BAYNCORE

A closer look at Intel Xeon and Xeon Phi (KNL) for HPC developers

Janko Strassburg

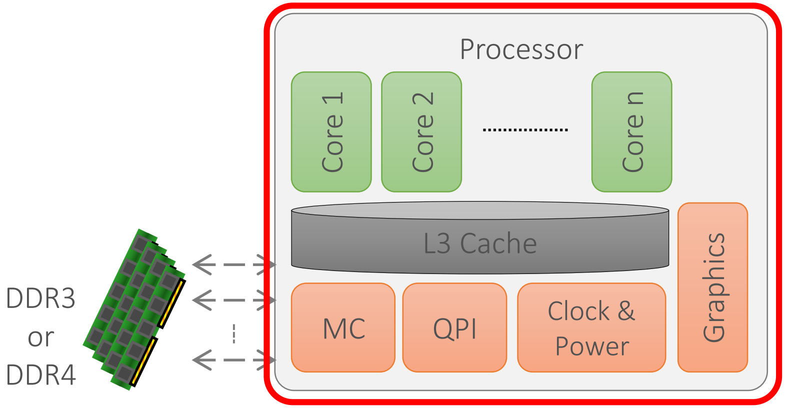
HPC Knowledge Meeting'16 - HPCKP

April 20th 2016, Barcelona

Today’s Intel solutions for HPC

The multi- and many-core era

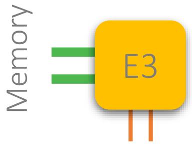
Multi-core	Many integrated core (MIC)
C/C++/Fortran, OMP/MPI/Cilk+/TBB	C/C++/Fortran, OMP/MPI/Cilk+/TBB
Bootable, native execution model	PCIe coprocessor, native and offload execution models
Up to 18 cores, 3 GHz, 36 threads ... until slide 16	Up to 61 cores, 1.2 GHz, 244 threads
Up to 768 GB, 68 GB/s, 432 GFLOP/s DP	Up to 16 GB, 352 GB/s, 1.2 TFLOP/s DP
256-bit SIMD, FMA, gather (AVX2)	512-bit SIMD, FMA, gather/scatter, EMU (IMCI)
Targeted at general purpose applications Single thread performance (ILP) Memory capacity	**Targeted at highly parallel applications** High parallelism (DLP, TLP) High memory bandwidth



Intel® Xeon® processor architecture

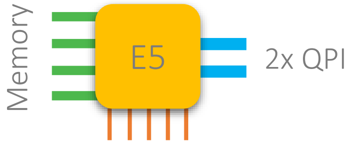
Intel® Xeon® processors and platforms

Intel® Xeon® E3



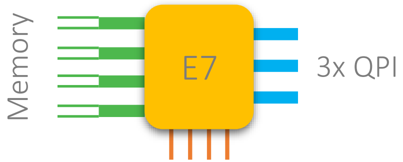
PCIe3

Intel® Xeon® E5



PCIe3

Intel® Xeon® E7

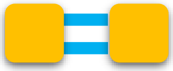


PCIe3

Intel® Xeon®
E5-1xxx
E3-1xxx

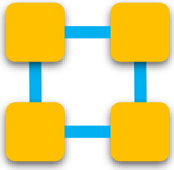
CPU/Socket

Intel® Xeon®
E5-2xxx

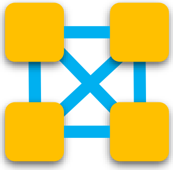


QPI

Intel® Xeon®
E5-4xxx



Intel® Xeon®
E7-4xxx

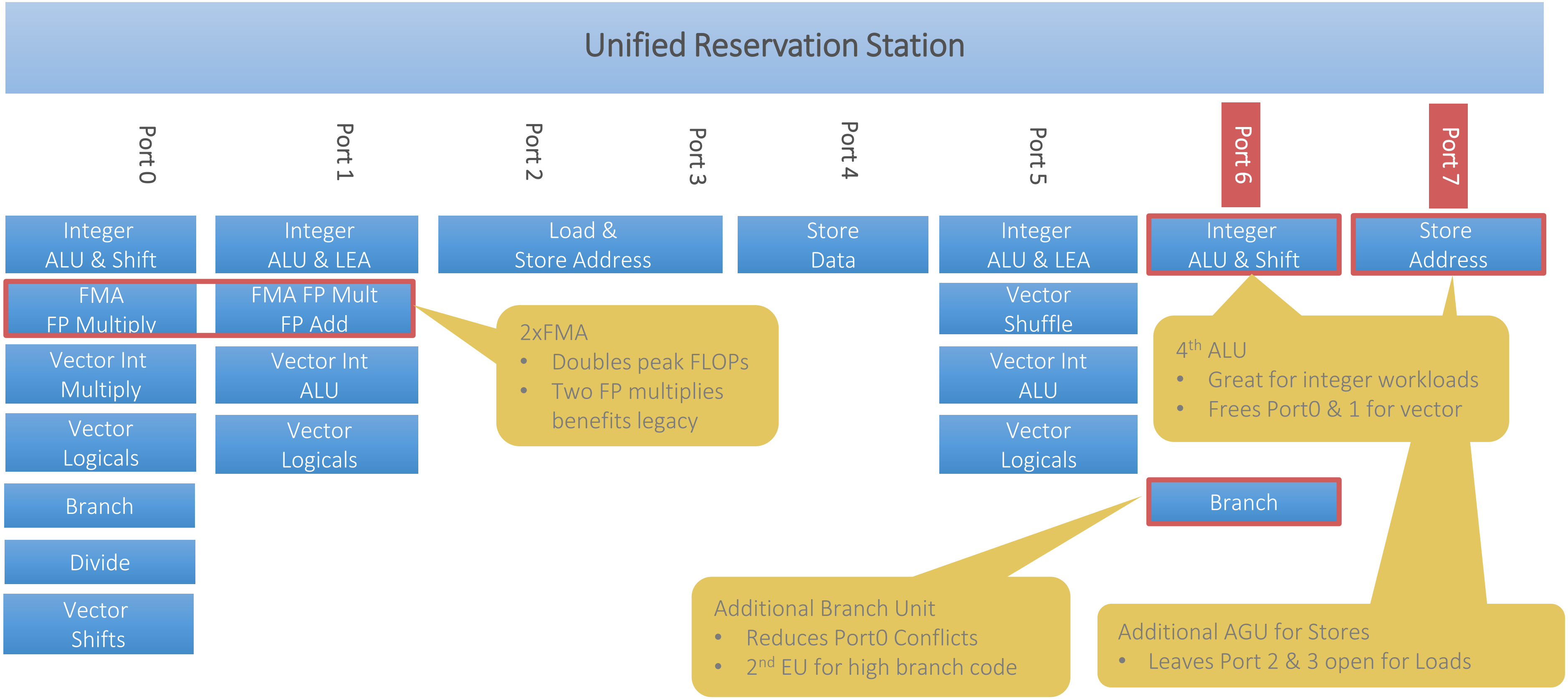


Intel® Xeon®
E7-xxxx

>4S

E5-2600 v3 (Haswell EP)
E5-2600 v4 (Broadwell EP)

Haswell execution unit overview



2xFMA

- Doubles peak FLOPs
- Two FP multiplies benefits legacy

4th ALU

- Great for integer workloads
- Frees Port0 & 1 for vector

Additional Branch Unit

- Reduces Port0 Conflicts
- 2nd EU for high branch code

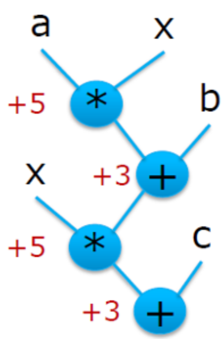
Additional AGU for Stores

- Leaves Port 2 & 3 open for Loads

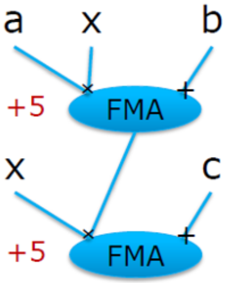
Fused Multiply and Add (FMA) instruction

Example: polynomial evaluation

$$ax^2 + bx + c$$
$$=$$
$$x(ax + b) + c$$



16 cycle latency
2 cycle throughput



10 cycle latency
1 cycle throughput

Micro-Architecture	Instruction Set	SP FLOPs per cycle	DP FLOPs per cycle
Nehalem	SSE (128-bits)	8	4
Sandy Bridge	AVX (256-bits)	16	8
Haswell	AVX2 (FMA) (256-bits)	32	16

2x peak FLOPs/cycle *(throughput)*

Latency (clocks)	Xeon E5 v2	Xeon E5 v3	Ratio (lower is better)
MulPS, PD	5	5	
AddPS, PD	3	3	
Mul+Add /FMA	8	5	0.625

>37% reduced latency
(5-cycle FMA latency same as an FP multiply)

Improves accuracy and performance for commonly used class of algorithms

Broadwell: 5th generation Intel® Core™ architecture

Microarchitecture changes

FP instructions performance improvements

- Decreased latency and increased throughput for most divider (radix-1024) uops
- Pseudo-double bandwidth for scalar divider uops
- Vector multiply latency decrease (from 5 to 3 cycles)

STLB improvements

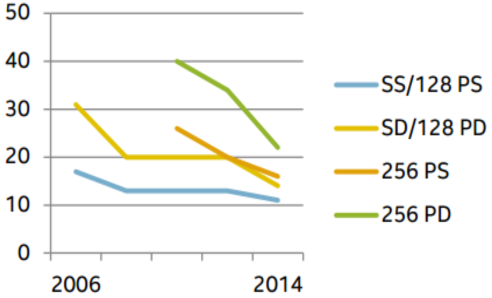
- Native, 16-entry 1G STLB array
- Increased size of STLB (from 1kB to 1.5kB)

Enabled two simultaneous page miss walks

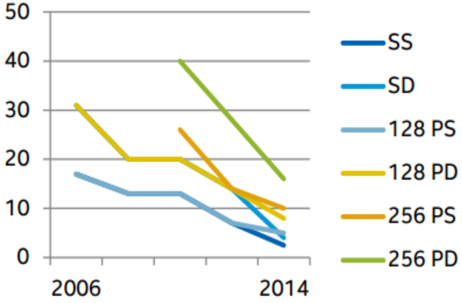
Other ISA performance improvements

- ADC, CMOV – 1 uop flow
- PCLMULQDQ – 2 uop/7 cycles to 1 uop/5 cycles
- VCVTPS2PH (mem form) – 4 uops to 3 uops

Divide Latency (cycles)



Divide Throughput (cycles to start next)



Skylake: 6th generation Intel® Core™ architecture

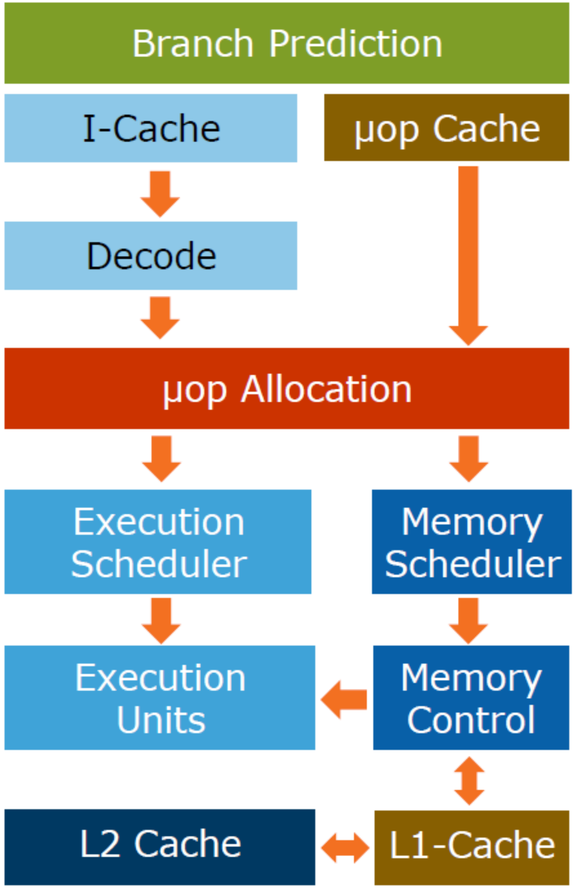
Dedicated server and client IP configurations

Improved microarchitecture

- Higher capacity front-end (up to 6 instr/cycle)
- Improved branch predictor
- Deeper Out-of-Order buffers
- More execution units, shorter latencies
- Deeper store, fill, and write-back buffers
- Smarter prefetchers
- Improved page miss handling
- Better L2 cache miss bandwidth
- Improved Hyper-Threading
- Performance/watt enhancements

New instructions supported

- Software Guard Extensions (SGX)
- Memory Protection Extensions (MPX)
- AVX-512 (Xeon versions only)



Intel® Xeon® Processor E5-2600 v4 Product Family Overview

New Features:

- Broadwell microarchitecture
- Built on 14nm process technology
- Socket compatible◊ replacement/ upgrade on Grantley-EP platforms

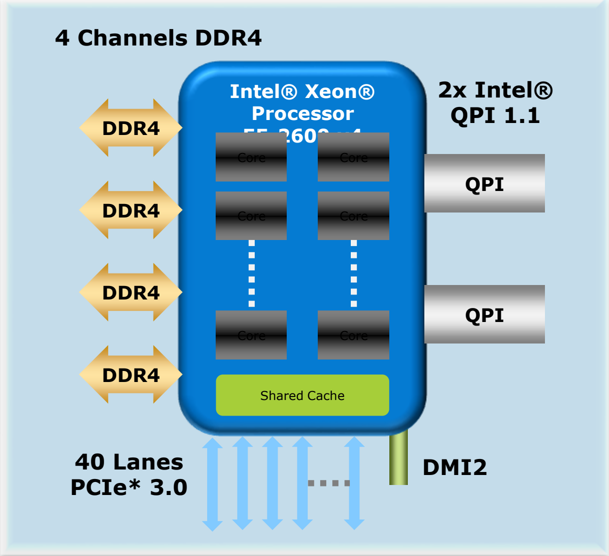
New Performance Technologies:

- Optimized Intel® AVX Turbo mode
- Intel TSX instructions^

Other Enhancements:

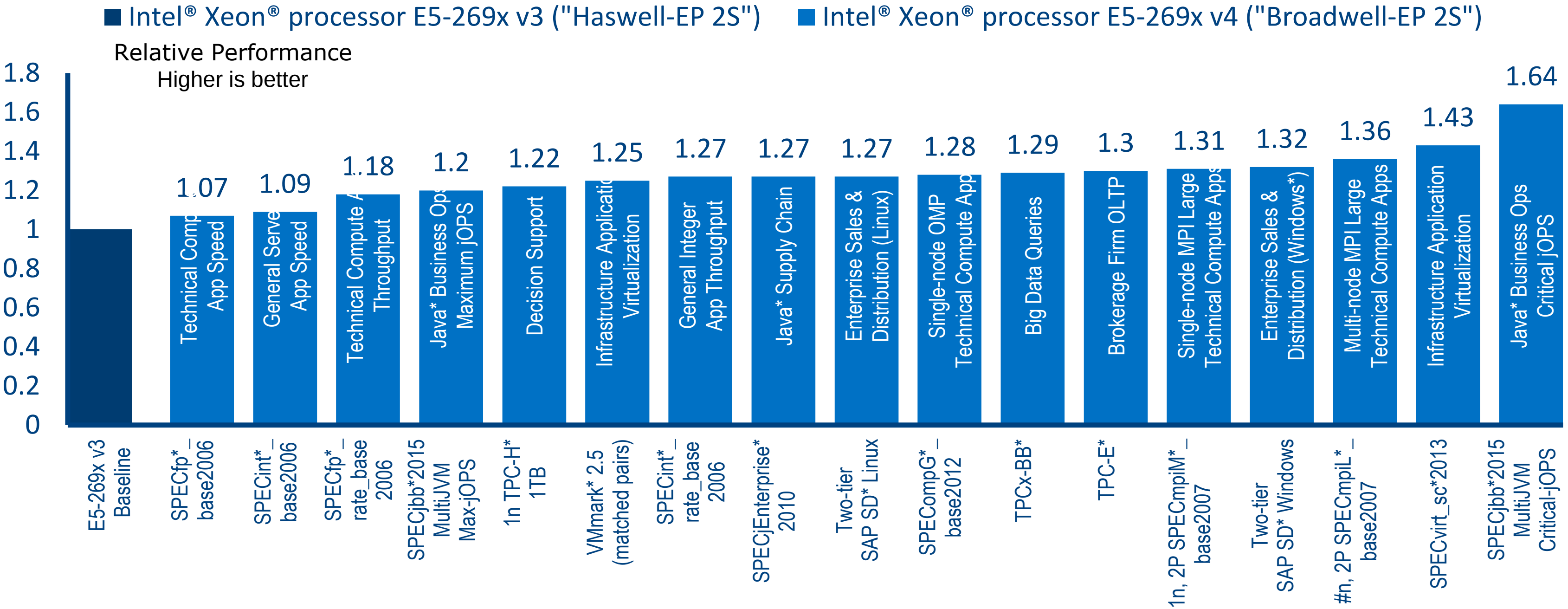
- Virtualization speedup
- Orchestration control
- Security improvements

Features	Xeon E5-2600 v3 (Haswell-EP)	Xeon E5-2600 v4 (Broadwell-EP)
Cores Per Socket	Up to 18	Up to 22
Threads Per Socket	Up to 36 threads	Up to 44 threads
Last-level Cache (LLC)	Up to 45 MB	Up to 55 MB
QPI Speed (GT/s)	2x QPI 1.1 channels 6.4, 8.0, 9.6 GT/s	
PCIe* Lanes / Speed(GT/s)	40 / 10 / PCIe* 3.0 (2.5, 5, 8 GT/s)	
Memory Population	4 channels of up to 3 RDIMMs or 3 LRDIMMs	+ 3DS LRDIMM†
Memory RAS	ECC, Patrol Scrubbing, Demand Scrubbing, Sparing, Mirroring, Lockstep Mode, x4/x8 SDDC	+ DDR4 Write CRC
Max Memory Speed	Up to 2133	Up to 2400
TDP (W)	160 (Workstation only), 145, 135, 120, 105, 90, 85, 65, 55	



◊ Requires BIOS and firmware update; ^ not available broadly on E5-2600 v3; † Depends on market availability

Up to 1.27x Average Generational Gains on Servers using Intel® Xeon® Processor E5-2600 v4 Product Family

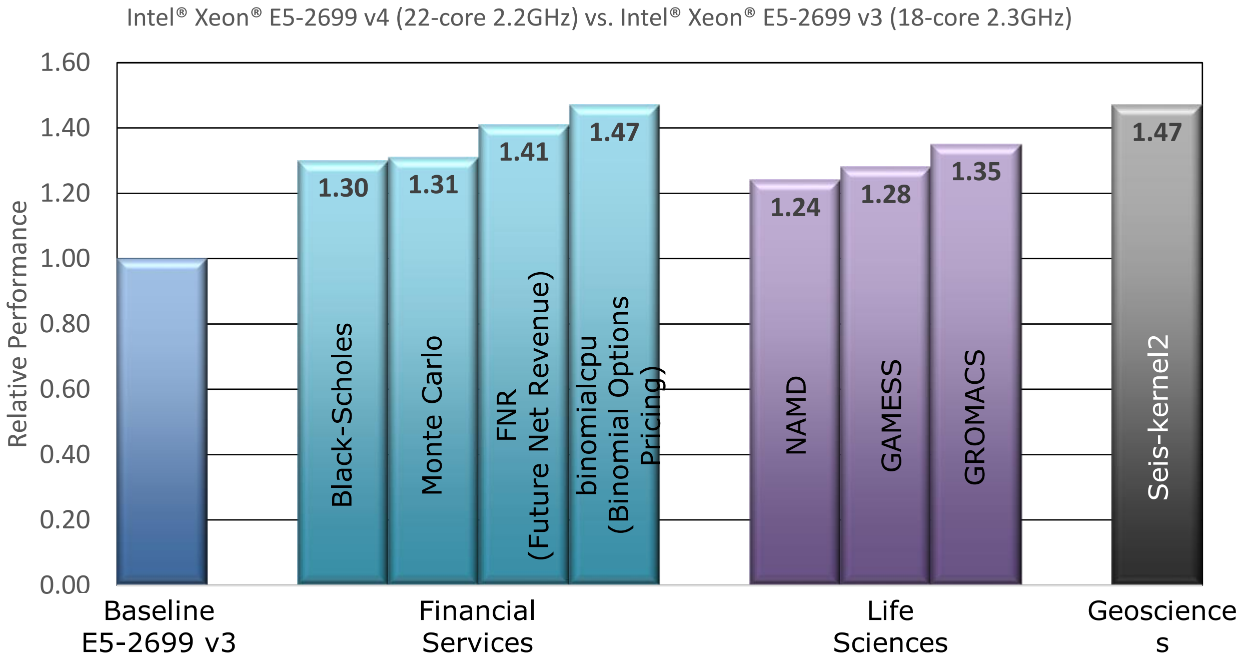


Normalized Generational Performance Summary (based on published industry benchmark results)

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. Configurations: see next page. For more information go to http://www.intel.com/performance/datacenter.

