The Intel® Xeon Phi Coprocessor

Dr-Ing. Michael Klemm
Software and Services Group
Intel Corporation
(michael.klemm@intel.com)

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.

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

Intel Technologies for HPC

Processors

Intel® Xeon® Processor

Coprocessor

Intel® Many Integrated Core

Network & Fabric

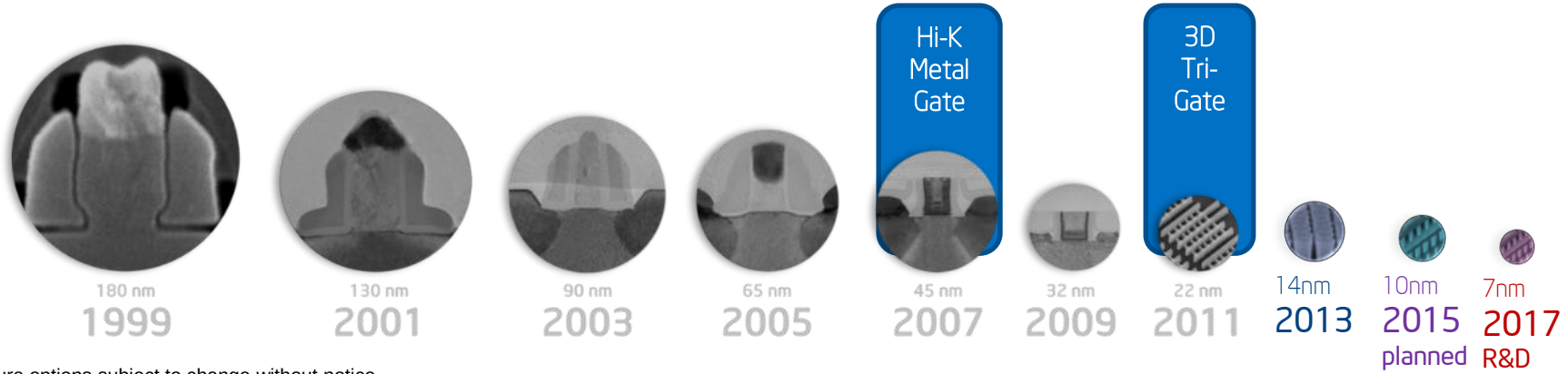
I/O & Storage

Software & Services

Transforming the Economics of HPC

Executing to Moore's Law

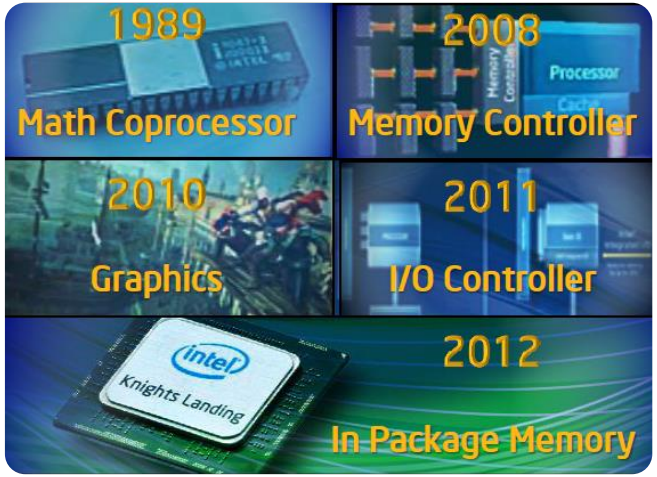
Predictable Silicon Track Record – well and alive at Intel.
Enabling new devices with higher performance and functionality while controlling power, cost, and size



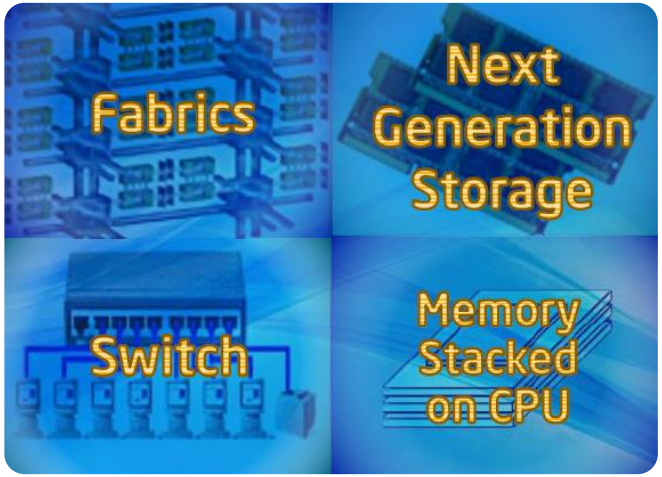
Future options subject to change without notice.

Driving Innovation and Integration

Enabled by Leading Edge Process Technologies



Integrated Today



Coming in the Future

The Magic of Integration

Moore's Law at Work & Architecture Innovations

6666x

1970s

150 MFLOPS

CRAY-1

2013

1000000 MFLOPS

Intel® Xeon Phi™

#1 TOP500 June 2014

33 PFLOPS HPL

54 PFLOPS Peak

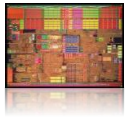
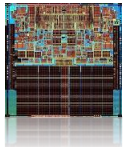
32000 Intel® Xeon® E5v2 Processors

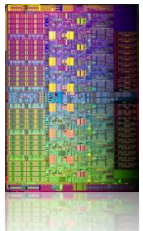
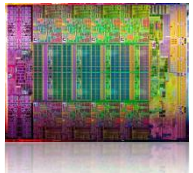
48000 Intel® Xeon Phi™ Coprocessors

Intel® Many Integrated Core Architecture (Intel® MIC) & Intel® Xeon Phi™ Coprocessor

Intel Architecture Multicore and Manycore

More cores. Wider vectors. Co-Processors.

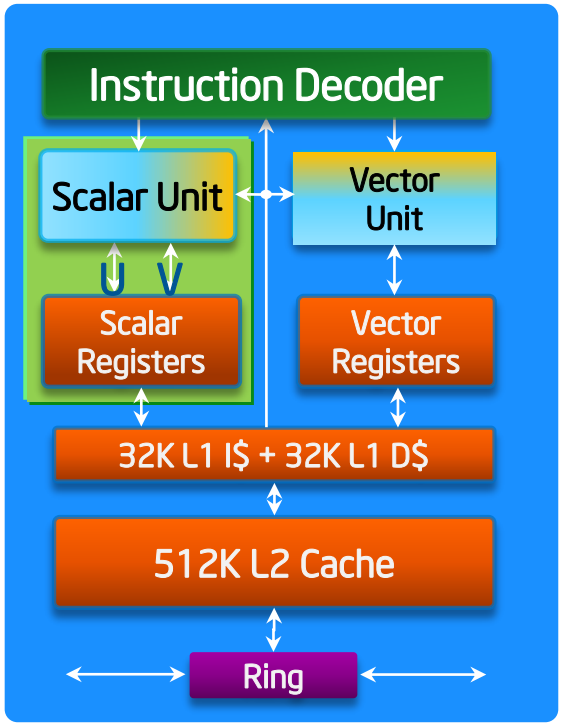



Images do not reflect actual die sizes. Actual production die may differ from images.

