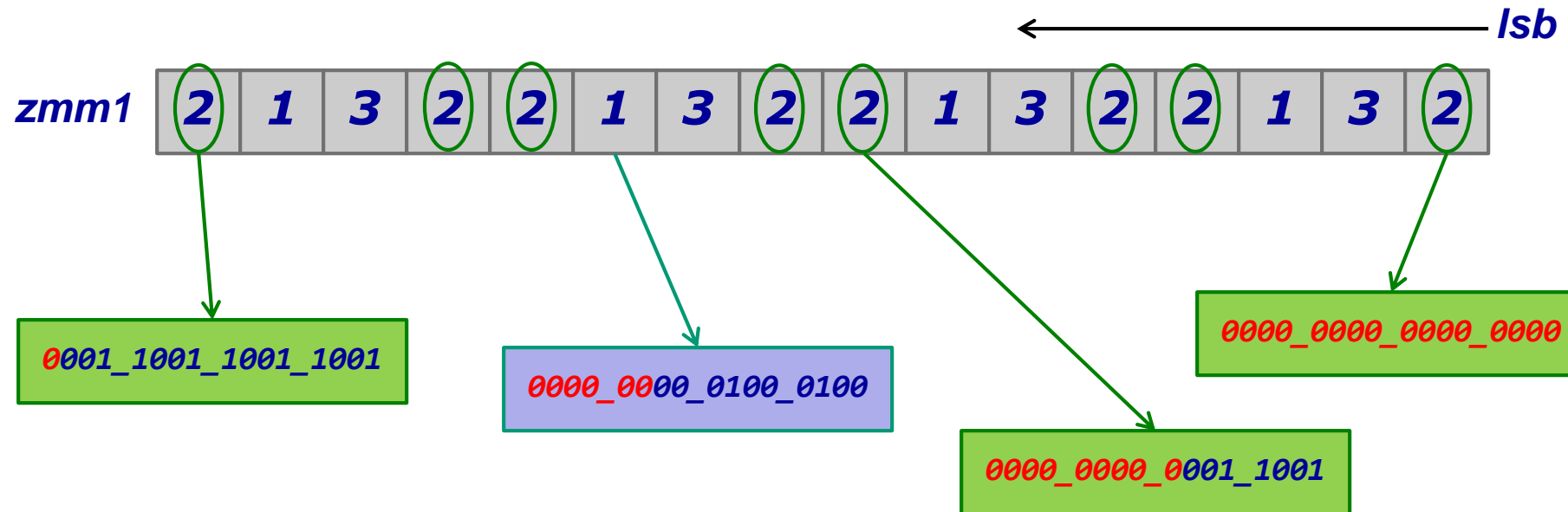
VPTESTNM{D,Q} k2{k1}, zmm2, zmm3/mem

VPLZCNT{D,Q} zmm1 {k1}, zmm2/mem

```
index = vload &B[i] // Load 16 B[i]
pending_elem = 0xFFFF; // all still
remaining
do {
    curr_elem = get_conflict_free_subset(index, pending_elem)
    old_val = vgather {curr_elem} A, index // Grab A[B[i]]
    new_val = vadd old_val, +1.0 // Compute new
    values
    vscatter A {curr_elem}, index, new_val // Update A[B[i]]
    pending_elem = pending_elem ^ curr_elem // remove done idx
} while (pending_elem)
```

# VPCONFLICT{D,Q}

- VPCONFLICT{D,Q} zmm1{k1}{z}, zmm2/B(mV)
  - For every element in ZMM2, compare it against everybody and generate a mask identifying the matches (but ignoring elements to the 'left' of the current one –i.e. “newer”)
    - Store every mask in every element destination in ZMM1



# Optimized Algorithm

```
for each 16 scalar iterations {  
    indices = vload &index_array[i]  
    vpconflictd comparisons, indices  
    vplzcntd tmp_lzcnt, comparisons  
    vpsubd perm_idx, all_31s, tmp_lzcnt  
    temp_values = do_first_iteration(); // gather + compute  
    vptestmd to_do {k0}, comparisons, all_ones // anything left?
```

*Obtain recurrence indices*

```
while (to_do) {  
    vpbroadcastmd tmp, to_do  
    vptestnmd mask {to_do}, comparisons, tmp  
    vpermd tmp_values {mask}, perm_idx  
    tmp_values = do_work(mask); // just compute!  
    to_do ^= mask;  
} while(to_do);  
vscatter indices, A, tmp_values
```

*Store results*

*Re-do conflicting indices reusing results directly from the vector*



# Embedded Broadcasts, Masking Support

- **VFMADD231PS zmm1, zmm2, C {1to16}**
  - Scalars *from memory* are first class citizens
    - Broadcast one scalar from memory into all vector elements before operation
  - Memory fault suppression avoids fetching the scalar if no mask bit is set to 1
- **Other “tuples” supported**
  - Memory only touched if at least one consumer lane needs the data
  - For instance, when broadcast a tuple of 4 elements, the semantics check for every element being really used
    - E.g.: element 1 checks for mask bits 1, 5, 9, 13, ...

```
float32 A[N], B[N], C;
```

```
for(i=0; i<8; i++)  
{  
    if(A[i]!=0.0)  
        A[i] = A[i] + C* B[i];  
}
```

```
VBROADCASTSS zmm1 {k1}, [rax]  
VBROADCASTF64X2 zmm2 {k1}, [rax]  
VBROADCASTF32X4 zmm3 {k1}, [rax]  
VBROADCASTF32X8 zmm4, {k1},  
[rax]
```

# Tools overview

## MPI

- development: Intel MPI
- quick summary: MPS
- drill down, correctness: ITAC

## Threads

- development: OpenMP, TBB, Cilk
- correctness: Inspector
- parallelism: Advisor
- scaling: VTune

## Memory

- analysis: MPS, VTune, Advisor

## Libraries

- math: MKL
- data analytics: DAAL
- media: IPP

## Vectors

- development: C++/Fortran Compilers
- parallelism: Advisor
- analysis: VTune, Advisor
- diagnostics: compiler reports, Advisor

# Legal Disclaimer & Optimization Notice

- **INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.**
- **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.**
- **Copyright © 2015, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.**

## 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