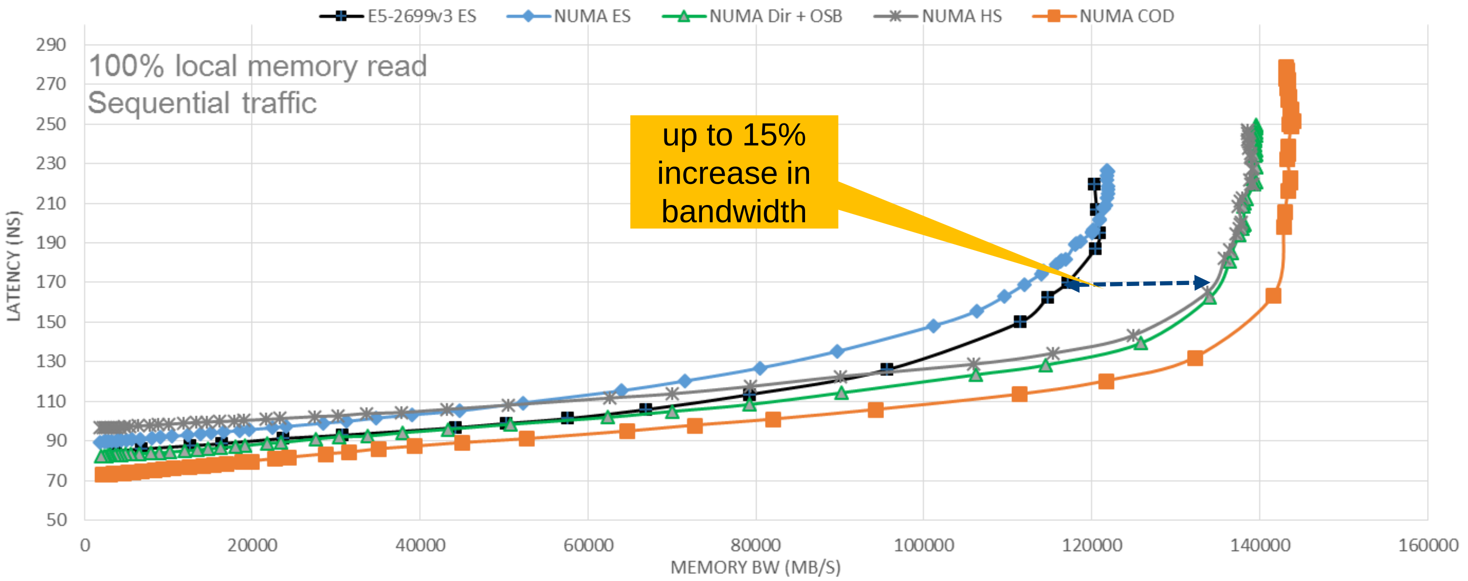
High Performance Computing Performance

Compute



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 based on Intel® internal measurements as of February 29, 2016. Configurations: see slide 10

Home Snoop w/DIR+OSB Provides up to 15% more Bandwidth vs Early Snoop on E5-26xx v3

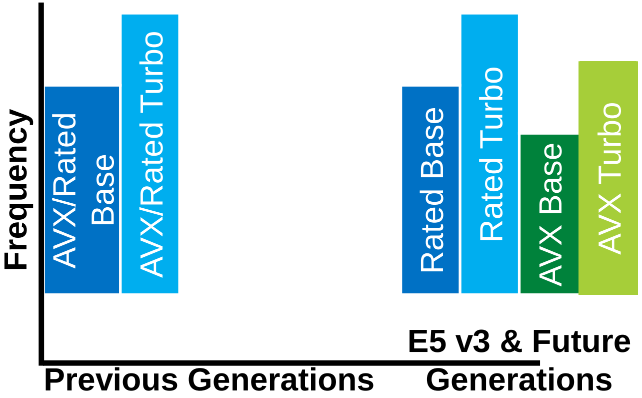


Memory Read Latency and Bandwidth

Source as of 21 July 2015: Intel internal measurements on platform with two E5-26xx v4 (22C, CLR:2.8GHz), Turbo enabled, 4x32GB 1DPC DDR4-2400, RHEL 7.0. Platform with two E5-2699 v3, Turbo enabled, 4x32GB DDR4-2133, RHEL 7.0. 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 *Other names and brands may be claimed as the property of others.

Intel® Turbo Boost Technology 2.0 and Intel® AVX*

- Amount of turbo frequency achieved depends on:
  - Type of workload, number of active cores, estimated current & power consumption, and processor temperature
- Due to workload dependency, separate AVX base & turbo frequencies will be defined for Intel® Xeon® processors starting with E5 v3 product family



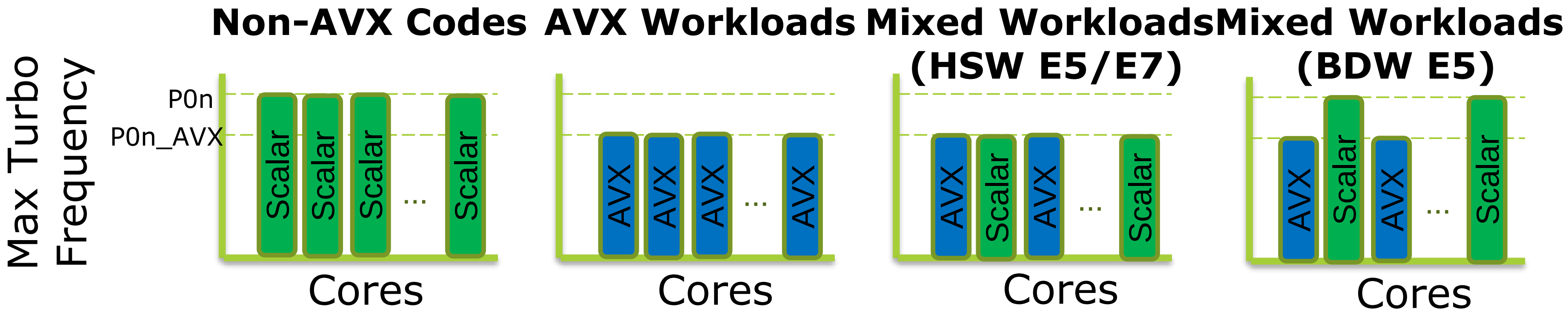
Additional Resources:

- Whitepaper - Optimize Performance with Intel AVX
- Intel® Xeon® Turbo Boost Opportunistic Frequency Upside
- Using Intel AVX to Achieve Maximum Performance on Intel Xeon Processors

*Intel® AVX refers to Intel® AVX, Intel® AVX2 or Intel® AVX-512

Per-Core AVX Max Turbo Optimization

on Intel® Xeon® processor E5-2600 v4 Product Family



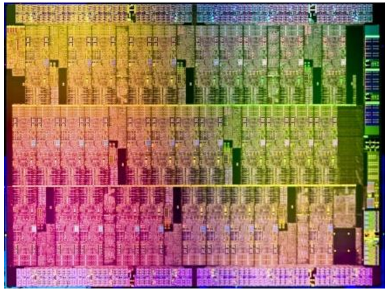
Non-AVX workloads running on all cores have a max turbo frequency of P0n.

AVX workloads have a lower max turbo frequency of “P0n AVX”

On E5-2600 v3 family, workloads with a mix of cores running AVX and non-AVX experienced lower max turbo frequency on all

On E5-2600 v4 family, cores running AVX do not automatically decrease the max turbo frequency of other cores in the socket

New manufacturing and algorithmic techniques providing higher potential turbo frequencies for improved performance in systems with heterogeneous workloads



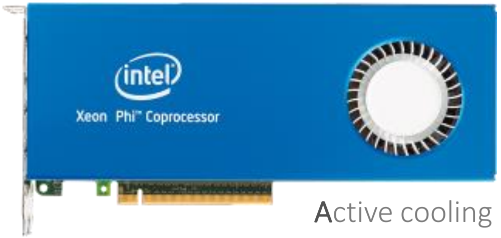
Intel® Xeon Phi™ (co)processor architecture

Intel® Many Integrated Core architecture (Intel® MIC)

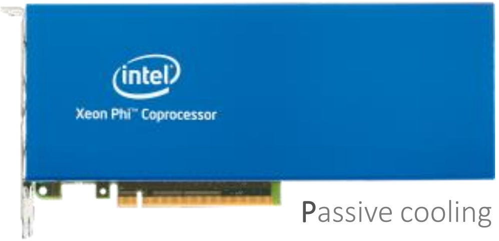
Intel® Xeon Phi™ architecture family

Intel® Xeon Phi^{TM} coprocessor product family **“Knights Corner”**	Intel® Xeon Phi^{TM} coprocessor product family **“Knights Landing”**	Upcoming generation of the Intel® MIC architecture **“Knights Hill”**
2013	2H’2015	2017?
22 nm process	14 nm process	10 nm process
1 TeraFLOP DP peak	3+ TeraFLOP DP peak	?
57-61 cores In-order core architecture 1 Vector Unit per core	72 cores (36 tiles) Out-of-order architecture based on Intel® Atom^{TM} core 2 Vector Units per core Up to 3x single thread performance w.r.t. Knights Corner	?
6-16 GB GDDR5 memory	On package, 8-16 GB high bandwidth memory (HBM) with flexible models: cache, flat, hybrid Up to 768 GB DDR4 main memory	?
Intel® Initial Many Core Instructions (IMIC)	Intel® Advanced Vector Extensions (AVX-512) Binary compatible with AVX2	?
PCIe coprocessor	Stand alone processor and PCIe coprocessor versions	?
Intel® True Scale fabric	Intel® Omni-Path™ fabric (integrated in some models)	2nd generation Intel® Omni-Path™ fabric

Intel® Xeon Phi^{TM} coprocessor product lineup



Active cooling



Passive cooling

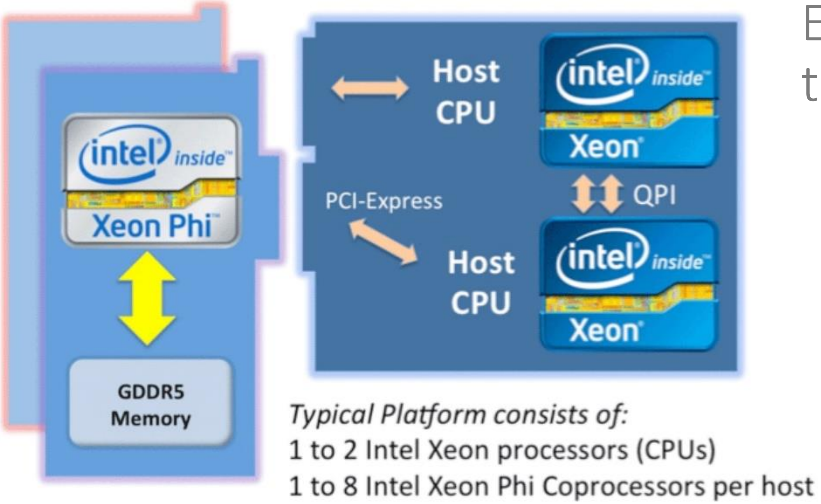
Dense form factor

No thermal solution (X)

Family	Specifications	Product name
7 Family Highest performance, more memory Performance leadership	61 cores 16GB GDDR5 352 GB/s > 1.2TF DP 270-300W TDP	7120P (Q2'13)+ 7120X (Q2'13)+ 7120D (Q1'14)+ 7120A (Q2'14)+
5 Family Optimized for high density environments Performance/watt leadership	60 cores 8GB GDDR5 320-352 GB/s > 1TF DP 225-245W TDP	5110P (Q4'12) 5120D (Q2'13)
3 Family Outstanding parallel computing solution Performance/$ leadership	57 cores 6-8GB GDDR5 240-320 GB/s > 1TF DP 270-300W TDP	3120A (Q2'13) 3120P (Q2'13) 31S1P (Q2'13)*

(+) Special offer with a free 12-month trial of Intel® Parallel Studio XE Cluster Edition - until September 30, 2016

Intel® Xeon Phi^{TM} platform architecture



Typical Platform consists of:
1 to 2 Intel Xeon processors (CPUs)
1 to 8 Intel Xeon Phi Coprocessors per host

(c) 2013 Jim Jeffers and James Reinders, used with permission.

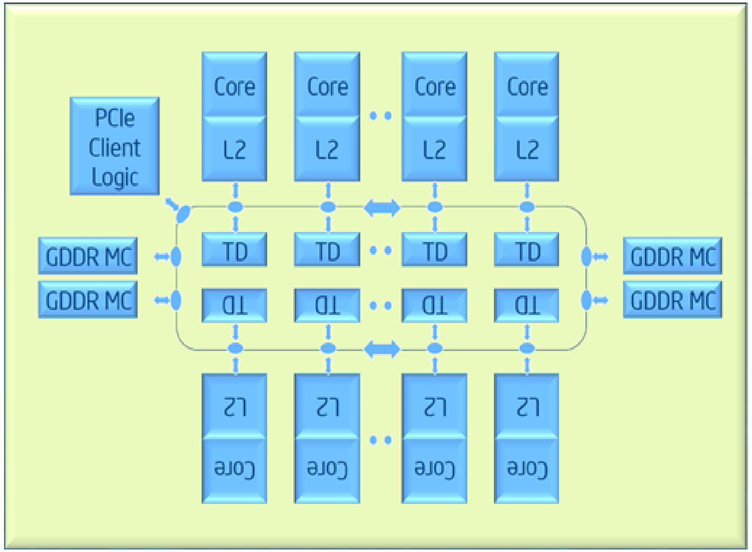
Each coprocessor connected to one host through PCIe bus

- PCIe Gen 2 (client) x16
  - Between 6-14 GB/s (relatively slow)
- Up to 8 coprocessors per host
- Inter-node coprocessors communication through Ethernet or InfiniBand
  - InfiniBand allows PCIe peer-to-peer interconnect without host intervention

Each coprocessor can be accessed as a network node

- It has its own IP address
- Runs a special uLinux OS (BusyBox)
  - Intel® Many Core Software Stack (MPSS)

Intel® Xeon Phi^{TM} uncore architecture



(c) 2013 Jim Jeffers and James Reinders, used with permission.