	Intel® Xeon® processor 64-bit	Intel Xeon processor 5100 series	Intel Xeon processor 5500 series	Intel Xeon processor 5600 series	Intel Xeon processor E5 Product Family	Intel Xeon processor code name Ivy Bridge	Intel Xeon processor code name Haswell	Intel® Xeon Phi™ Coprocessor code name Knights Corrner
Core(s)	1	2	4	6	8	12	18	61
Threads	2	2	8	12	16	24	36	244

Intel® Xeon Phi™ Coprocessor extends established CPU architecture and programming concepts to highly parallel applications

Each Intel® Xeon Phi™ Coprocessor core is a fully functional multi-thread execution unit



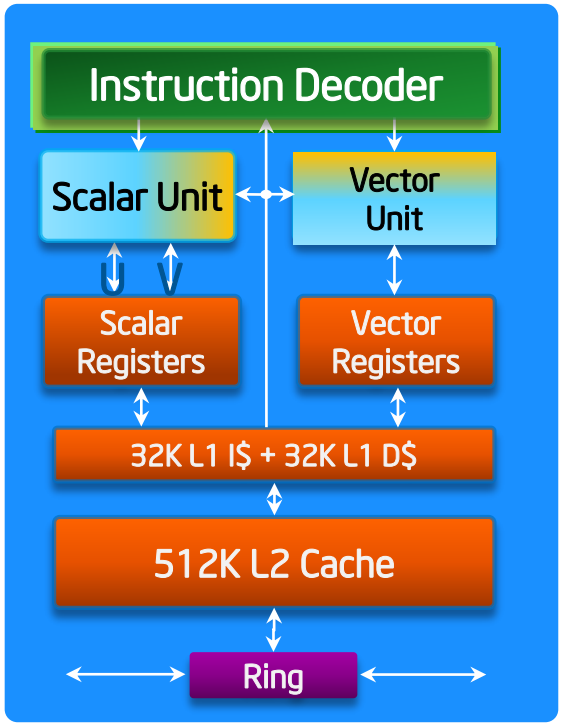
Core unit based on Intel® Pentium® processor family

- Two pipelines (U and V)
  - Dual-issue on scalar instructions
- Scalar pipeline 1 clock latency
- 64-bit data path

4 hardware threads per core

- Thread context: GPRs, ST0-7, etc.
- "Smart" round-robin scheduling
  - Prefetch buffers 2 inst-bundles / context
  - Next ready context selected in order

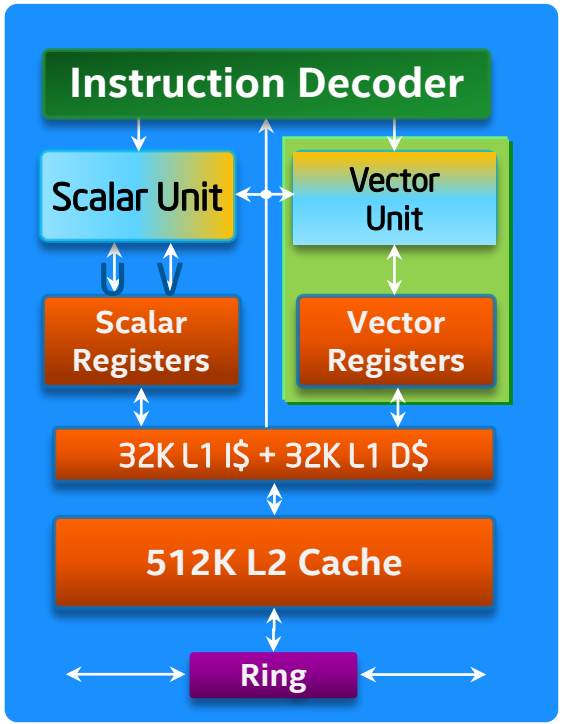
Each Intel® Xeon Phi™ Coprocessor core is a fully functional multi-thread execution unit



Instruction decoder is fully pipelined but is designed as a 2-cycle unit

- Enables significant increase to maximum core frequency, but…
  - Core cannot issue instructions from same context in adjacent cycles
  - Means minimum two threads per core to use all available compute cycles

Each Intel® Xeon Phi™ Coprocessor core is a fully functional multi-thread vector unit

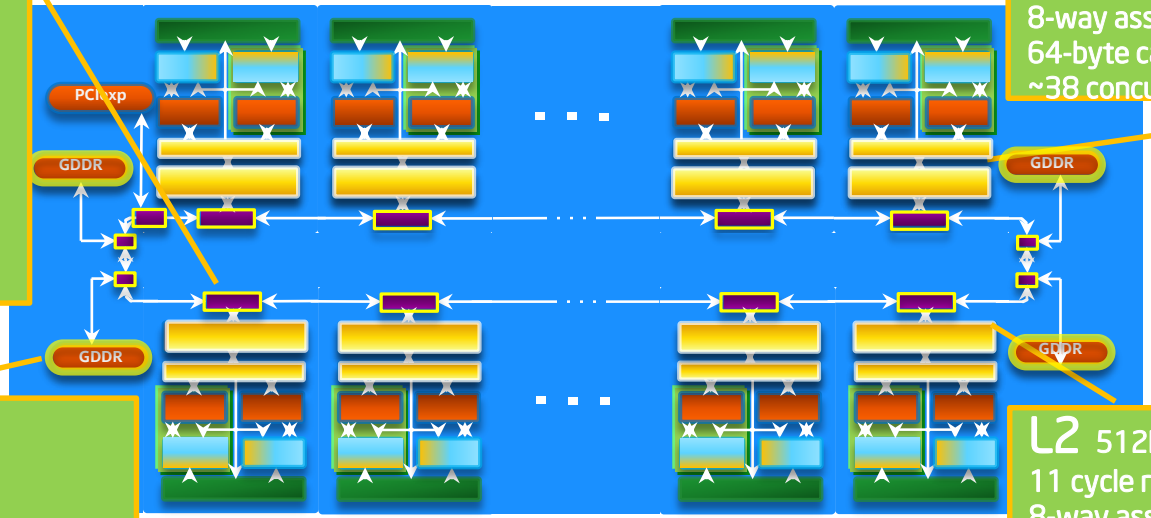


Vector unit width 512 bits!