High bandwidth interconnect

- Bidirectional ring topology

Fully cache-coherent SMP on-a-chip

- Distributed global tag directory (TD)
- About 31 MB of “L2 cloud”
  - >100-cycle latency for remote L2 access

8-16 GB GDDR5 main memory (ECC)

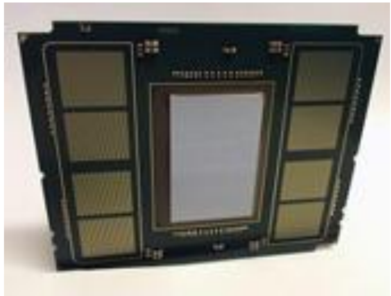
- 8 memory controllers (MC)
  - >300-cycle latency access
- 2 GDDR5 32-bit channels per MC
- Up to 5.5 GT/s per channel
- 352 GB/s max. theoretic bandwidth
  - Practical peak about 150-180 GB/s

ECC on GDDR5/L2 for reliability

Knights Landing: 2nd generation Intel® Xeon Phi™

Performance
3+ TeraFLOPS of double-precision peak theoretical performance per single socket node
3x Single-Thread Performance compared to Knights Corner
Most of today's parallel optimizations carry forward to KNL simply by recompile

Integration	
Intel® Omni Path™ fabric integration	
High-performance on-package memory (MCDRAM)	Over 5x STREAM vs. DDR4 (Over ~400 GB/s vs ~90 GB/s)
	Up to 16GB at launch
	NUMA support
	Over 5x Energy Efficiency vs. GDDR5
	Over 3x Density vs. GDDR5
	In partnership with Micron Technology
	Flexible memory modes including cache and flat

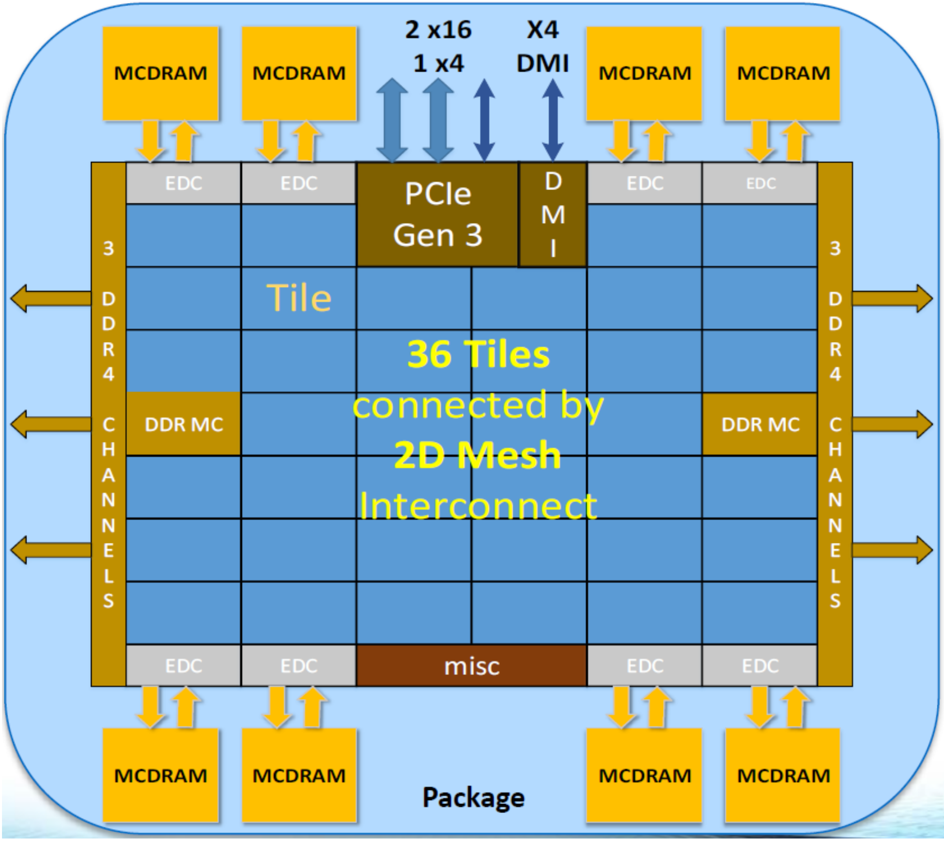


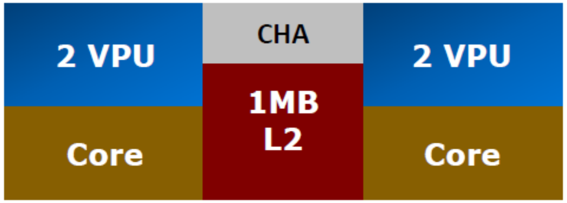
Microarchitecture	
Over 8 billion transistors per die based on Intel's 14 nanometer manufacturing technology	
Binary compatible with Intel® Xeon® Processors with support for Intel® Advanced Vector Extensions 512 (Intel® AVX-512)	
72 cores in a 2D Mesh architecture	
2 cores per tile with 2 VPUs per core	
1MB L2 cache shared between 2 cores in a tile (cache-coherent)	
Cores based on Intel® Atom™ (Silvermont) microarchitecture with many HPC enhancements	4 Threads / Core
	2X Out-of-Order Buffer Depth
	Gather/scatter in hardware
	Advanced Branch Prediction
	High cache bandwidth
	32KB Icache, Dcache
	2 x 64B Load ports in Dcache
	46/48 Physical/virtual address bits
Multiple NUMA domain support per socket	

Server processor
Standalone bootable processor (running host OS) and PCIe coprocessor
Platform memory: up to 384GB DDR4 using 6 channels
Reliability ("Intel server-class reliability")
Power Efficiency (Over 25% better than discrete coprocessor) → Over 10 GF/W
Density (3+ KNL with fabric in 1U)
Up to 36 lanes PCIe* Gen 3.0

Availability
First commercial HPC systems in 2H'15
Knights Corner to Knights Landing upgrade program available today
Intel Adams Pass board (1U half-width) is custom designed for Knights Landing (KNL) and will be available to system integrators for KNL launch; the board is OCP Open Rack 1.0 compliant, features 6 ch native DDR4 (1866/2133/2400MHz) and 36 lanes of integrated PCIe* Gen 3 I/O

Knights Landing platform overview





Single socket node

- 36 tiles connected by coherent 2D-Mesh
- Every tile is 2 OoO cores +
  2 512-bit VPU/core + 1 MB L2

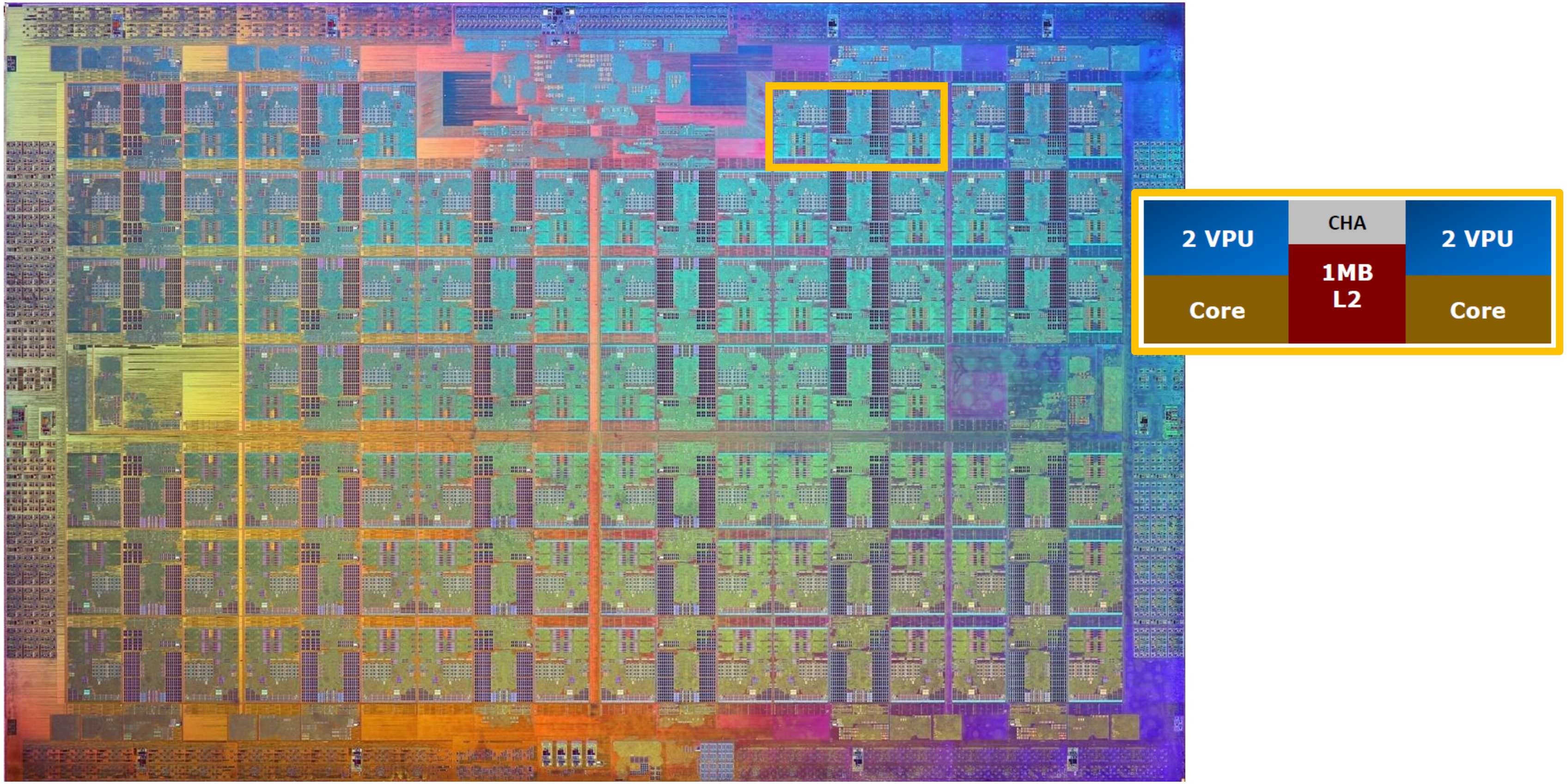
Memory

- MCDRAM, 16 GB on-package; High BW
- DDR4, 6 channels @ 2400 up to 384GB

IO & Fabric

- 36 lanes PCIe Gen3
- 4 lanes of DMI for chipset
- On-package Omni-Path fabric

Intel® Knights Landing die



2 VPU

CHA

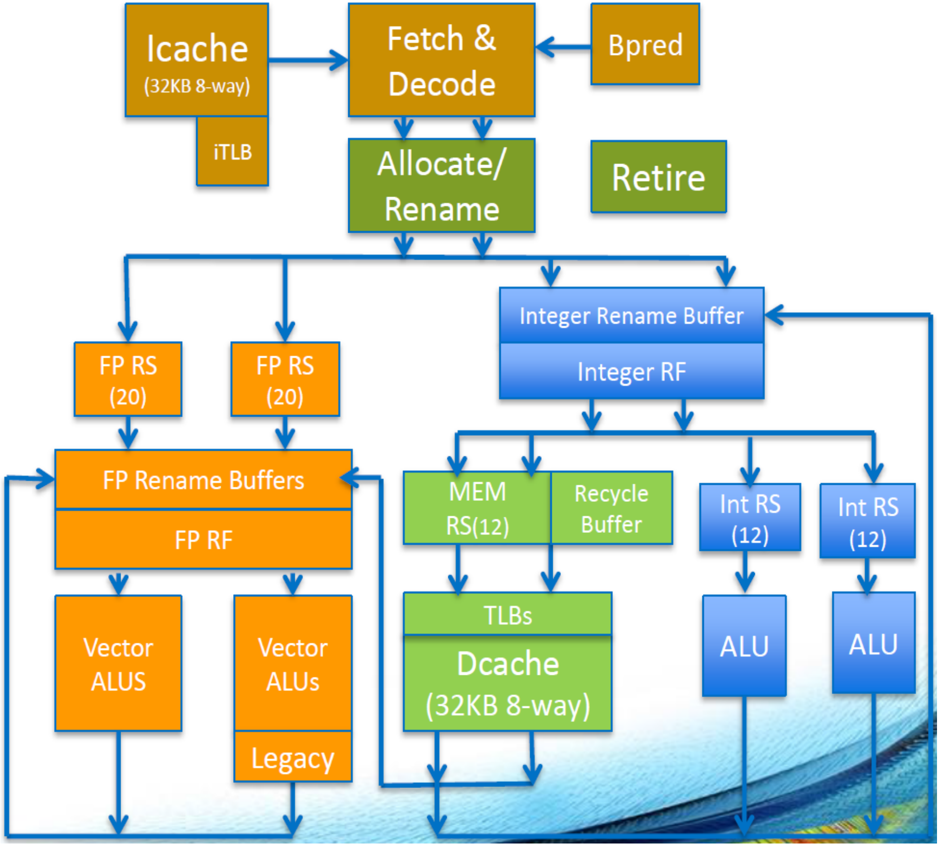
2 VPU

1MB L2

Core

Core

Knights Landing core architecture



OoO core w/ 4 SMT threads

- 2-wide decode/rename/retire
- Up to 6-wide at execution
- Int and FP RS OoO
- 2 AVX-512 VPUs

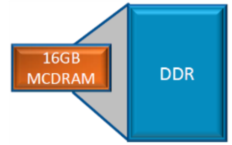
$s/TLBs

- 64-bit Dcache ports (2-load & 1-store)
- 1st level uTLB w/ 64 entries
- 2nd level dTLB w/ 256-4K, 128-2M, 16-1G pages

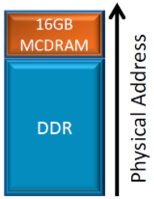
Others

- L1 (IPP) and L2 prefetcher.
- Fast unaligned support
- Fast gather/scatter support

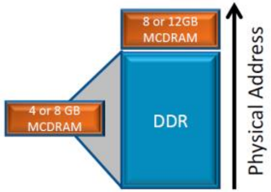
Knights Landing's on package HBM memory



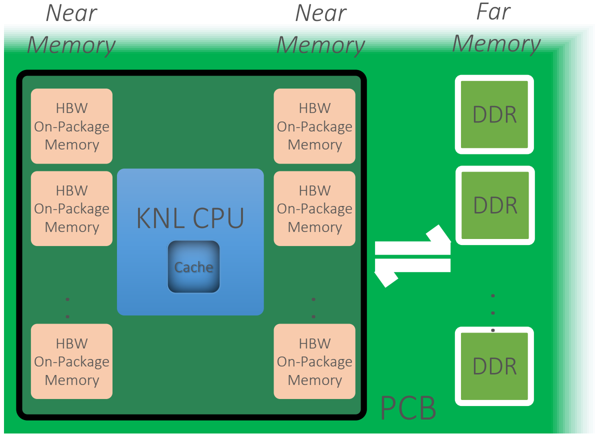
Cache Model Let the ***hardware automatically manage*** the integrated on-package memory as an "L3" direct-mapped cache between KNL CPU and external DDR



Flat Model ***Manually manage*** how your application uses the integrated on-package memory and external DDR for peak performance



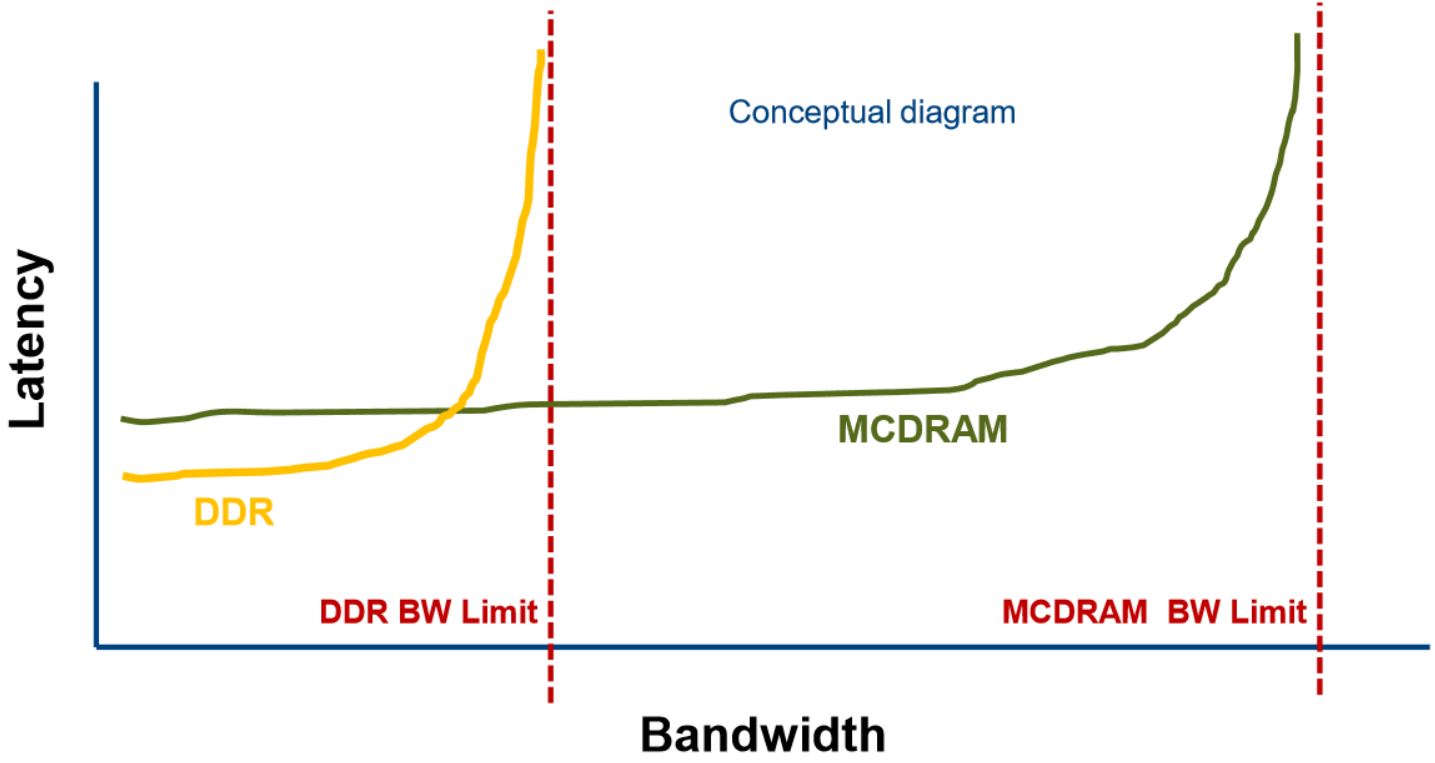
Hybrid Model Harness the ***benefits of both*** cache and flat models by segmenting the integrated on-package memory



Maximizes performance through higher memory bandwidth and flexibility

- Explicit allocation allowed with open-sourced API memkind, Fortran attributes, and C++ allocator
- Mode chosen at boot time

DDR and MCDRAM Bandwidth vs. Latency

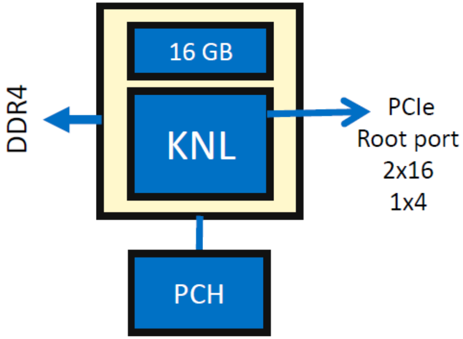
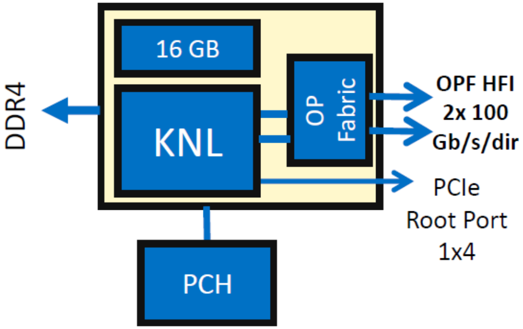
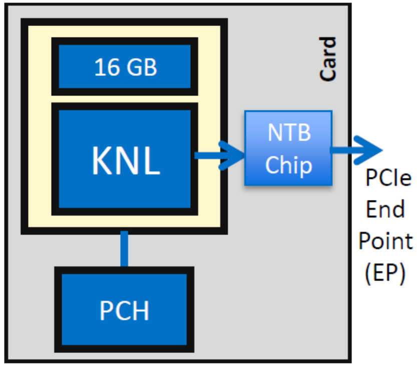