- 32 512-bit vector registers per context
  - Each holds 16 floats or 8 doubles
  - ALUs support int32/float32 operations, float64 arithmetic, int64 logic ops
  - Ternary ops including Fused-Multiply-Add
  - Broadcast/swizzle support, float16 up-convert
  - 8 vector mask registers for per lane conditional operations
  - Most ops: 4-cycle latency 1-cycle throughput
    - Matches 4-cycle round robin of integer unit
  - Mostly IEEE 754 2008 compliant
    - Not supported: MMX™ technology, Streaming SIMD Extensions (SSE), Intel® Advanced Vector Extensions (Intel® AVX)

Individual cores are tied together via fully coherent caches into a bidirectional ring on the Intel® Xeon Phi™ coprocessor

Bidirectional ring
180 GB/sec
Distributed Tag Directory (DTD) reduces ring snoop traffic
Gen2x16 PCI Express* 64-256 byte packets peer-to-peer R/W



L1 32K I/D-cache per core
1 cycle access latency
3 cycle addr-gen interlock l.
8-way associativity
64-byte cache line
~38 concurrent access/core

GDDR5 Memory
16 32-bit channels
- Up to 5.5 GT/sec
8 GB – 275 ns access latency

L2 512K cache per core
11 cycle raw latency
8-way associativity
64-byte cache line
Streaming HW prefetcher
~38 concurrent access/core

Cache Hierarchy

Parameter	L1	L2
Coherence	MESI	MESI
Size	32KB + 32 KB	512 KB
Associativity	8-way	8-way
Line Size	64 Bytes	64 Bytes
Banks	8	8
Access Time	2 cycle	11 cycle
Policy	Pseudo LRU	Pseudo LRU
Duty Cycle	1 per clock	1 per clock
Ports	Read or Write	Read or Write

There is no L3 cache!

Intel® Xeon Phi™ Product Family

based on Intel® Many Integrated Core (MIC) Architecture

2013:

Intel® Xeon Phi™ Coprocessor x100 Product Family

"Knights Corner"

22 nm process

Up to 61 Cores

Up to 16GB Memory

2015:

Intel® Xeon Phi™ Coprocessor x200 Product Family

"Knights Landing"

14 nm

Processor & Coprocessor

Up to 72 cores

On Package, High-Bandwidth Memory

Future Knights:

Upcoming Gen of the Intel® MIC Architecture

In planning

Continued roadmap commitment

*Per Intel's announced products or planning process for future products

Optimization Notice

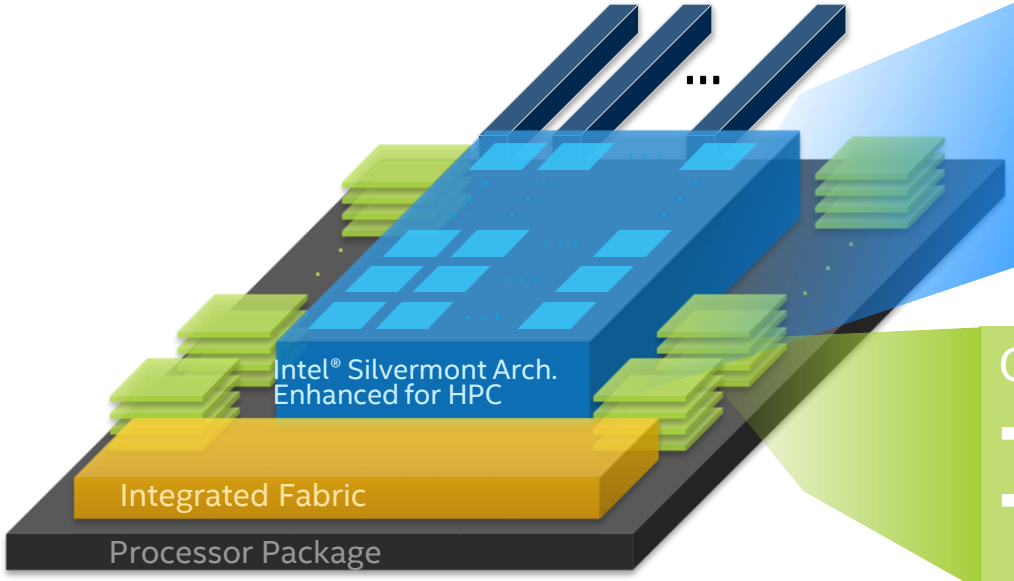
Next Intel® Xeon Phi™ Processor
Codename: Knights Landing

Designed using Intel's cutting-edge
14nm process

Not bound by "offloading" bottlenecks
Standalone CPU
or PCIe Coprocessor

Leadership compute & memory bandwidth
Integrated
On-Package Memory

Next Intel® Xeon Phi™ Processor
Codename: Knights Landing



Compute: Energy-efficient IA cores[2]

- *Microarchitecture enhanced for HPC[3]*
- *3x Single Thread Performance vs Knights Corner[4]*
- *Intel Xeon Processor Binary Compatible[5]*

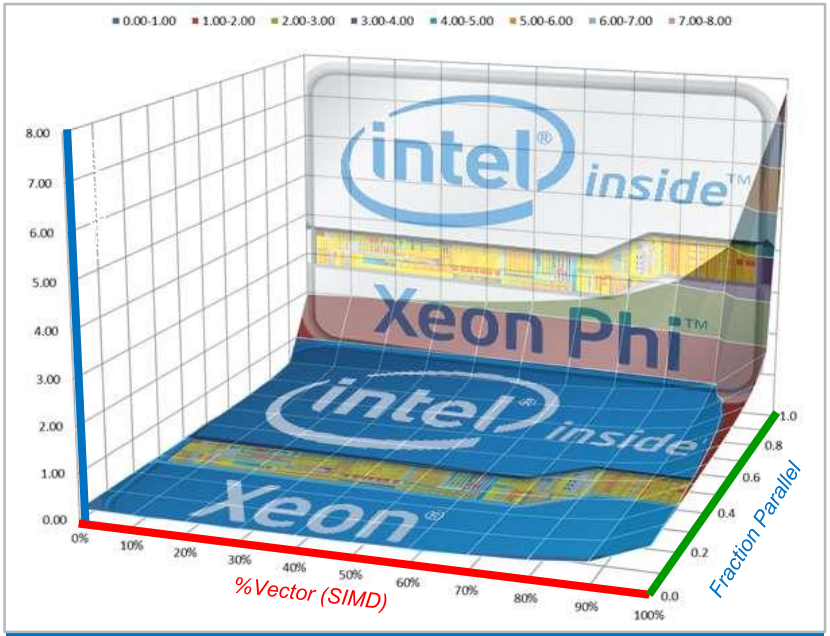
On-Package Memory:

- *up to 16GB at launch*
- *1/3x the Space[6]*
- *5X Bandwidth vs DDR4[7]*
- *5x Power Efficiency[6]*

Jointly Developed with Micron Technology

Programming for Intel Architectures

Highly Parallel Applications



Theoretical acceleration of a highly parallel processor over a Intel® Xeon® parallel processor (<1: Intel® Xeon® faster) – For illustration only