MCDRAM latency more than DDR at low loads but much less at high loads

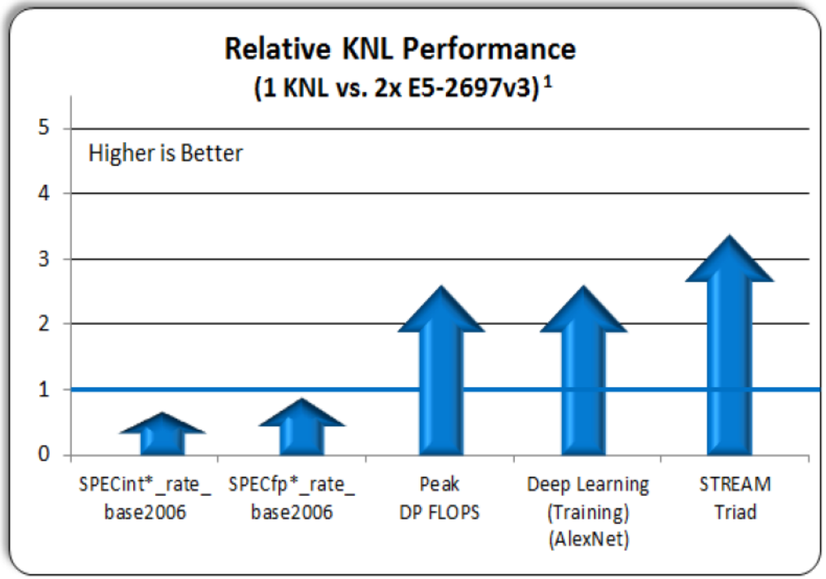
Memory Placement KNL specifics

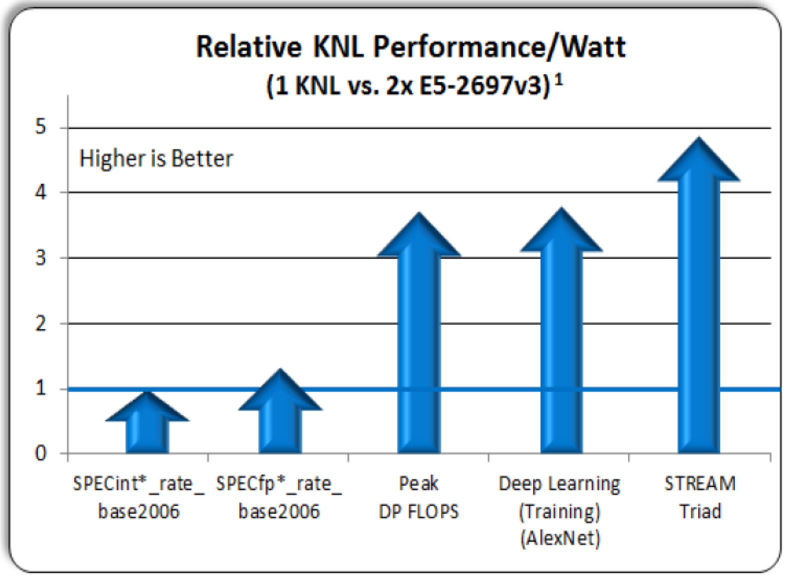
- Does the entire application fit MCDRAM?
- Flat, cache or hybrid mode
- Keep heavily used data in MCDRAM
- Consider affinity

Knights Landing products

 16 GB KNL DDR4 PCIe Root port 2x16 1x4 PCH	 16 GB KNL OP Fabric DDR4 OPF HFI 2x 100 Gb/s/dir PCIe Root Port 1x4 PCH	 Card 16 GB KNL NTB Chip PCIe End Point (EP) PCH
KNL	KNL w/ Omni-Path	KNL Card
6 DDR channels Up to 16 GB MCDRAM 36-lanes Gen3 PCIe (root port)	6 DDR channels Up to 16 GB MCDRAM 4-lanes Gen3 PCIe (root port) Omni-Path fabric (200 Gb/s/dir)	No DDR Channels Up to 16 GB MCDRAM 16-lanes Gen3 PCIe (end point) NTB Chip to create PCIe EP
Self boot socket		PCIe Card

Knights Landing performance





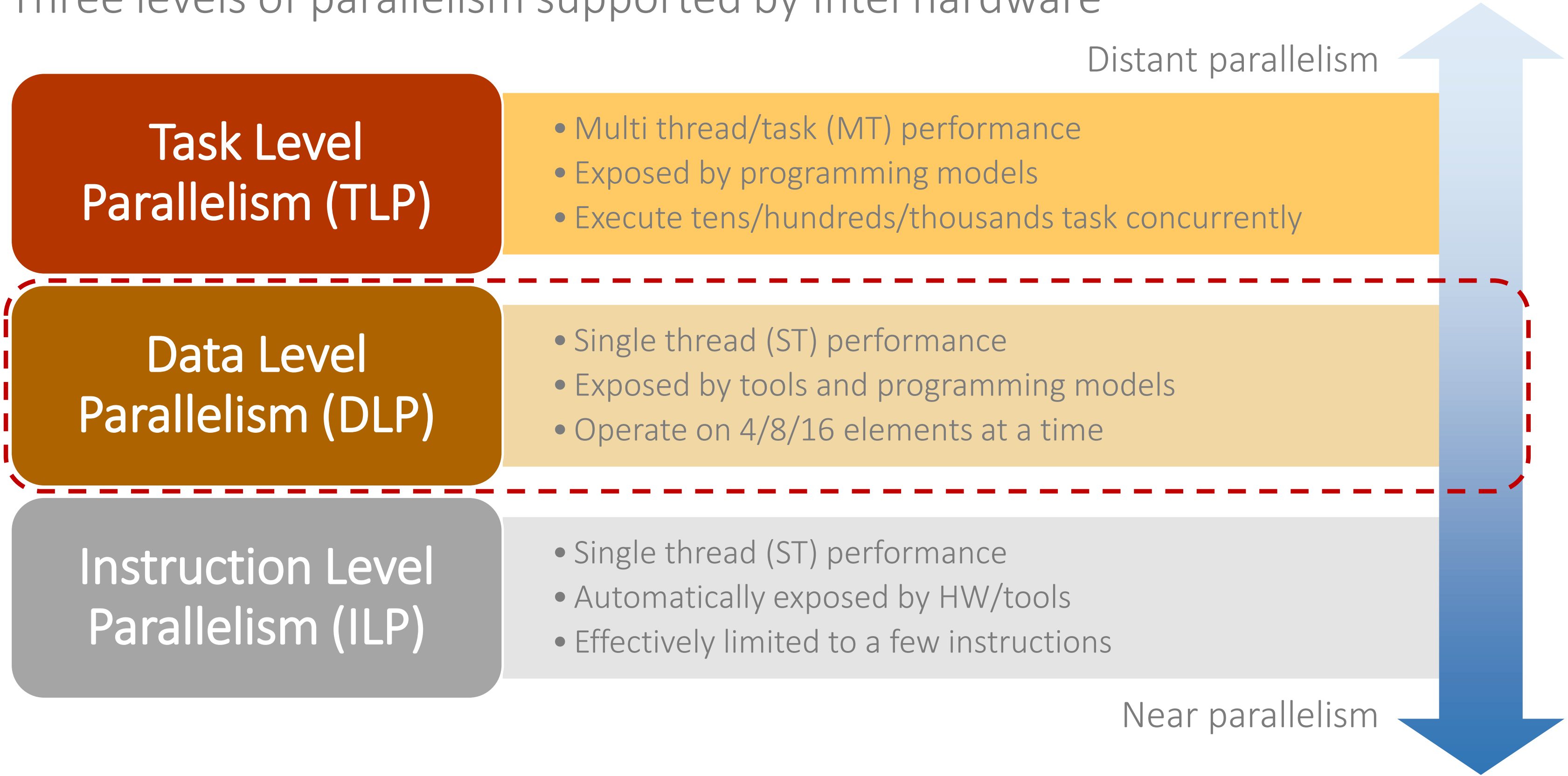
[1] Projected KNL Performance (1 socket, 200W CPU TDP) vs. 2 Socket Intel® Xeon® processor E5-2697v3 (2x145W CPU TDP)

Significant performance improvement for compute and bandwidth sensitive workloads, while still providing good general purpose throughput performance

Best practices for SIMD vectorization

Exploiting the parallel universe

Three levels of parallelism supported by Intel hardware



Distant parallelism

Task Level Parallelism (TLP)

- Multi thread/task (MT) performance
- Exposed by programming models
- Execute tens/hundreds/thousands task concurrently

Data Level Parallelism (DLP)

- Single thread (ST) performance
- Exposed by tools and programming models
- Operate on 4/8/16 elements at a time

Instruction Level Parallelism (ILP)

- Single thread (ST) performance
- Automatically exposed by HW/tools
- Effectively limited to a few instructions

Near parallelism

Programmers responsibility to expose DLP/TLP

Single Instruction Multiple Data (SIMD)

Technique for exploiting DLP on a single thread

- Operate on more than one element at a time
- Might decrease instruction counts significantly

Elements are stored on SIMD registers or *vectors*

Code needs to be *vectorized*

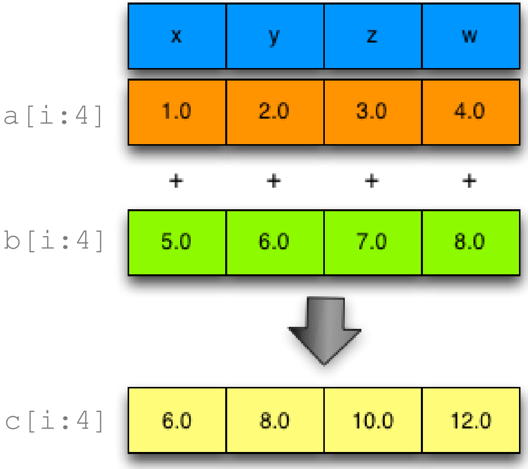
- Vectorization usually on *inner* loops
- Main and *remainder* loops are generated

```
for (int i = 0; i < N; i++)
    c[i] = a[i] + b[i];
```

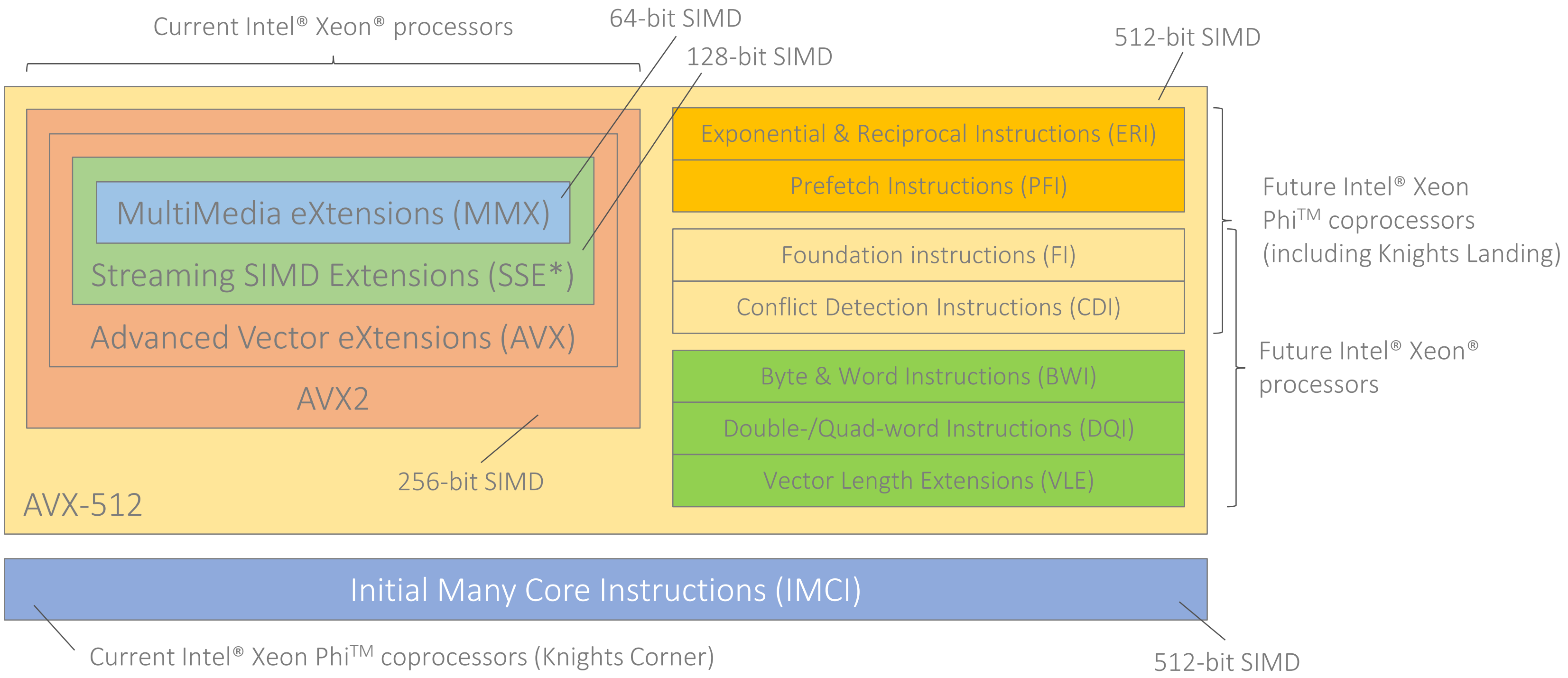
Scalar loop

```
for (int i = 0; i < N; i += 4)
    c[i:4] = a[i:4] + b[i:4];
```

SIMD loop
(4 elements)



Past, present, and future of Intel SIMD types


Current Intel® Xeon® processors
64-bit SIMD
128-bit SIMD
512-bit SIMD
MultiMedia eXtensions (MMX)
Streaming SIMD Extensions (SSE*)
Advanced Vector eXtensions (AVX)
AVX2
256-bit SIMD
AVX-512
Exponential & Reciprocal Instructions (ERI)
Prefetch Instructions (PFI)
Foundation instructions (FI)
Conflict Detection Instructions (CDI)
Byte & Word Instructions (BWI)
Double-/Quad-word Instructions (DQI)
Vector Length Extensions (VLE)
Future Intel® Xeon Phi™ coprocessors (including Knights Landing)
Future Intel® Xeon® processors
Initial Many Core Instructions (IMCI)
Current Intel® Xeon Phi™ coprocessors (Knights Corner)
512-bit SIMD

For more information about Intel® AVX-512 instructions, check out James Reinders' initial and updated post for this topic.

Intel® AVX2/IMCI/AVX-512 differences

	Intel® Initial Many Core Instructions **IMCI**	Intel® Advanced Vector Extensions 2 **AVX2**	Intel® Advanced Vector Extensions 512 **AVX-512**
Introduction	2012	2013	2015-2016
Products	Knights Corner	Haswell, Broadwell	Knights Landing, future Intel® Xeon® and Xeon® Phi™ products
Register file	SP/DP/int32/int64 data types 32 x 512-bit SIMD registers 8 x 16-bit mask registers	SP/DP/int32/int64 data types 16 x 256-bit SIMD registers No mask registers (instr. blending)	SP/DP/int32/int64 data types 32 x 512-bit SIMD registers 8 x (up to) 64-bit mask
ISA features	Not compatible with AVX*/SSE* No unaligned data support Embedded broadcast/cvt/swizzle MVEX encoding	Fully compatible with AVX/SSE* Unaligned data support (penalty) VEX encoding	Fully compatible with AVX*/SSE* Fast unaligned data support Embedded broadcast/rounding EVEX encoding
Instruction features	Fused multiply-and-add (FMA) Partial gather/scatter Transcendental support	Fused multiply-and-add (FMA) Full gather	Fused multiply-and-add (FMA) Full gather/scatter Transcendental support (ERI only) Conflict detection instructions PFI/BWI/DQI/VLE (if applies)

Intel® AVX-512 is a major step in unifying the instruction set of Intel® MIC and Intel® Xeon® architecture

Side effects of SIMD vectorization

```
float a[1024], b[1024], c[1024];
    …
for (int i = 0; i < 1024; i++)
    c[i] = a[i] + b[i];
```

#Instructions	Scalar	AVX2 (256-bit)	IMCI AVX-512 (512-bit)
Loads (hit) to a[], b[]	960 + 960	64 + 64	0
Loads (miss) to a[], b[]	64 + 64	64 + 64	64 + 64
SP adds	1024	128	64
Stores to c[]	1024	128	64
Total (Reduction)	4096 (x1)	512 (x8)	256 (x16)

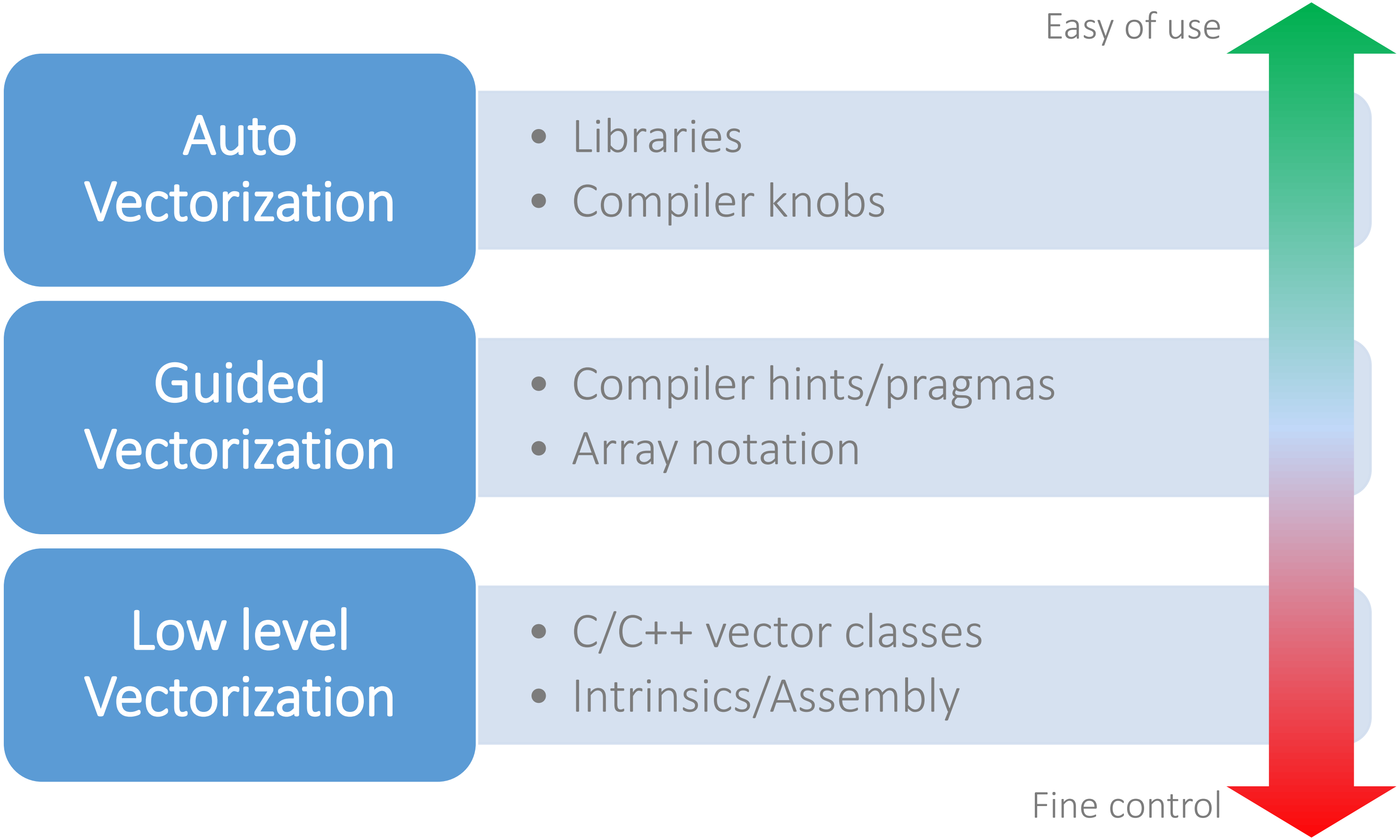
Assumptions

- 64-byte cache lines, 4-byte SP elements (float)
- 32-byte (AVX2) and 64-byte (IMCI/AVX-512) SIMD registers
- No hardware prefetcher, no ld+op instructions, arrays not cached

Observations

- Significant instruction count reduction (up to *vector-length*)
  - IPC decreases, but so does execution time as well
  - Usually translated into speedup
- Compute-bound codes turn into memory-bound codes
  - If code already was memory bound, no benefits at all (other than energy reduction)

Vectorization on Intel compilers



Easy of use

Auto Vectorization

- Libraries
- Compiler knobs

Guided Vectorization

- Compiler hints/pragmas
- Array notation

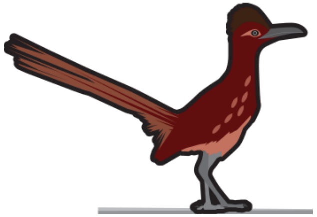
Low level Vectorization

- C/C++ vector classes
- Intrinsics/Assembly

Fine control

Rely on Intel® performance libraries

Highly efficient SIMD implementation of common functions for multiple Intel® processors



INTEL® INTEGRATED PERFORMANCE PRIMITIVES (INTEL® IPP)
A library of optimized building blocks for media and data applications. Take advantage of the unique capabilities of Intel processor families using optimized low-level APIs with significant emphasis on signal processing and certain media-focused applications, with cross-OS support and an internal dispatcher capable of selecting the prime optimization path.

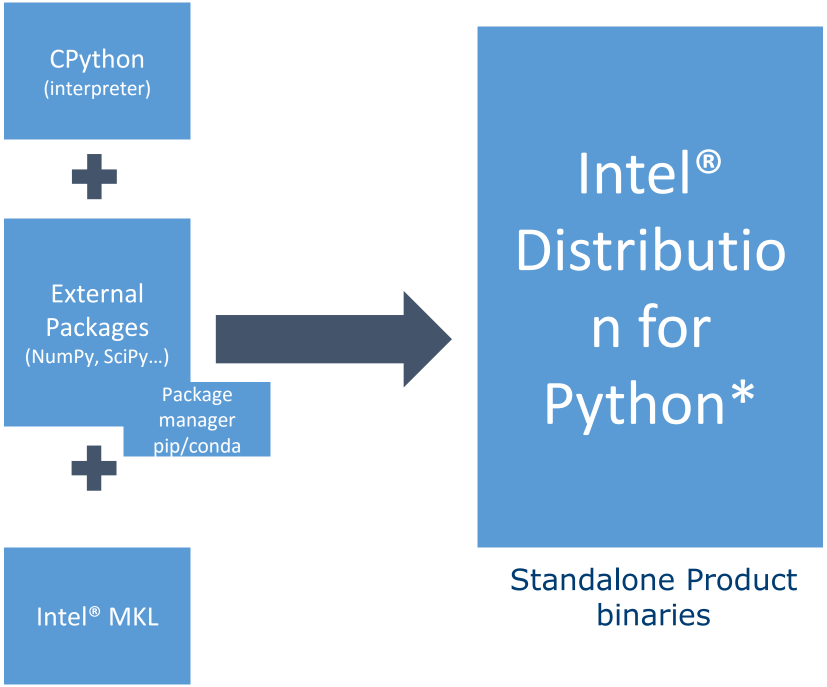
INTEL® MATH KERNEL LIBRARY (INTEL® MKL)
The fastest and most used math library for Intel® and compatible processors. Harness the power of today's processors—with increasing core counts, wider vector units, and more varied architectures. Includes highly vectorized and threaded linear algebra, fast Fourier Transforms, vector math, and statistics functions. Through a single API call, these functions automatically scale for future processor architectures by selecting the best code path for each.

INTEL® DATA ANALYTICS ACCELERATION LIBRARY (INTEL® DAAL)
Crunch more big data on the same node with Intel® DAAL for C++ and Java. The library provides highly optimized algorithmic building blocks to speed big data analytics performance on platforms from edge devices to servers. It encompasses data analysis stages (preprocessing, transformation, analysis, modeling, and decision making) for offline, streaming, and distributed analytics usages. Tight integration with popular data platforms (including Hadoop* and Spark*) enables highly efficient data access.

All libraries available at no cost with Community Licensing (Intel® support not included)

Intel® Distribution for Python*

- Performance-optimized Python Distribution for technical computing & data analysis
- Performance accelerations powered by Intel® MKL
- NumPy/SciPy packages accelerated with Intel® MKL
  - NumPy: fundamental package for scientific computation in Python. Support for large multi-dimensional arrays & matrices. High level mathematical functions
  - SciPy: science & engineering modules
  - Pandas, sympy, matplotlib, scikit-learn
- Easy, Intuitive product experience – easy installation, package management, access to performance
- Python 2.7 & 3.5
- Windows & Linux. Mac OS in 2016



Standalone Product binaries

Contents of Intel® Distribution for Python*

Access multiple options with our Python Distribution

Accelerate with native libraries ✓

- NumPy, SciPy, Scikit-Learn, Theano, Pandas, pyDAAL
- Intel MKL, Intel IPP, Intel DAAL

Exploit vectorization and threading ✓

- Cython + Intel C++ compiler
- Numba + Intel LLVM

Better/Composable threading ✓

- Cython, Numba
- Threading composability for MKL, CPython, Blaze/Dask, Numba

Multi-node parallelism ✓

- Mpi4Py, Distarray
- Intel native libraries: Intel MPI

Integration with Big Data, ML platforms and frameworks

- Spark, Hadoop, Trusted Analytics Platform

Work in Progress

Better performance profiling ✓

- Extensions for profiling mixed Python & native/JIT codes

Join the Intel® Distribution for Python* 2017 Beta

Auto vectorization

For C/C++ and Fortran

Relies on the compiler for vectorization of inner loops

- No source code changes
- Enabled with `-vec` compiler knob (default in `-O2` and `-O3` optimization levels)

Opt. level	Description
`-O0`	Disables all optimizations.
`-O1`	Enables optimizations for speed which are know to not cause code size increase.
`-O2/-O` (default)	Enables intra-file interprocedural optimizations for speed, including: • Vectorization • Loop unrolling
`-O3`	Performs `-O2` optimizations and enables more aggressive loop transformations such as: • Loop fusion • Block unroll-and-jam • Collapsing IF statements This option is recommended for applications that have loops that heavily use floating-point calculations and process large data sets. However, it might incur in slower code, numerical stability issues, and compilation time increase.

Auto vectorization: not all loops will vectorize

Data dependencies between iterations

- Proven Read-after-Write data (i.e., loop carried) dependencies
- Assumed data dependencies
    - Aggressive optimizations (e.g., IPO) might help

RaW dependency

```
for (int i = 0; i < N; i++)
    a[i] = a[i-1] + b[i];
```

Vectorization won’t be efficient

- Compiler estimates how better the vectorized version will be
- Affected by data alignment, data layout, etc.

Inefficient vectorization

```
for (int i = 0; i < N; i++)
    a[c[i]] = b[d[i]];
```

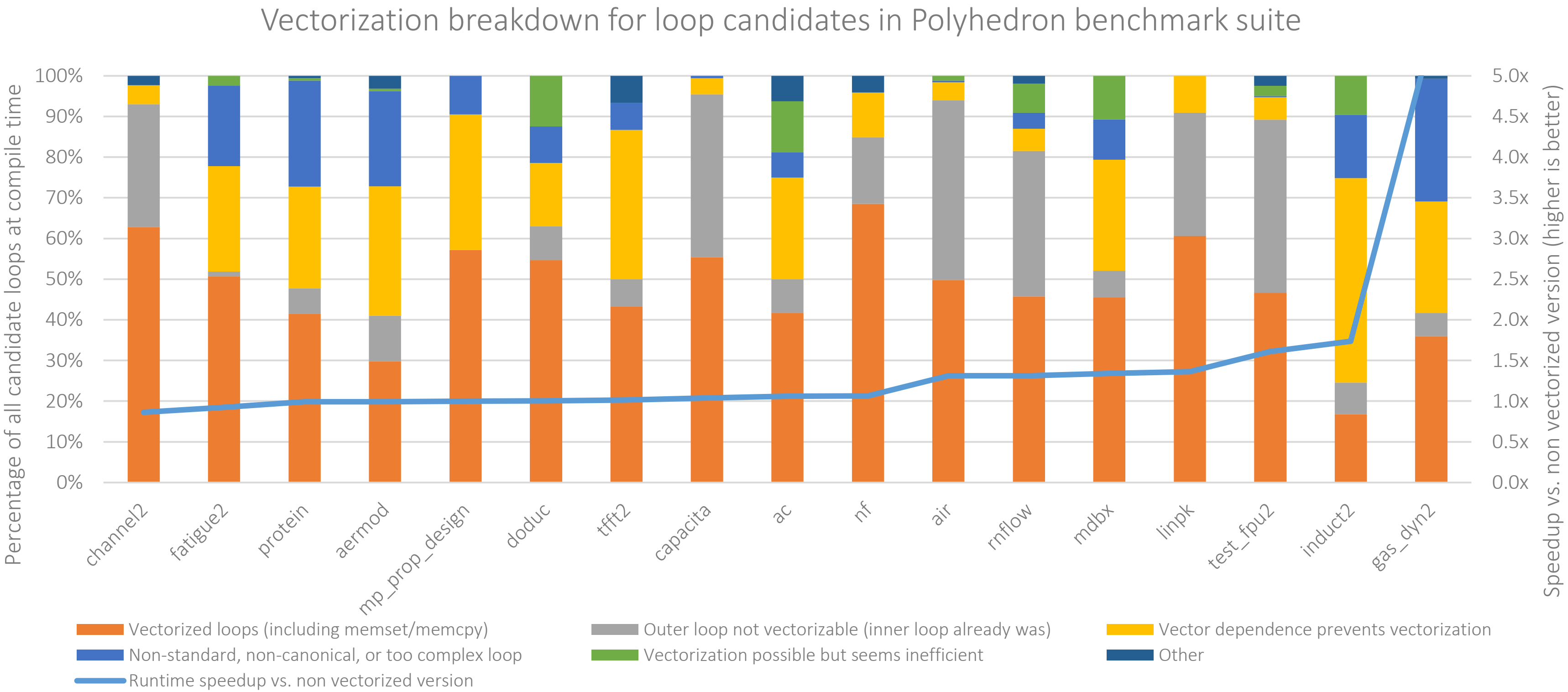
Unsupported loop structure

- While-loop, for-loop with unknown number of iterations
- Complex loops, unsupported data types, etc.
- (Some) function calls within loop bodies
    - Not the case for SVML functions

Function call within loop body

```
for (int i = 0; i < N; i++)
    a[i] = foo(b[i]);
```

Auto vectorization on Intel compilers



Polyhedron benchmark suite
Intel® Xeon Phi^{TM} 7120A, 61 cores x 4 threads
Intel® Fortran Compiler 15.0.1.14 [`-O3 -fp-model fast=2 -align array64byte -ipo -mmic`]

Validating vectorization success

Generate compiler report about optimizations

`-qopt-report[=n]`	Generate report (level [1..5], default 2)
`-qopt-report-file=<fname>`	Optimization report file (stderr, stdout also valid)
`-qopt-report-phase=<phase>`	Info about opt. phase:

`loop`	Loop nest optimizations
`par`	Auto-parallelization
`vec`	**Vectorization**
`openmp`	OpenMP
`offload`	Offload
`ipo`	Interprocedural optimizations
`pgo`	Profile Guided optimizations
`cg`	Code generation optimizations
`tcollect`	Trace analyzer (MPI) collection
`all`	All optimizations (default)

```
LOOP BEGIN at gas_dyn2.f90(193,11) inlined into gas_dyn2.f90(4326,31)
   remark #15300: LOOP WAS VECTORIZED
   remark #15448: unmasked aligned unit stride loads: 1
   remark #15450: unmasked unaligned unit stride loads: 1
   remark #15475: --- begin vector loop cost summary ---
   remark #15476: scalar loop cost: 53
   remark #15477: vector loop cost: 14.870
   remark #15478: estimated potential speedup: 2.520
   remark #15479: lightweight vector operations: 19
   remark #15481: heavy-overhead vector operations: 1
   remark #15488: --- end vector loop cost summary ---
   remark #25456: Number of Array Refs Scalar Replaced In Loop: 1
   remark #25015: Estimate of max trip count of loop=4
LOOP END
```

Vectorized loop

```
LOOP BEGIN at gas_dyn2.f90(2346,15)
   remark #15344: loop was not vectorized: vector dependence prevents vectorization
   remark #15346: vector dependence: assumed OUTPUT dependence between IOLD line 376 and IOLD line 354
   remark #25015: Estimate of max trip count of loop=3000001
LOOP END
```

Non-vectorized loop

Guiding vectorization: disambiguation hints

Get rid of assumed vector dependencies

Assume function arguments won’t be aliased

- C/C++: compile with `-fargument-noalias`

C99 “restrict” keyword for pointers

- Or compile with `-restrict` knob

```
void v_add(float *restrict c,
           float *restrict a,
           float *restrict b)
{
    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];
}
```

Ignore assumed vector dependencies with Intel-specific compiler directive

- C/C++: `#pragma ivdep`
- Fortran: `!dir$ ivdep`

```
void v_add(float *c, float *a, float *b)
{
#pragma ivdep
    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];
}
```

Target architecture compiler options

On which architecture do we want to run our program?

Option	Description
`-mmic`	Builds an application that runs natively on Intel® MIC Architecture.

`-x`*`feature`*
`-xHost`
	Tells the compiler which processor features it may target, referring to which instruction sets and optimizations it may generate (not available for Intel® Xeon Phi^{TM} architecture). Values for *feature* are:
	• **`COMMON-AVX512`** (includes AVX512 FI and CDI instructions)
	• **`MIC-AVX512`** (includes AVX512 FI, CDI, PFI, and ERI instructions)
	• **`CORE-AVX512`** (includes AVX512 FI, CDI, BWI, DQI, and VLE instructions)
	• **`CORE-AVX2`**
	• `CORE-AVX-I` (including RDRND instruction)
	• `AVX`
	• `SSE4.2, SSE4.1`
	• `ATOM_SSE4.2, ATOM_SSSE3` (including MOVBE instruction)
	• `SSSE3, SSE3, SSE2`
	When using **`-xHost`**, the compiler will generate instructions for the highest instruction set available on the compilation host processor.

`-ax`*`feature`*	Tells the compiler to generate multiple, feature-specific auto-dispatch code paths for Intel® processors if there is a performance benefit. Values for *feature* are the same described for `-xfeature` option. Multiple features/paths possible, e.g.: `-axSSE2,AVX`. It also generates a baseline code path for the default case.

Vectorized code will be different depending on the chosen target architecture

Some Intel-specific compiler directives

For C/C++ and Fortran

Directive	Description
`[no]block_loop`	Enables or disables loop blocking for the immediately following nested loops.
`distribute, distribute_point`	Instructs the compiler to prefer loop distribution at the location indicated.
`inline`	Instructs the compiler to inline the calls in question.
`ivdep`	Instructs the compiler to ignore assumed vector dependencies.
`loop_count`	Indicates the loop count is likely to be an integer.
`optimization_level`	Enables control of optimization for a specific function.
`parallel/noparallel`	Facilitates auto-parallelization of an immediately following loop; using keyword `always` forces the compiler to auto-parallelize; `noparallel` pragma prevents auto-parallelization.
`[no]unroll`	Instructs the compiler the number of times to unroll/not to unroll a loop
`[no]unroll_and_jam`	Prevents or instructs the compiler to partially unroll higher loops and jam the resulting loops back together.
`unused`	Describes variables that are unused (warnings not generated).
`[no]vector`	Specifies whether the loop should be vectorised. In case of forcing vectorization that should be according to the given clauses.

Enforcing vectorization with SIMD directives

Intel-specific idioms

C/C++ (also part of Cilk™ Plus)

- Enforcing loop vectorization ignoring **all** dependencies
  - `#pragma simd` in front of vectorizable loop
  - `_Simd` keyword right after `for/cilk_for` loop keyword
- Declaring vectorized functions
  - `__attribute__((vector))/__declspec(vector)` on Linux/Windows

```
void vadd(float *c, float *a, float *b) {
#pragma simd
    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];
}
```

SIMD loop

```
__declspec(vector)
void vadd(float c, float a, float b) {
    c = a + b;
}
    …
for (int i = 0; i < N; i++)
    vadd(C[i], A[i], B[i]);
```

SIMD function

Fortran

- `!dir$ simd`, `!dir$ attributes vector`

All directive idioms accept additional clauses (e.g., define reductions, etc.)