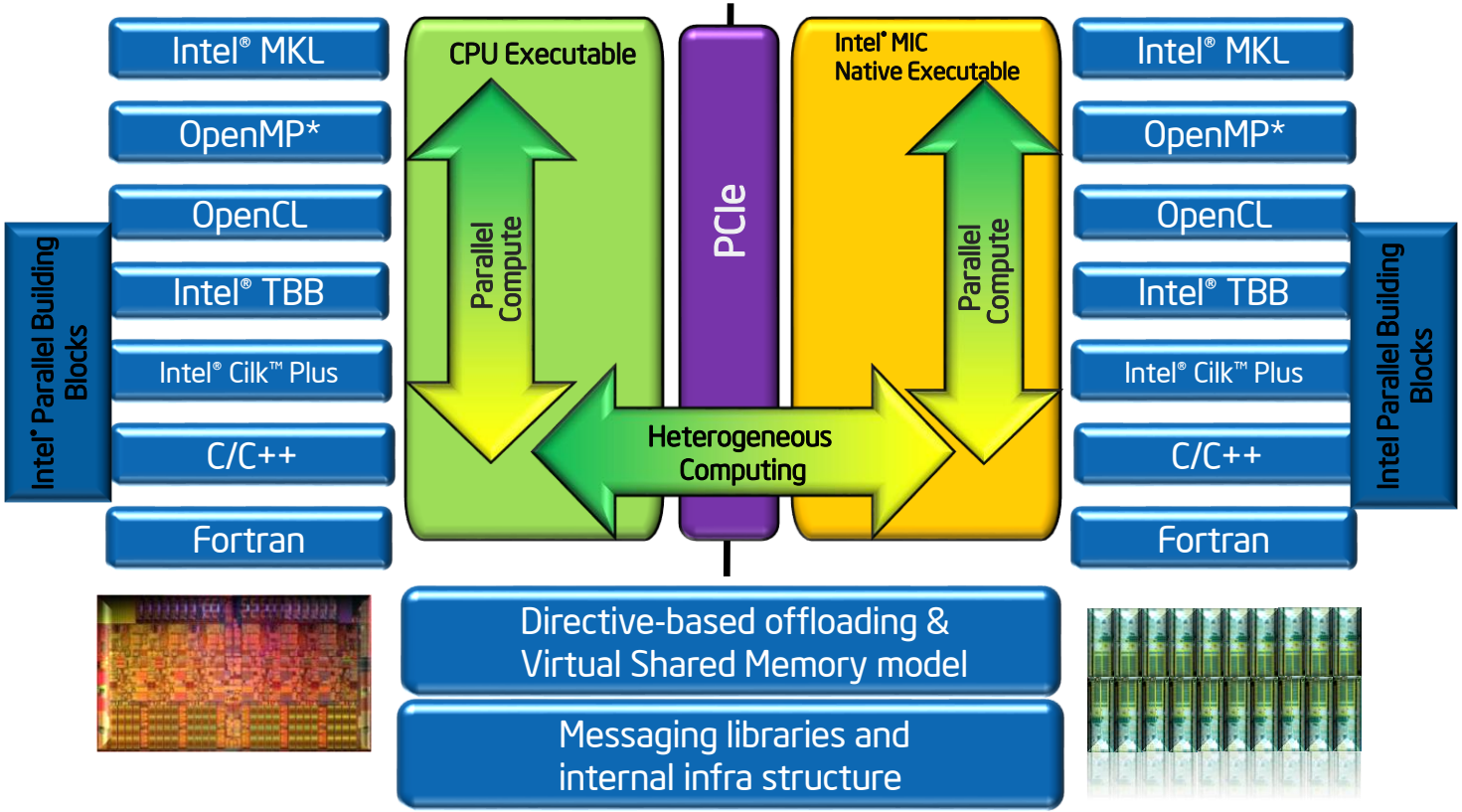
Efficient vectorization, threading, and parallel execution drives higher performance for suitable scalable applications

Parallel Programming for Intel® Architecture

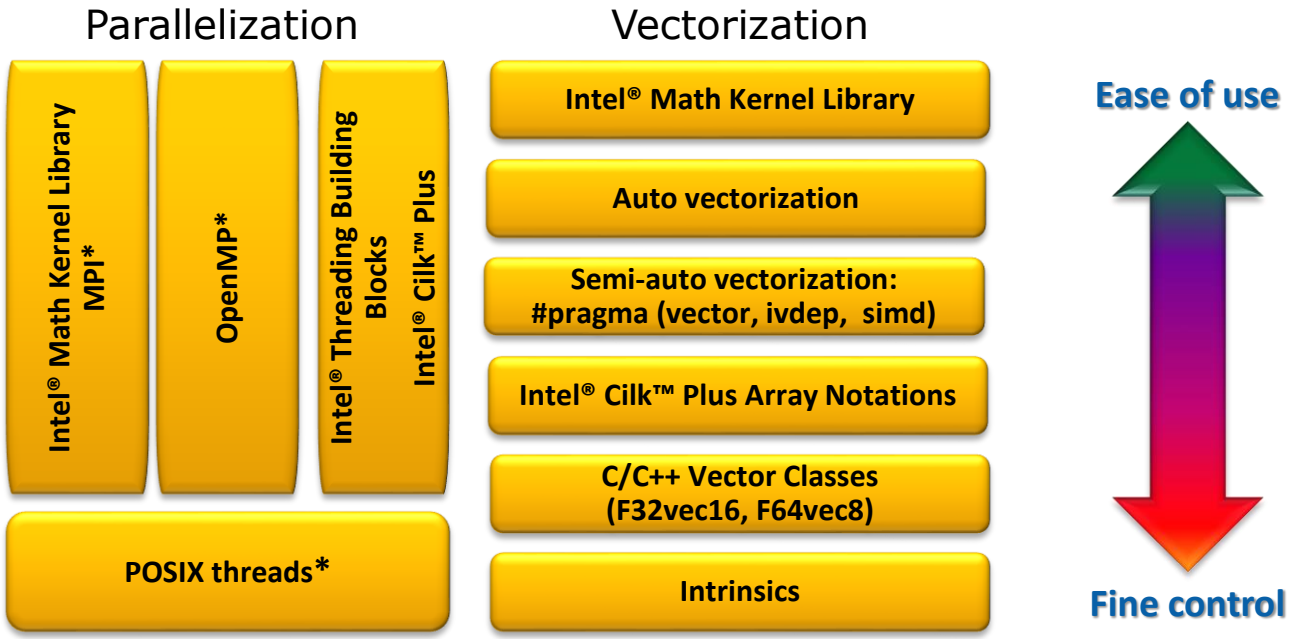
NODES	Use Intel® MPI, Co-Array Fortran
CORES	Use threads directly (pthreads) or via OpenMP*, C++11 Use tasking, Intel® TBB / Cilk™ Plus
VECTORS	Intrinsics, auto-vectorization, vector-libraries Language extensions for vector programming (SIMD)
BLOCKING	Use caches to hide memory latency Organize memory access for data reuse
DATA LAYOUT	Structure of arrays facilitates vector loads / stores, unit stride Align data for vector accesses

Parallel programming to utilize the hardware resources,
in an abstracted and portable way

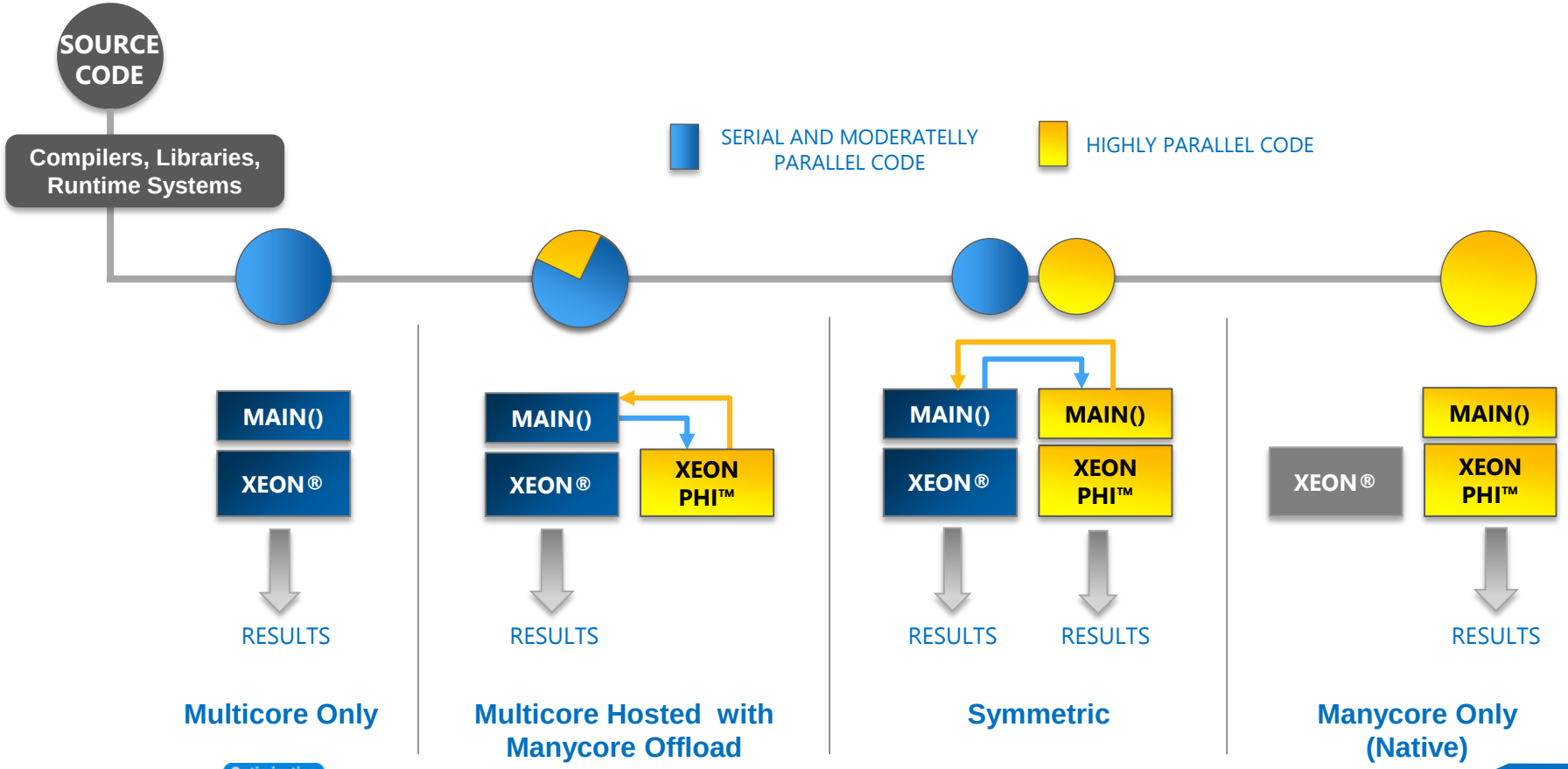
Heterogeneous Programming



Native Programming for Intel Xeon Phi



Flexible Execution Models for Heterogeneous Platforms



Case Study: NWChem CCSD(T)

Finding Offload Candidates

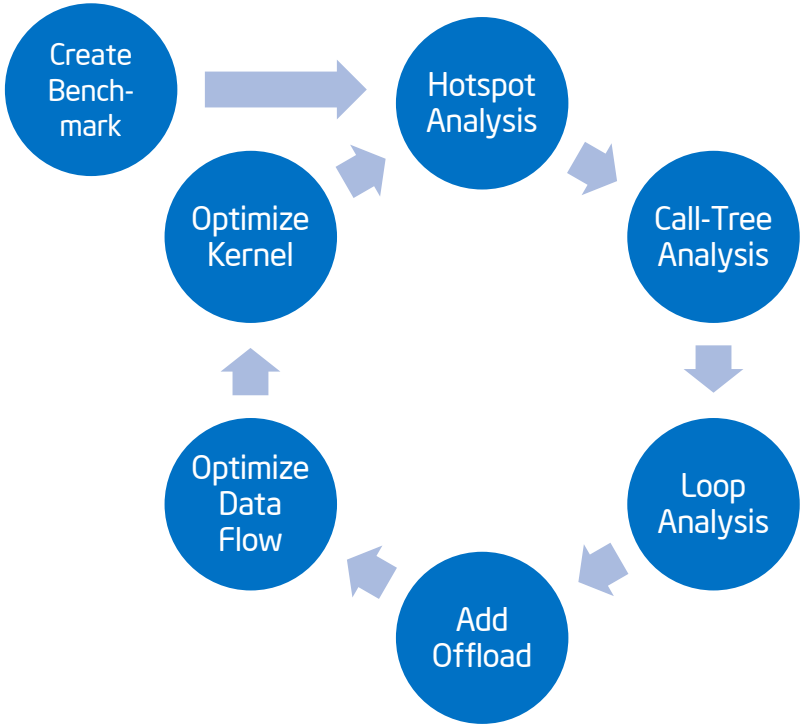
Requirements for offload candidates

- Compute-intensive code regions (kernels)
- Highly parallel
- Compute scaling stronger than data transfer, e.g., compute $O(n^3)$ vs. data size $O(n^2)$

Finding offload candidates

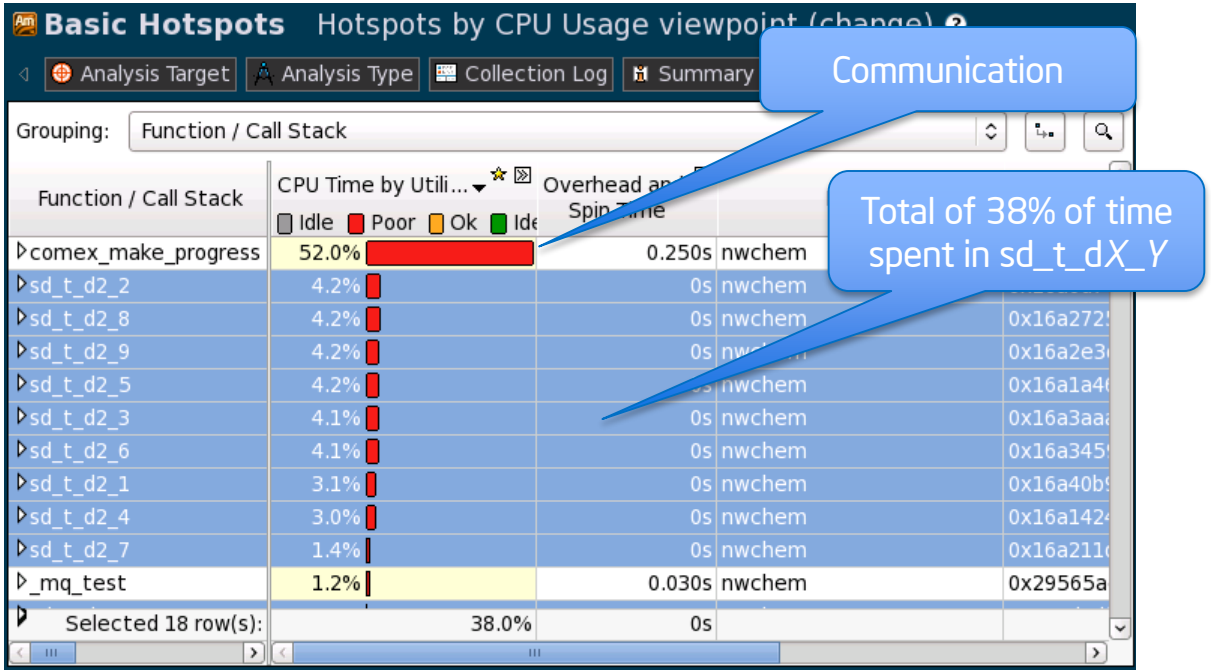
- Create a benchmark to trigger application code of interest
- Find hotspots in the application
- Determine input and output data
- Determine data sizes transferred