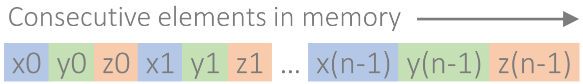
Improving vectorization: data layout

Vectorization more efficient with unit strides

- Non-unit strides will generate gather/scatter
- Unit strides also better for data locality
- Compiler might refuse to vectorize

Array of Structures (AoS)

```
struct coordinate {
    float x, y, z;
} crd[N];
    …
for (int i = 0; i < N; i++)
    … = … f(crd[i].x, crd[i],y, crd[i].z);
```



Consecutive elements in memory

Layout your data as Structure of Arrays (SoA)

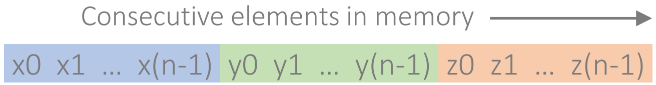
- As opposite to Array of Structures (SoA)

Traverse matrices in the right direction

- C/C++: `a[i][:]`, Fortran: `a(:,i)`
- Loop interchange might help
  - Usually the compiler is smart enough to apply it
  - Check compiler optimization report

Structure of Arrays (SoA)

```
struct coordinate {
    float x[N], y[N], z[N];
} crd;
    …
for (int i = 0; i < N; i++)
    … = … f(crd.x[i], crd.y[i], crd.z[i]);
```



Consecutive elements in memory

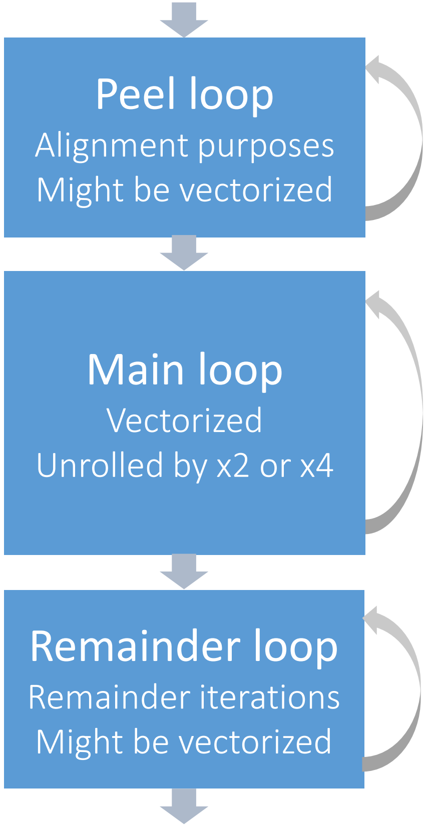
Improving vectorization: data alignment (cont'd)

How to…	Language	Syntax	Semantics
…align data	C/C++	`void* _mm_malloc(int size, int n)`	Allocate memory on heap aligned to *n* byte boundary.
	C/C++	`int posix_memalign (void **p, size_t n, size_t size)`	
	C/C++	`__declspec(align(n)) array` (Windows) `__attribute__(align(n)) array` (Linux)	Alignment for variable declarations.
	C++11	`alignas(expression|type)`	
	Fortran (not in common section)	`!dir$ attributes align:n::array`	
	Fortran (compiler option)	`-alignnbyte`	
…tell the compiler about it	C/C++	`#pragma vector aligned`	Vectorize assuming all array data accessed are aligned (may cause fault otherwise).
	Fortran	`!dir$ vector aligned`	
	C/C++	`__assume_aligned(array, n)`	Compiler may assume array is aligned to *n* byte boundary.
	Fortran	`!dir$ assume_aligned array:n`	

n=64 for Intel® Xeon Phi™ coprocessors, *n*=32 for AVX, *n*=16 for SSE

Padding might be necessary to guarantee aligned access to matrices

Vectorization with multi-version loops

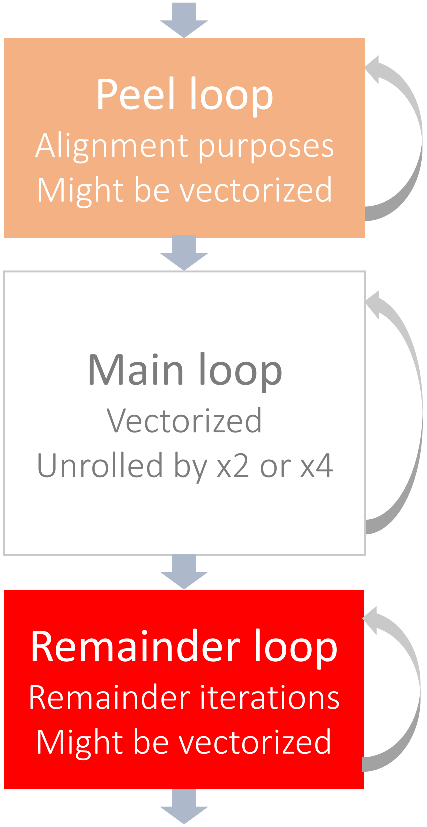


```
LOOP BEGIN at gas_dyn2.f90(2330,26)
<Peeled>
   remark #15389: vectorization support: reference AMAC1U has unaligned access
   remark #15381: vectorization support: unaligned access used inside loop body
   remark #15301: PEEL LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at gas_dyn2.f90(2330,26)
   remark #25084: Preprocess Loopnests: Moving Out Store
   remark #15388: vectorization support: reference AMAC1U has aligned access
   remark #15399: vectorization support: unroll factor set to 2
   remark #15300: LOOP WAS VECTORIZED
   remark #15475: --- begin vector loop cost summary ---
   remark #15476: scalar loop cost: 8
   remark #15477: vector loop cost: 0.620
   remark #15478: estimated potential speedup: 15.890
   remark #15479: lightweight vector operations: 5
   remark #15488: --- end vector loop cost summary ---
   remark #25018: Total number of lines prefetched=4
   remark #25019: Number of spatial prefetches=4, dist=8
   remark #25021: Number of initial-value prefetches=6
LOOP END

LOOP BEGIN at gas_dyn2.f90(2330,26)
<Remainder>
   remark #15388: vectorization support: reference AMAC1U has aligned access
   remark #15388: vectorization support: reference AMAC1U has aligned access
   remark #15301: REMAINDER LOOP WAS VECTORIZED
LOOP END
```

Improving vectorization: trip count hints



Vectorization can be seen as aggressive unrolling

- Main loop usually unrolled by x2 or x4
- Peel and remainder loop are vectorized with masks
- If trip count is low, vectorization might not be efficient
  - Remainder loop becomes the hotspot

Take a look at remainder loops

- Specify loop trip counts for efficient vectorization
  - `#pragma/!dir$ loop_count (n1,[n2…])`
  - `#pragma/!dir$ loop_count min(n1),max(n2),avg(n3)`
- Consider safe padding option (Intel® Xeon Phi^{TM} only)
  - Otherwise, remainder loops using gather/scatter loops
  - `-qopt-assume-safe-padding` to avoid it

Low level (explicit) vectorization

A.k.a "ninja programming" (C/C++ only)

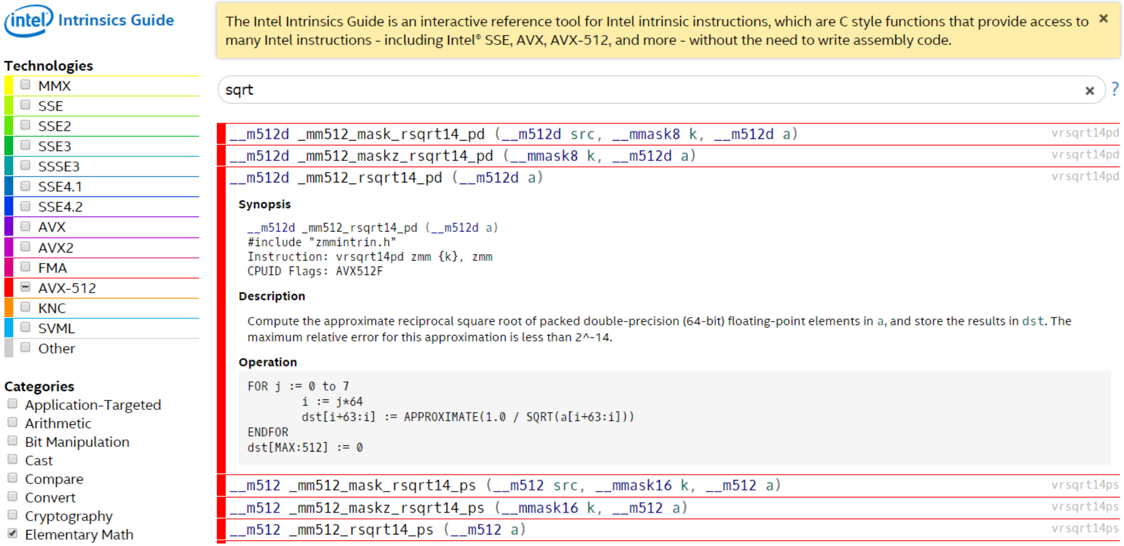
Vectorization relies on the programmer with some help from the compiler

Might be convenient for low level performance tuning of critical hotspots

Not portable among different SIMD architectures

SIMD C++ classes	Intrinsics	Inline assembly

`#include <fvec.h>`
 	
`F32vec4 a,b,c;`
`a = b + c;`
	`#include <xmmintrin.h>`
	
	`__m128 a,b,c;`
	`a = _mm_add_ps(b,c);`
		`__m128 a,b,c;`
		`__asm {`
		`    movaps xmm0,b`
		`    movaps xmm1,c`
		`    addps xmm0,xmm1`
		`    movaps a, xmm0`
		`}`



The Intel Intrinsics 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, AVX-512, and more - without the need to write assembly code. ×

Technologies

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

Categories

- □ Application-Targeted
- □ Arithmetic
- □ Bit Manipulation
- □ Cast
- □ Compare
- □ Convert
- □ Cryptography
- ☑ Elementary Math

sqrt × ?

```
__m512d _mm512_mask_rsqrt14_pd (__m512d src, __mmask8 k, __m512d a)      vrsqrt14pd
__m512d _mm512_maskz_rsqrt14_pd (__mmask8 k, __m512d a)                   vrsqrt14pd
__m512d _mm512_rsqrt14_pd (__m512d a)                                     vrsqrt14pd
```

Synopsis

```
__m512d _mm512_rsqrt14_pd (__m512d a)
#include "zmmintrin.h"
Instruction: vrsqrt14pd zmm {k}, zmm
CPUID Flags: AVX512F
```

Description

Compute the approximate reciprocal square root of packed double-precision (64-bit) floating-point elements in a, and store the results in dst. The maximum relative error for this approximation is less than 2^-14.

Operation

```
FOR j := 0 to 7
        i := j*64
        dst[i+63:i] := APPROXIMATE(1.0 / SQRT(a[i+63:i]))
ENDFOR
dst[MAX:512] := 0
```

```
__m512 _mm512_mask_rsqrt14_ps (__m512 src, __mmask16 k, __m512 a)       vrsqrt14ps
__m512 _mm512_maskz_rsqrt14_ps (__mmask16 k, __m512 a)                  vrsqrt14ps
__m512 _mm512_rsqrt14_ps (__m512 a)                                     vrsqrt14ps
```

Design, analysis and verification tools

Intel® Parallel Studio XE 2016 (Professional/Cluster Editions)

Intel® Advisor XE 2016

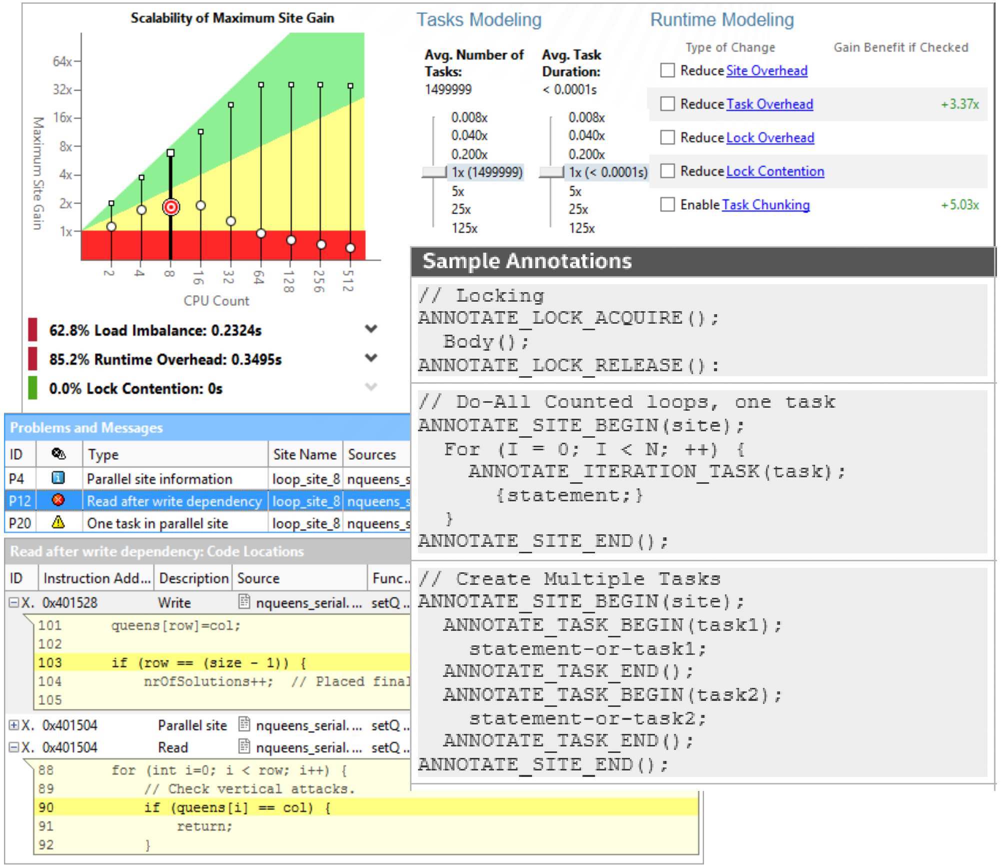
A design/analysis tool for ***threading*** your code

“What-if” analysis tool for thread design and prototyping

- Analyze, design, tune, and check your threading design before implementation
- Explore and test threading options without disrupting normal development
- Predict performance scaling on Intel® Xeon® and Xeon Phi™ architectures

What’s new in 2016 version?

- Tool completely redesign to add vectorization capabilities as well



🟥 62.8% Load Imbalance: 0.2324s

🟥 85.2% Runtime Overhead: 0.3495s

🟩 0.0% Lock Contention: 0s

Tasks Modeling

Avg. Number of Tasks:	Avg. Task Duration:
1499999	< 0.0001s
0.008x	0.008x
0.040x	0.040x
0.200x	0.200x
1x (1499999)	1x (< 0.0001s)
5x	5x
25x	25x
125x	125x

Runtime Modeling

Type of Change	Gain Benefit if Checked
□ Reduce Site Overhead	
□ Reduce Task Overhead	+3.37x
□ Reduce Lock Overhead	
□ Reduce Lock Contention	
□ Enable Task Chunking	+5.03x

Sample Annotations

```
// Locking
ANNOTATE_LOCK_ACQUIRE();
  Body();
ANNOTATE_LOCK_RELEASE():
```

```
// Do-All Counted loops, one task
ANNOTATE_SITE_BEGIN(site);
  For (I = 0; I < N; ++) {
    ANNOTATE_ITERATION_TASK(task);
      {statement;}
  }
ANNOTATE_SITE_END();
```

```
// Create Multiple Tasks
ANNOTATE_SITE_BEGIN(site);
  ANNOTATE_TASK_BEGIN(task1);
    statement-or-task1;
  ANNOTATE_TASK_END();
  ANNOTATE_TASK_BEGIN(task2);
    statement-or-task2;
  ANNOTATE_TASK_END();
ANNOTATE_SITE_END();
```

Problems and Messages

ID		Type	Site Name	Sources
P4	ℹ	Parallel site information	loop_site_8	nqueens_s
P12	⊗	Read after write dependency	loop_site_8	nqueens_s
P20	⚠	One task in parallel site	loop_site_8	nqueens_s

Read after write dependency: Code Locations

ID	Instruction Add...	Description	Source	Func...
⊟X.	0x401528	Write	nqueens_serial. ...	setQ ..

```
    queens[row]=col;

    if (row == (size - 1)) {
       nrOfSolutions++;  // Placed final

```

ID	Instruction Add...	Description	Source	Func...
⊞X.	0x401504	Parallel site	nqueens_serial. ...	setQ ..
⊟X.	0x401504	Read	nqueens_serial. ...	setQ ..

```
    for (int i=0; i < row; i++) {
       // Check vertical attacks.
       if (queens[i] == col) {
          return;
       }
```

NEW!

Intel® Advisor XE 2016

A design/analysis tool for vectorising your code

Survey analysis

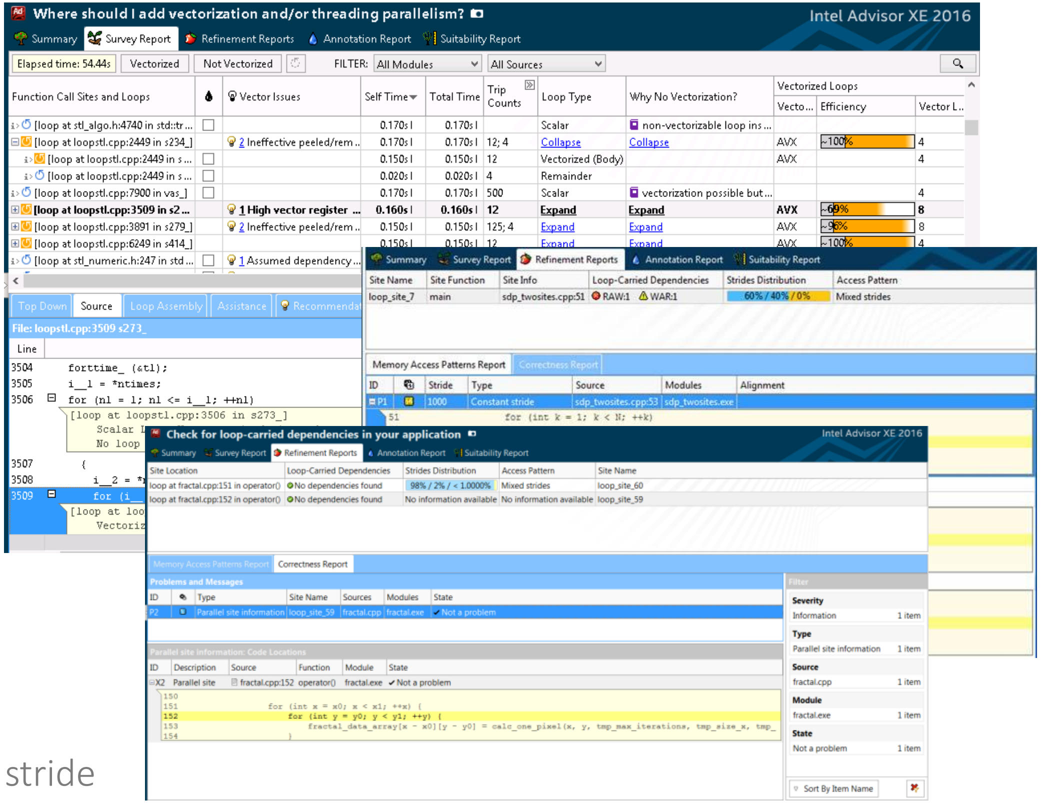
- See what prevents vectorization
- Detect vectorization issues
- Source/assembly integration
- Optimization reports
- Automatic recommendations

Trip-count analysis

- How many iterations in a loop
- Quantify peel/main/remainder

Deeper analyses

- Correctness analysis to see if a loop can be safely vectorized
- Memory access pattern (MAP) to figure out actual vectorization stride



Where should I add vectorization and/or threading parallelism?