Offload Analysis Methodolgy



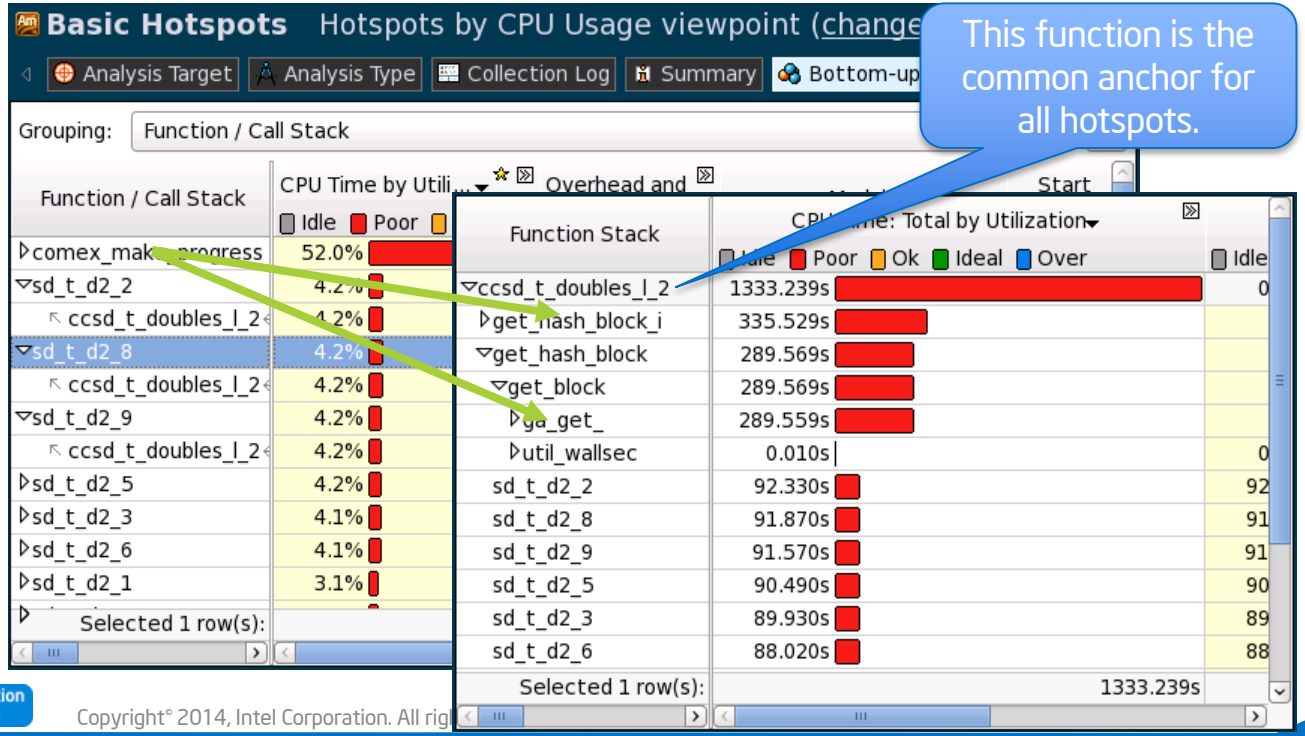
Example: NWChem Hotspots

NWChem hotspot profile



Example: NWChem Hotspots

Call-tree analysis shows relationship of hotspots



Example: Loop Analysis

All kernels expose the same structure

7 perfectly nested loops

Trip count per loop is equal to “tile size” (20-30)

Naïve per-kernel solution is obvious

```
      subroutine sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,
     1                     h7d,triplesx,t2sub,v2sub)
      implicit none                                          offload
      integer h3d,h2d,h1d,p6d,p5d,p4d,h7d
      integer h3,h2,h1,p6,p5,p4,h7
      double precision triplesx(h3d,h2d,h1d,p6d,p5d,p4d)
      double precision t2sub(h7d,p4d,p5d,h1d)
      double precision v2sub(h3d,h2d,p6d,h7d)
      do p4=1,p4d                                            multi-threading
      do p5=1,p5d
      do p6=1,p6d
      do h1=1,h1d
      do h2=1,h2d
      do h3=1,h3d
      do h7=1,h7d
       triplesx(h3,h2,h1,p6,p5,p4)=
     1     triplesx(h3,h2,h1,p6,p5,p4)
     1   - t2sub(h7,p4,p5,h1)*v2sub(h3,h2,p6,h7)           SIMD
      enddo
      enddo
      enddo
      enddo
      enddo
      enddo
      enddo
      end
```

Issues w/ Naïve Offload Solution

Offloading individual kernels requires GBs of data transfers (1-2 GB per offload)

Outer loop does not expose enough parallelism (20-30 threads)

Vectorization potential too low (about 80%)

```
      subroutine sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,
     1                     h7d,triplesx,t2sub,v2sub)
      implicit none
      integer h3d,h2d,h1d,p6d,p5d,p4d,h7d
      integer h3,h2,h1,p6,p5,p4,h7
      double precision triplesx(h3d,h2d,h1d,p6d,p5d,p4d)
      double precision t2sub(h7d,p4d,p5d,h1d)
      double precision v2sub(h3d,h2d,p6d,h7d)
cdir$ offload target(mic) in(t2sub:length(h7d*p4d*p5d*h1d))
     1        in(v2sub:length(h3d*h2d*p6d*h7d))
     2        inout(triplesx:length(h3d*h2d*h1d*p6d*p5d*p4d))
!$omp parallel do
      do p4=1,p4d
      do p5=1,p5d
      do p6=1,p6d
      do h1=1,h1d
      do h2=1,h2d
      do h3=1,h3d
      do h7=1,h7d
        triplesx(h3,h2,h1,p6,p5,p4)=triplesx(h3,h2,h1,p6,p5,p4)
     1    - t2sub(h7,p4,p5,h1)*v2sub(h3,h2,p6,h7)
      enddo
      enddo
      enddo
      enddo
      enddo
      enddo
      enddo
!$omp end parallel do
      end
```