Intel Advisor XE 2016

Elapsed time: 54.44s

Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	Vectorized Loops		
								Vecto...	Efficiency	Vector L...
[loop at stl_algo.h:4740 in std::tr...	□		0.170s	0.170s		Scalar	non-vectorizable loop ins...			
[loop at loopstl.cpp:2449 in s234_]		2 Ineffective peeled/rem...	0.170s	0.170s	12; 4	Collapse	Collapse	AVX	~100%	4
[loop at loopstl.cpp:2449 in s...	□		0.150s	0.150s	12	Vectorized (Body)		AVX		4
[loop at loopstl.cpp:2449 in s...	□		0.020s	0.020s	4	Remainder				
[loop at loopstl.cpp:7900 in vas_]	□		0.170s	0.170s	500	Scalar	vectorization possible but...			4
[loop at loopstl.cpp:3509 in s2...		**1 High vector register ...**	**0.160s**	**0.160s**	**12**	**Expand**	**Expand**	**AVX**	**~69%**	**8**
[loop at loopstl.cpp:3891 in s279_]		2 Ineffective peeled/rem...	0.150s	0.150s	125; 4	Expand	Expand	AVX	~96%	8
[loop at loopstl.cpp:6249 in s414_]			0.150s	0.150s	12	Expand	Expand	AVX	~100%	4
[loop at stl_numeric.h:247 in std...	□	1 Assumed dependency...								

File: loopstl.cpp:3509 s273_

```
Line
3504        forttime_ (&tl);
3505        i__1 = *ntimes;
3506  ⊟     for (nl = 1; nl <= i__1; ++nl)
          [loop at loopstl.cpp:3506 in s273_]
              Scalar L
              No loop
3507          {
3508              i__2 = *
3509  ⊟           for (i__
          [loop at loo
              Vectoriz
```

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_7	main	sdp_twosites.cpp:51	RAW:1 ⚠ WAR:1	60% / 40% / 0%	Mixed strides

ID		Stride	Type	Source	Modules	Alignment
P1		1000	Constant stride	sdp_twosites.cpp:53	sdp_twosites.exe	

```
51          for (int k = 1; k < N; ++k)
```

Check for loop-carried dependencies in your application

Intel Advisor XE 2016

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Site Name
loop at fractal.cpp:151 in operator()	No dependencies found	98% / 2% / < 1.0000%	Mixed strides	loop_site_60
loop at fractal.cpp:152 in operator()	No dependencies found	No information available	No information available	loop_site_59

Problems and Messages

ID		Type	Site Name	Sources	Modules	State
P2		Parallel site information	loop_site_59	fractal.cpp	fractal.exe	✓ Not a problem

Parallel site information: Code Locations

ID	Description	Source	Function	Module	State
X2	Parallel site	fractal.cpp:152	operator()	fractal.exe	✓ Not a problem

```
150
151             for (int x = x0; x < x1; ++x) {
152                 for (int y = y0; y < y1; ++y) {
153                     fractal_data_array[x - x0][y - y0] = calc_one_pixel(x, y, tmp_max_iterations, tmp_size_x, tmp_
154                 }
```

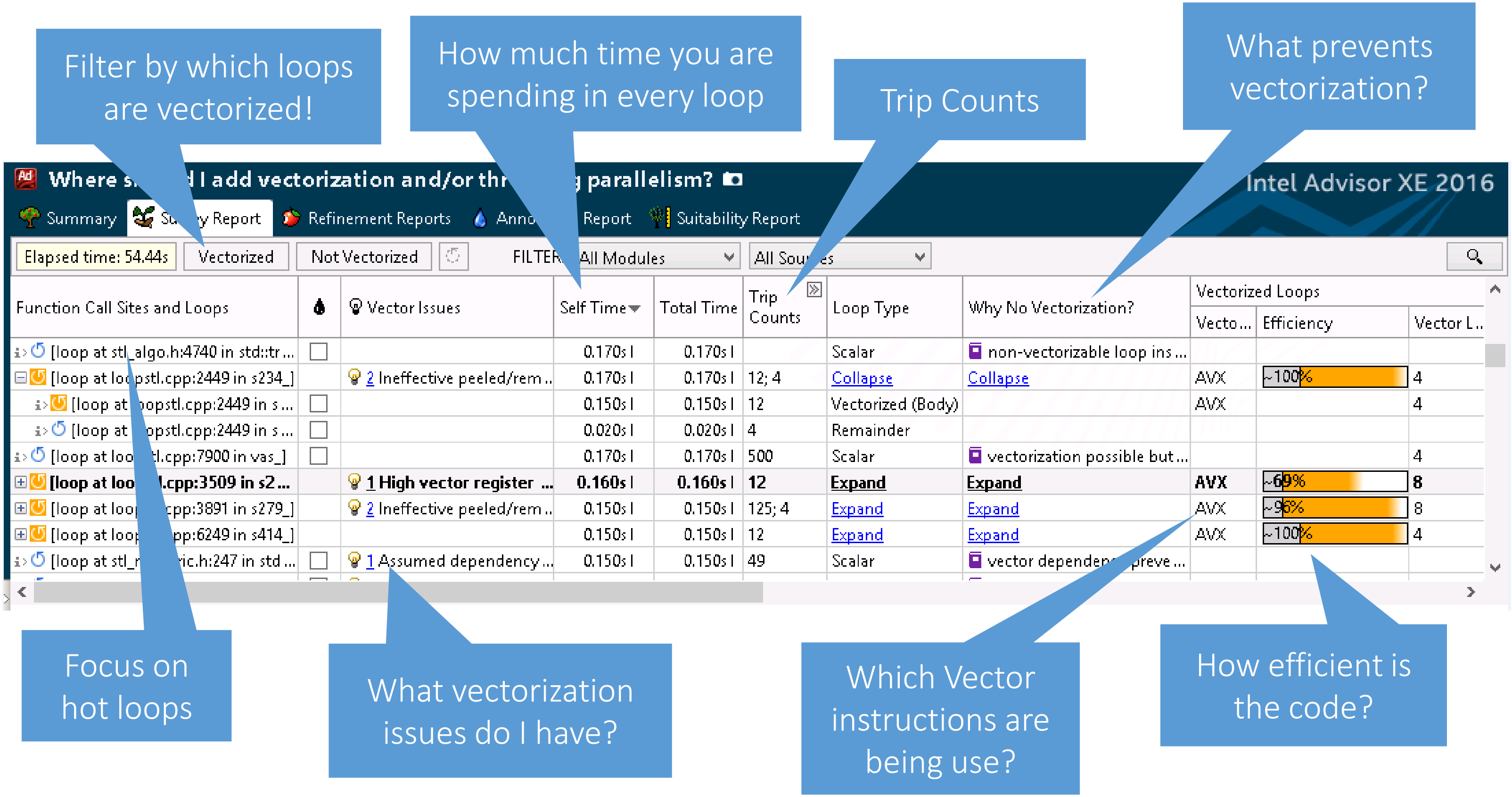
Filter

Severity	
Information	1 item
Type	
Parallel site information	1 item
Source	
fractal.cpp	1 item
Module	
fractal.exe	1 item
State	
Not a problem	1 item

Complete tutorial in latest Intel's magazine "The Parallel Universe" (Issue 22)

Survey report: the right data at your fingertips

Get all the data you need for high impact vectorization



Filter by which loops are vectorized!

How much time you are spending in every loop

Trip Counts

What prevents vectorization?

Focus on hot loops

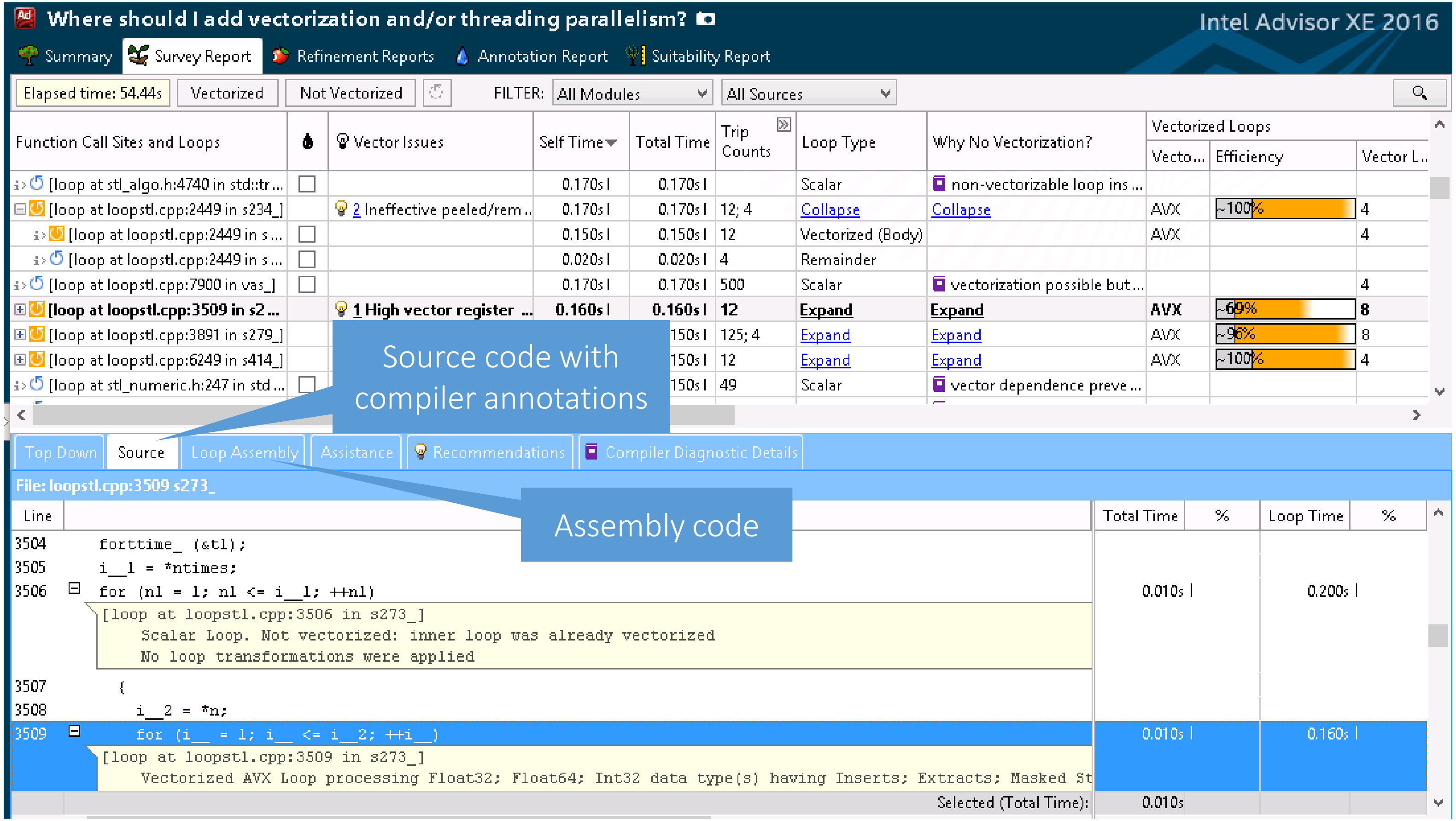
What vectorization issues do I have?

Which Vector instructions are being use?

How efficient is the code?

Function Call Sites and Loops	💧	Vector Issues	Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	Vectorized Loops		
								Vecto...	Efficiency	Vector L...
[loop at stl_algo.h:4740 in std::tr...	☐		0.170s	0.170s		Scalar	non-vectorizable loop ins...			
[loop at lo pstl.cpp:2449 in s234_]		2 Ineffective peeled/rem...	0.170s	0.170s	12; 4	Collapse	Collapse	AVX	~100%	4
[loop at opstl.cpp:2449 in s...	☐		0.150s	0.150s	12	Vectorized (Body)		AVX		4
[loop at opstl.cpp:2449 in s...	☐		0.020s	0.020s	4	Remainder				
[loop at loo tl.cpp:7900 in vas_]	☐		0.170s	0.170s	500	Scalar	vectorization possible but...			4
[loop at loo l.cpp:3509 in s2...		**1 High vector register ...**	**0.160s**	**0.160s**	**12**	**Expand**	**Expand**	**AVX**	**~69%**	**8**
[loop at loop cpp:3891 in s279_]		2 Ineffective peeled/rem...	0.150s	0.150s	125; 4	Expand	Expand	AVX	~96%	8
[loop at loop pp:6249 in s414_]			0.150s	0.150s	12	Expand	Expand	AVX	~100%	4
[loop at stl_r ric.h:247 in std...	☐	1 Assumed dependency...	0.150s	0.150s	49	Scalar	vector dependen preve...			

Source code and assembly integration



Where should I add vectorization and/or threading parallelism?

Intel Advisor XE 2016

Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	Vectorized Loops		
								Vecto...	Efficiency	Vector L...
[loop at stl_algo.h:4740 in std::tr ...	□		0.170s	0.170s		Scalar	non-vectorizable loop ins ...			
[loop at loopstl.cpp:2449 in s234_]		2 Ineffective peeled/rem ...	0.170s	0.170s	12; 4	Collapse	Collapse	AVX	~100%	4
[loop at loopstl.cpp:2449 in s ...	□		0.150s	0.150s	12	Vectorized (Body)		AVX		4
[loop at loopstl.cpp:2449 in s ...	□		0.020s	0.020s	4	Remainder				
[loop at loopstl.cpp:7900 in vas_]	□		0.170s	0.170s	500	Scalar	vectorization possible but ...			4
[loop at loopstl.cpp:3509 in s2 ...		**1 High vector register ...**	**0.160s**	**0.160s**	**12**	**Expand**	**Expand**	**AVX**	**~69%**	**8**
[loop at loopstl.cpp:3891 in s279_]				150s	125; 4	Expand	Expand	AVX	~96%	8
[loop at loopstl.cpp:6249 in s414_]				150s	12	Expand	Expand	AVX	~100%	4
[loop at stl_numeric.h:247 in std ...	□			150s	49	Scalar	vector dependence preve ...			

Source code with compiler annotations

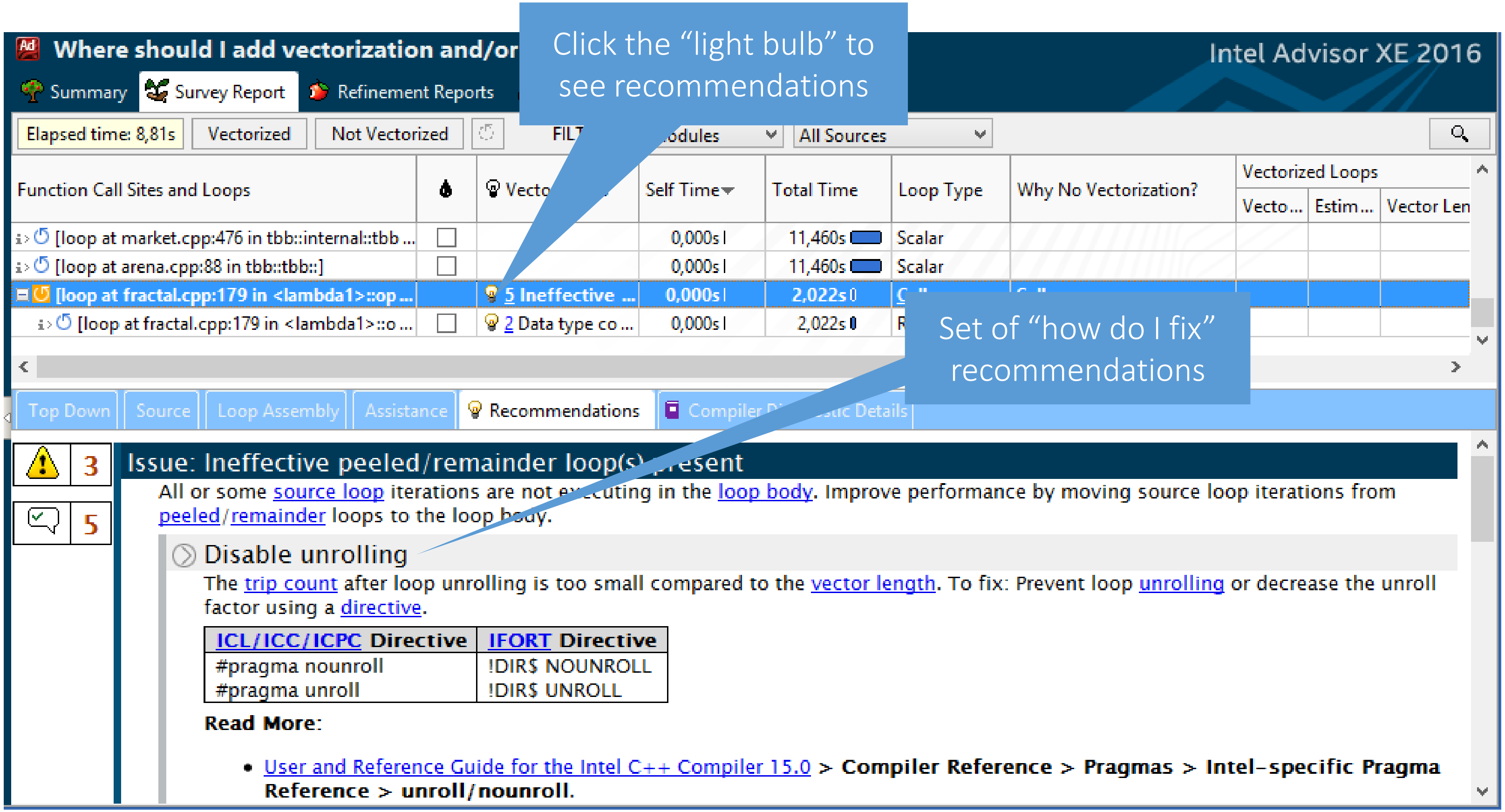
File: loopstl.cpp:3509 s273_

Assembly code

```
Line                                                             Total Time   %   Loop Time   %
3504      forttime_ (&tl);
3505      i__l = *ntimes;
3506  ⊟   for (nl = 1; nl <= i__l; ++nl)                            0.010s          0.200s
       [loop at loopstl.cpp:3506 in s273_]
          Scalar Loop. Not vectorized: inner loop was already vectorized
          No loop transformations were applied
3507        {
3508          i__2 = *n;
3509  ⊟       for (i__ = 1; i__ <= i__2; ++i__)                     0.010s          0.160s
       [loop at loopstl.cpp:3509 in s273_]
          Vectorized AVX Loop processing Float32; Float64; Int32 data type(s) having Inserts; Extracts; Masked St
                                               Selected (Total Time):  0.010s
```

Recommendations

Get specific advice for improving vectorization



Where should I add vectorization and/or

Intel Advisor XE 2016

Function Call Sites and Loops		Vecto	Self Time	Total Time	Loop Type	Why No Vectorization?	Vectorized Loops		
							Vecto...	Estim...	Vector Len
[loop at market.cpp:476 in tbb::internal::tbb ...	□		0,000s	11,460s	Scalar				
[loop at arena.cpp:88 in tbb::tbb::]	□		0,000s	11,460s	Scalar				
[loop at fractal.cpp:179 in <lambda1>::op ...		5 Ineffective ...	0,000s	2,022s	C	C			
[loop at fractal.cpp:179 in <lambda1>::o ...	□	2 Data type co ...	0,000s	2,022s	R				

Issue: Ineffective peeled/remainder loop(s) present

All or some source loop iterations are not executing in the loop body. Improve performance by moving source loop iterations from peeled/remainder loops to the loop body.