offload

multi-threading

SIMD

Optimization of Data Transfers

Use call-tree analysis to find common anchor for hotspots

- Hoist data transfers up as high as possible
- Make offload regions as large as possible

```
cdir$ offload_transfer target(mic) nocopy(triplesx:length(triplesx_l) ALLOC)
cdir$ offload_transfer target(mic) nocopy(t2sub:length(t2sub_l) ALLOC)
cdir$ offload_transfer target(mic) nocopy(v2sub:length(v2sub_l) ALLOC)
cdir$ offload target(mic) nocopy(triplesx:length(0) REUSE)
      call zero_triplesx(triplesx)
      do ...
        if (...)
cdir$ offload target(mic) in(triplesx:length(0),REUSE)
     1        in(t2sub:length(2sub_l),REUSE)
     3        in(v2sub:length(v2sub),REUSE)
     2        in(h3d,h2d,h1d,p6d,p5d,p4d,h7d)
           call sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,h7,triplesx,t2sub,v2sub)
         endif

c     sd_t_d1_2 until sd_t_d1_9
         enddo
c     Similar structure for sd_t_d2_1 until sd_t_d2_9

cdir$ offload_transfer target(mic) out(triplesx:length(triplesx_l) REUSE)
```

data env.

offload

Kernel Optimizations

Outer loop does not expose enough parallelism (20-30 threads)

Use `collapse` clause to increase parallelism by 1-2 orders of magnitude.

```
      subroutine sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,
     1                     h7d,triplesx,t2sub,v2sub)
      implicit none
      integer h3d,h2d,h1d,p6d,p5d,p4d,h7d
      integer h3,h2,h1,p6,p5,p4,h7
      double precision triplesx(h3d,h2d,h1d,p6d,p5d,p4d)
      double precision t2sub(h7d,p4d,p5d,h1d)
      double precision v2sub(h3d,h2d,p6d,h7d)
!$omp parallel do collapse(3)
      do p4=1,p4d
      do p5=1,p5d
      do p6=1,p6d
      do h1=1,h1d
      do h2=1,h2d
      do h3=1,h3d
      do h7=1,h7d
        triplesx(h3,h2,h1,p6,p5,p4)=triplesx(h3,h2,h1,p6,p5,p4)
     1    - t2sub(h7,p4,p5,h1)*v2sub(h3,h2,p6,h7)
      enddo
      enddo
      enddo
      enddo
      enddo
      enddo
      enddo
!$omp end parallel do
      end
```

multi-threading

Kernel Optimizations, Part 2

- Loop ordering not optimal for SIMD execution
  - Too low trip count for inner loop
- Index analysis shows that loops can be reordered
  - Swap h7 and h3
  - Swap h7 and h2 again

```
      subroutine sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,
     1                     h7d,triplesx,t2sub,v2sub)
      implicit none
      integer h3d,h2d,h1d,p6d,p5d,p4d,h7d
      integer h3,h2,h1,p6,p5,p4,h7
      double precision triplesx(h3d,h2d,h1d,p6d,p5d,p4d)
      double precision t2sub(h7d,p4d,p5d,h1d)
      double precision v2sub(h3d,h2d,p6d,h7d)
!$omp parallel do collapse(3)
      do p4=1,p4d
      do p5=1,p5d
      do p6=1,p6d
      do h1=1,h1d
      do h2=1,h2d
      do h3=1,h3d
      do h7=1,h7d
        triplesx(h3,h2,h1,p6,p5,p4)=triplesx(h3,h2,h1,p6,p5,p4)
     1    - t2sub(h7,p4,p5,h1)*v2sub(h3,h2,p6,h7)
      enddo
      enddo
      enddo
      enddo
      enddo
      enddo
      enddo
!$omp end parallel do
      end
```

multi-threading

Kernel Optimizations, Part 2

- Loop ordering not optimal for SIMD execution
  - Too low trip count for inner loop
- Index analysis shows that loops can be reordered
  - Swap h7 and h3
  - Swap h7 and h2 again
- Loops h2 and h3 can be collapsed

```
      subroutine sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,
     1                     h7d,triplesx,t2sub,v2sub)
      implicit none
      integer h3d,h2d,h1d,p6d,p5d,p4d,h7d
      integer h3,h2,h1,p6,p5,p4,h7
      double precision triplesx(h3d,h2d,h1d,p6d,p5d,p4d)
      double precision t2sub(h7d,p4d,p5d,h1d)
      double precision v2sub(h3d,h2d,p6d,h7d)
!$omp parallel do collapse(3)
      do p4=1,p4d
      do p5=1,p5d
      do p6=1,p6d
      do h1=1,h1d
      do h7=1,h7d
      do h2=1,h2d
      do h3=1,h3d
        triplesx(h3,h2,h1,p6,p5,p4)=triplesx(h3,h2,h1,p6,p5,p4)
     1    - t2sub(h7,p4,p5,h1)*v2sub(h3,h2,p6,h7)
      enddo
      enddo
      enddo
      enddo
      enddo
      enddo
      enddo
!$omp end parallel do
      end
```

multi-threading

Kernel Optimizations, Part 2

- Loop ordering not optimal for SIMD execution
  - Too low trip count for inner loop
- Index analysis shows that loops can be reordered
  - Swap h7 and h3
  - Swap h7 and h2 again
- Loops h2 and h3 can be collapsed

```
      subroutine sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,
     1                     h7d,triplesx,t2sub,v2sub)
      implicit none
      integer h3d,h2d,h1d,p6d,p5d,p4d,h7d
      integer h3,h2,h1,p6,p5,p4,h7
      double precision triplesx(h3d,h2d,h1d,p6d,p5d,p4d)
      double precision t2sub(h7d,p4d,p5d,h1d)
      double precision v2sub(h3d,h2d,p6d,h7d)
!$omp parallel do collapse(3)
      do p4=1,p4d
      do p5=1,p5d
      do p6=1,p6d
      do h1=1,h1d
      do h7=1,h7d
      do h2=1,h2d
      do h3=1,h3d
        triplesx(h3,h2,h1,p6,p5,p4)=triplesx(h3,h2,h1,p6,p5,p4)
     1    - t2sub(h7,p4,p5,h1)*v2sub(h3,h2,p6,h7)
      enddo
      enddo
      enddo
      enddo
      enddo
      enddo
      enddo
!$omp end parallel do
      end
```

multi-threading

Kernel Optimizations, Part 2

- Loop ordering not optimal for SIMD execution
  - Too low trip count for inner loop
- Index analysis shows that loops can be reordered
  - Swap h7 and h3
  - Swap h7 and h2 again
- Loops h2 and h3 can be collapsed

```
      subroutine sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,
     1                     h7d,triplesx,t2sub,v2sub)
      implicit none
      integer h3d,h2d,h1d,p6d,p5d,p4d,h7d
      integer h3,h2,h1,p6,p5,p4,h7
      double precision triplesx(h3d*h2d,h1d,p6d,p5d,p4d)
      double precision t2sub(h7d,p4d,p5d,h1d)
      double precision v2sub(h3d*h2d,p6d,h7d)
!$omp parallel do collapse(3)
      do p4=1,p4d
      do p5=1,p5d
      do p6=1,p6d
      do h1=1,h1d
      do h7=1,h7d
      do h2h3=1,h2d*h3d
        triplesx(h2h3,h1,p6,p5,p4)=triplesx(h2h3,h1,p6,p5,p4)
     1    - t2sub(h7,p4,p5,h1)*v2sub(h2h3,p6,h7)
      enddo
      enddo
      enddo
      enddo
      enddo
      enddo
      enddo
!$omp end parallel do
      end
```

multi-threading

SIMD

about 50% speed-up

Kernel Optimizations, Multi-versioning

```
      subroutine sd_t_d1_1(h3d,h2d,h1d,p6d,p5d,p4d,
     1                     h7d,triplesx,t2sub,v2sub)
      implicit none
      integer h3d,h2d,h1d,p6d,p5d,p4d,h7d
      integer h3,h2,h1,p6,p5,p4,h7
      integer rmndr
      double precision triplesx(h3d*h2d,h1d,p6d,p5d,p4d)
      double precision t2sub(h7d,p4d,p5d,h1d)
      double precision v2sub(h3d*h2d,p6d,h7d)
      rmndr = mod(h3d,8) + mod(h2d,8) + mod(h1d,8) +
     1        mod(p6d,8) + mod(p5d,8) + mod(p4d,8) +
     2        mod(h7d,8)
      if (rmndr.eq.0) then
!$omp parallel do collapse(3)
        do p4=1,p4d
        do p5=1,p5d
        do p6=1,p6d
        do h1=1,h1d
        do h7=1,h7d
!dec$ vector aligned
        do h2h3=1,h2d*h3d
          triplesx(h2h3,h1,p6,p5,p4)=triplesx(h2h3,h1,p6,p5,p4)
     1    - t2sub(h7,p4,p5,h1)*v2sub(h2h3,p6,h7)
        enddo
        enddo
        enddo
        enddo
        enddo
        enddo
!$omp end parallel do
c     continued from left column
      else
!$omp parallel do collapse(3)
        do p4=1,p4d
        do p5=1,p5d
        do p6=1,p6d
        do h1=1,h1d
        do h7=1,h7d
        do h2h3=1,h2d*h3d
          triplesx(h2h3,h1,p6,p5,p4)=triplesx(h2h3,h1,p6,p5,p4)
     1     - t2sub(h7,p4,p5,h1)*v2sub(h2h3,p6,h7)
        enddo
        enddo
        enddo
        enddo
        enddo
        enddo
        enddo
!$omp end parallel do
      endif
      end
```

about 15% speed-up

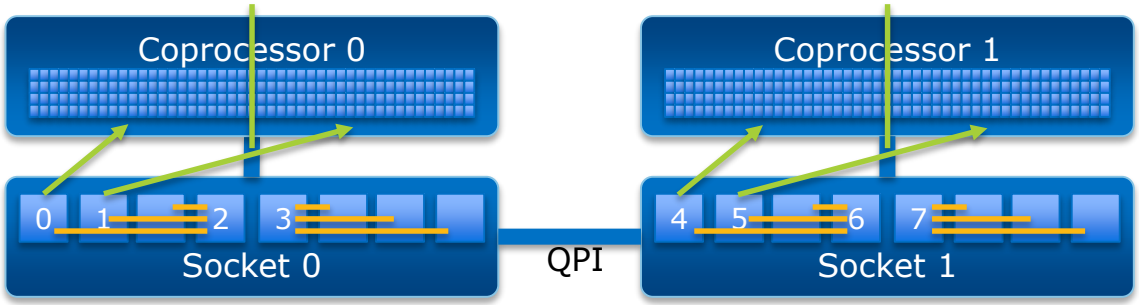
Device Partitioning

Host executes several MPI ranks

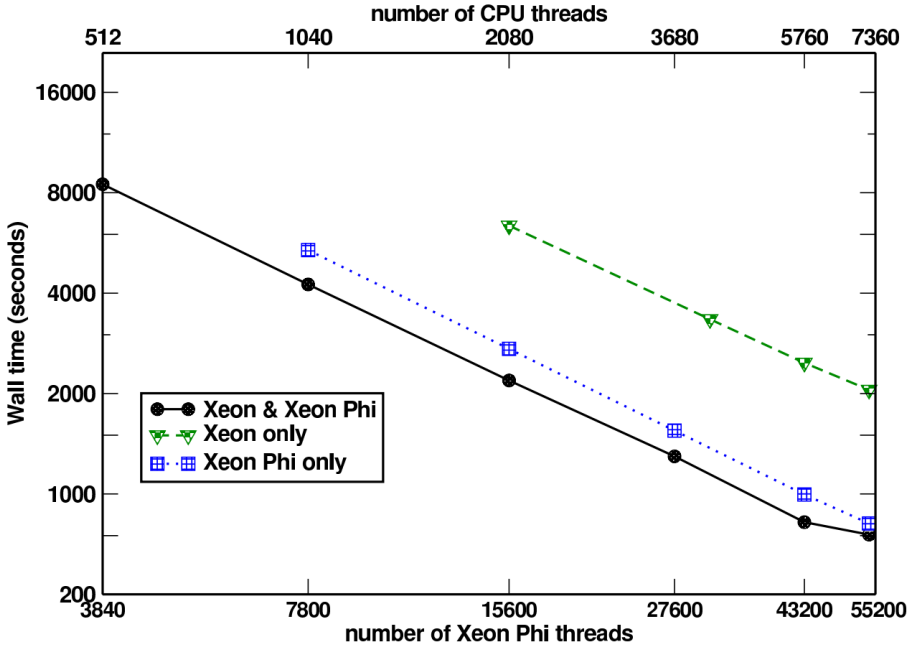
- Utilize coprocessor from several host processes concurrently
- Utilize host CPUs for increased performance

Partition coprocessors through OpenMP* runtime

- Less threading overhead, better overall system utilization
- Rank 0: OFFLOAD_DEVICES=0 KMP_PLACE_THREADS=30c,4t,0o
- Rank 1: OFFLOAD_DEVICES=0 KMP_PLACE_THREADS=30c,4t,30o
- Rank 4: OFFLOAD_DEVICES=1 KMP_PLACE_THREADS=30c,4t,0o
- Rank 5: OFFLOAD_DEVICES=1 KMP_PLACE_THREADS=30c,4t,30o



Performance Results



Performance tests 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. System configuration: Atipa Visione vf442 server with two Intel Xeon E5-2670 8-core processors at 2.6 GHz (128 GB DDR3 with 1333 MHz, Scientific Linux release 6.5) and Intel C600 IOH, two Intel Xeon Phi coprocessors 5110P (GDDR5 with 3.6 GT/sec, driver v3.1.2-1, flash image/micro OS 2.1.02.0390, Intel Composer XE 14.0.1.106). Benchmark perturbative triples correction to the CCSD(T) correlation energy of the 1,3,4,5-tetrasilylimidazol-2-ylidene molecule (formula $Si_4C_3N_2H_{12}$) in its triplet state.