Disable unrolling

The trip count after loop unrolling is too small compared to the vector length. To fix: Prevent loop unrolling or decrease the unroll factor using a directive.

ICL/ICC/ICPC Directive	IFORT Directive
#pragma nounroll #pragma unroll	!DIR$ NOUNROLL !DIR$ UNROLL

Read More:

- User and Reference Guide for the Intel C++ Compiler 15.0 > **Compiler Reference > Pragmas > Intel-specific Pragma Reference > unroll/nounroll.**

Click the “light bulb” to see recommendations

Set of “how do I fix” recommendations

Intel® VTune^{TM} Amplifier XE 2016

Performance profiler for serial/parallel programs

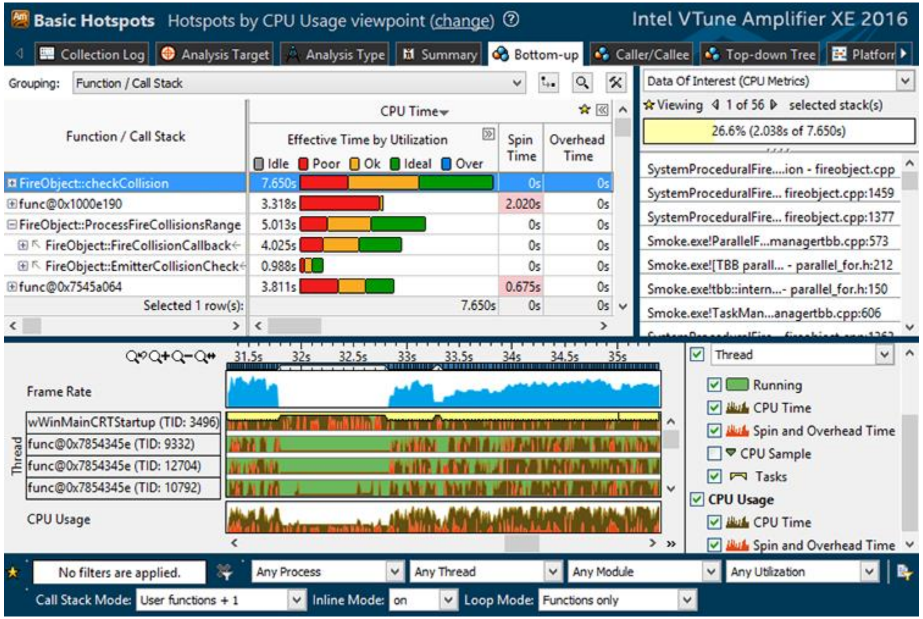
- GUI and command-line interfaces

Event collection/instrumentation

- No special recompiles
- Local/remote event collection
- Low overhead

Analysis features

- Quickly locate hotspots
- Identify issues in source code
- Threading analysis
- Visualize thread behaviour
- Find uarch bottlenecks



Grouping: Function / Call Stack

Function / Call Stack	CPU Time: Effective Time by Utilization (🟫 Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over)	Spin Time	Overhead Time
⊞ FireObject::checkCollision	7.650s	0s	0s
⊞ func@0x1000e190	3.318s	2.020s	0s
⊟ FireObject::ProcessFireCollisionsRange	5.013s	0s	0s
⊞ FireObject::FireCollisionCallback	4.025s	0s	0s
⊞ FireObject::EmitterCollisionCheck	0.988s	0s	0s
⊞ func@0x7545a064	3.811s	0.675s	0s
Selected 1 row(s):	7.650s	0s	0s

Data Of Interest (CPU Metrics)

Viewing 1 of 56 selected stack(s)

26.6% (2.038s of 7.650s)

SystemProceduralFire....ion - fireobject.cpp
SystemProceduralFire... fireobject.cpp:1459
SystemProceduralFire... fireobject.cpp:1377
Smoke.exe!ParallelF...managertbb.cpp:573
Smoke.exe![TBB parall... - parallel_for.h:212
Smoke.exe!tbb::intern...- parallel_for.h:150
Smoke.exe!TaskMan...anagertbb.cpp:606

wWinMainCRTStartup (TID: 3496)
func@0x7854345e (TID: 9332)
func@0x7854345e (TID: 12704)
func@0x7854345e (TID: 10792)

☑ Thread
☑ Running
☑ CPU Time
☑ Spin and Overhead Time
☐ CPU Sample
☑ Tasks
☑ CPU Usage
☑ CPU Time
☑ Spin and Overhead Time

No filters are applied. | Any Process | Any Thread | Any Module | Any Utilization

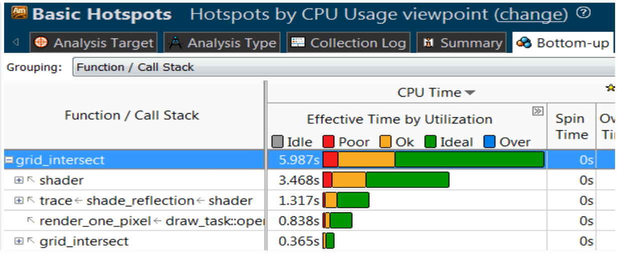
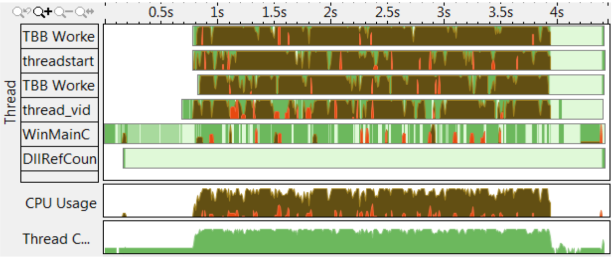
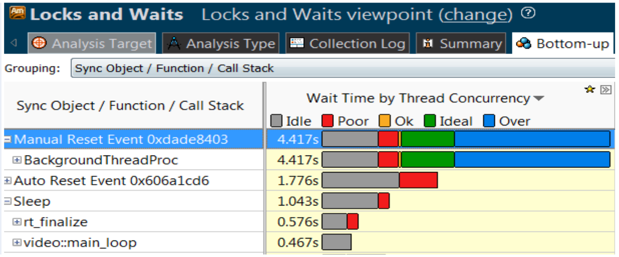
Call Stack Mode: User functions + 1 | Inline Mode: on | Loop Mode: Functions only

Check release notes and “What’s new in 2016?” for product updates

Intel® VTune^{TM} Amplifier XE analysis types

Software user mode sampling and tracing collector

Any processor, any virtual, no driver (about 5% overhead)

Analysis	Description	Sample
Basic hotspots	This analysis helps **understand application flow** and **identify sections of code that get a lot of execution time (hotspots)**. It also captures the call stacks for each of these functions so you can see how the hot functions are called.	
Concurrency	This analysis helps **identify hotspot functions where processor utilization is poor**, by providing information on how many threads were running (not waiting at a defined waiting or blocking API) at each moment during application execution. When cores are idle at a hotspot, you have an opportunity to improve performance by getting those cores working for you.	
Locks and waits	This analysis helps **identify the cause of ineffective processor utilization**. One of the most common problems is threads waiting too long on synchronization objects (locks). With this analysis you can **estimate the impact each synchronization object** has on the application and understand how long the application had to wait on each synchronization object, or in blocking APIs, such as sleep and blocking I/O.	

Basic Hotspots Hotspots by CPU Usage viewpoint (change)

Grouping: Function / Call Stack

Function / Call Stack	CPU Time Effective Time by Utilization □ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over	Spin Time	Ov Ti
⊟ grid_intersect	5.987s	0s	
⊞ shader	3.468s	0s	
⊞ trace ← shade_reflection ← shader	1.317s	0s	
render_one_pixel ← draw_task::oper	0.838s	0s	
⊞ grid_intersect	0.365s	0s	

Locks and Waits Locks and Waits viewpoint (change)

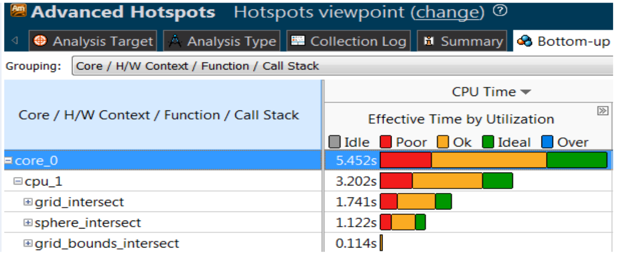
Grouping: Sync Object / Function / Call Stack

Sync Object / Function / Call Stack	Wait Time by Thread Concurrency □ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over
⊟ Manual Reset Event 0xdade8403	4.417s
⊞ BackgroundThreadProc	4.417s
⊞ Auto Reset Event 0x606a1cd6	1.776s
⊟ Sleep	1.043s
⊞ rt_finalize	0.576s
⊞ video::main_loop	0.467s

Intel® VTune^{TM} Amplifier XE analysis types

Hardware Event-Based Sampling (EBS)

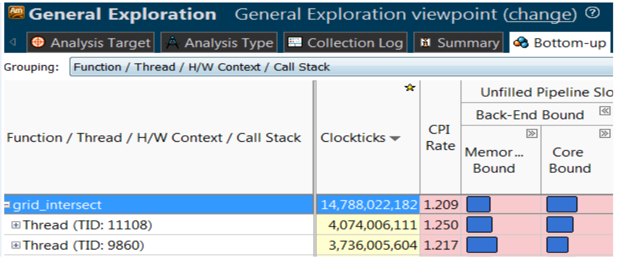
Higher resolution, system wide, lower overhead (about 2% overhead)

<table>
<tr><th>Analysis</th><th>Description</th><th>Sample</th></tr>
<tr>
<td>Advanced hotspots</td>
<td>This analysis is a fast and <b>easy way to identify performance-critical code sections (hotspots)</b>. By default, it does not capture the function call stacks as the hotspots are collected, but it can be used to sample all processes on the system.</td>
<td>


Advanced Hotspots Hotspots viewpoint (change)

Grouping: Core / H/W Context / Function / Call Stack

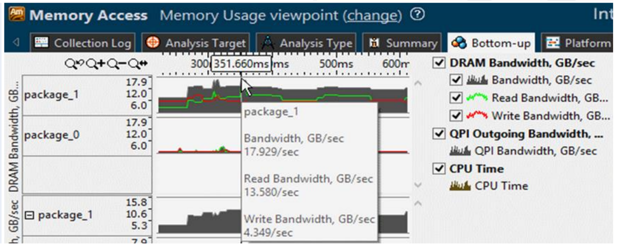
<table>
<tr><th rowspan="3">Core / H/W Context / Function / Call Stack</th><th>CPU Time</th></tr>
<tr><th>Effective Time by Utilization</th></tr>
<tr><th>⬛ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over</th></tr>
<tr><td>⊟ core_0</td><td>5.452s</td></tr>
<tr><td>⊟ cpu_1</td><td>3.202s</td></tr>
<tr><td>⊞ grid_intersect</td><td>1.741s</td></tr>
<tr><td>⊞ sphere_intersect</td><td>1.122s</td></tr>
<tr><td>⊞ grid_bounds_intersect</td><td>0.114s</td></tr>
</table>

</td>
</tr>
<tr>
<td>General exploration</td>
<td>This analysis is a good starting point to <b>triage hardware issues in your application</b> by understanding how efficiently your code is passing through the core pipeline. It <b>calculates a set of predefined ratios used for the metrics and facilitates identifying hardware-level performance problems</b>. The list of events and metrics collected during the General Exploration analysis depends on your microarchitecture.</td>
<td>


General Exploration General Exploration viewpoint (change)

Grouping: Function / Thread / H/W Context / Call Stack

<table>
<tr><th rowspan="3">Function / Thread / H/W Context / Call Stack</th><th rowspan="3">Clockticks</th><th rowspan="3">CPI Rate</th><th colspan="2">Unfilled Pipeline Slo</th></tr>
<tr><th colspan="2">Back-End Bound</th></tr>
<tr><th>Memor... Bound</th><th>Core Bound</th></tr>
<tr><td>⊟ grid_intersect</td><td>14,788,022,182</td><td>1.209</td><td></td><td></td></tr>
<tr><td>⊞ Thread (TID: 11108)</td><td>4,074,006,111</td><td>1.250</td><td></td><td></td></tr>
<tr><td>⊞ Thread (TID: 9860)</td><td>3,736,005,604</td><td>1.217</td><td></td><td></td></tr>
</table>

</td>
</tr>
<tr>
<td>Memory access
<i>New!</i></td>
<td>Use this analysis to <b>identify memory-related issues</b>, like NUMA problems and bandwidth-limited accesses, and attribute performance events to memory objects (data structures). This analysis replaces the “Bandwidth analysis” present in previous versions of the tool.</td>
<td>


Memory Access Memory Usage viewpoint (change)

package_1

Bandwidth, GB/sec
17.929/sec

Read Bandwidth, GB/sec
13.580/sec

Write Bandwidth, GB/sec
4.349/sec

☑ DRAM Bandwidth, GB/sec
☑ Bandwidth, GB/sec
☑ Read Bandwidth, GB...
☑ Write Bandwidth, GB...
☑ QPI Outgoing Bandwidth, ...
QPI Bandwidth, GB/sec
☑ CPU Time
CPU Time

</td>
</tr>
</table>

Intel® Parallel Studio XE 2016 components

	Full Licensing (including Intel® Premier Support)			Free Licensing			
Component	Composer Edition	Professional Edition	Cluster Edition	Student/ Educator	Open Source Contributor	Academic Researcher	Community (Everyone!)
Intel® C/C++ Compiler (including Intel® Cilk™ Plus)	✓	✓	✓	✓	✓		
Intel® Fortran Compiler	✓	✓	✓	✓	✓		
OpenMP 4.0	✓	✓	✓	✓	✓		
Intel® Threading Building Blocks (C++ only)	✓	✓	✓	✓	✓	✓	✓
Intel® IPP Library (C/C++ only)	✓	✓	✓	✓	✓	✓	✓
Intel® Math Kernel Library	✓	✓	✓	✓	✓	✓	✓
Intel® Data Analytics Acceleration Library	✓	✓	✓	✓	✓	✓	✓
Intel® MPI Library			✓	✓		✓	
Rogue Wave IMSL Library (Fortran only)	Bundled and Add-on	Add-on	Add-on				
Intel® Advisor XE		✓	✓	✓	✓		
Intel® Inspector XE		✓	✓	✓	✓		
Intel® VTune^{TM} Amplifier XE		✓	✓	✓	✓		
Intel® ITAC + MPI Performance Snapshot			✓	✓			

Beta Program Intel® Parallel Studio XE 2017

Compiler 17.0 is part of Intel® Parallel Studio XE 2017

Beta program of IPSXE-2107 started end of March 2016

- To join, please register at : http://bit.ly/psxe2017beta

- More information:
  - Overview beta program : https://software.intel.com/en-us/articles/intel-parallel-studio-xe-2017-beta
  - Release notes page: https://software.intel.com/en-us/articles/intel-parallel-studio-xe-2017-beta-release-notes

Intel® Parallel Studio XE – Improvements

- Annotated Source Listings
  - modified copy of source with line numbers and compiler diagnostics inserted
- Code Alignment for Functions and Loops
- Optimization Reports
  - More precise non-vectorization reasons
  - Significant Improvement in Variable Names and Memory References
- Additional Diagnostic Messages

How to get ready for Intel® AVX-512?

Start optimizing your application today for current generation of Intel® Xeon® processors and Intel® Xeon™ Phi coprocessors

- and/or -

Compile with latest compiler toolchains

- Intel compiler (v15.0+): `-xCOMMON-AVX512`, `-xMIC-AVX512`, `-xCORE-AVX512`
- GNU compiler (v4.9+): `-mavx512f`, `-mavx512cd`, `-mavx512er`, `-mavx512pf`

Tune your AVX-512 kernels on non-existing silicon

- Run your kernels on top of the Intel® Software Development emulator (SDE)
  - Emulate (future) Intel® Architecture Instruction Set Extensions (e.g. Intel® MPX, …)
- Tools available for detailed analysis
  - Instruction type histogram
  - Pointer/misalignment checker
- Also possible to debug the application being emulated

Key new features for software adaptation to KNL

Large impact: **Intel® AVX-512 instruction set**

- 32 512-bit FP/Int vector registers, 8 mask registers, HW gather/scatter
- Slightly different from future Intel® Xeon™ architecture AVX-512 extensions
- Backward compatible with SSE, AVX, AVX-2
- Apps built for HSW and earlier can run on KNL (few exceptions like TSX)
- Incompatible with 1st Generation Intel® Xeon Phi™ (KNC)

Medium impact: **new, on-chip high bandwidth memory (HBM)**

- Creates heterogeneous (NUMA) memory access
- Can be used transparently too however

Minor impact: **differences in floating point execution/rounding**

- New HW-accelerated transcendental functions like exp()

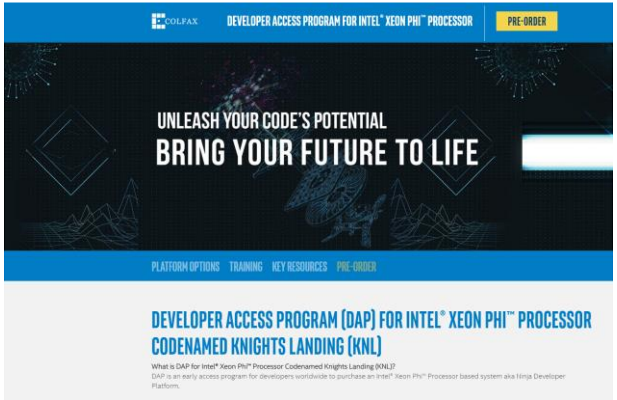
Pre-Order Developer Platform for Intel® Xeon Phi™ Processor *Today!* *Unleash your code's potential*

For Code/Application Developers

- Academic
- Scientific applications
- Physics
- Big data analytics
- Life sciences –Genomics
- Life Sciences -Molecular Dynamics
- Finance
- Oil & Gas
- Manufacturing
- Modeling,
- Simulation
- Visualization

Leading edge platform capabilities, performance to deliver multi-threaded, vectorized software for today's HPC workloads !

http://xeonphideveloper.com



DEVELOPER ACCESS PROGRAM FOR INTEL® XEON PHI™ PROCESSOR

UNLEASH YOUR CODE'S POTENTIAL

BRING YOUR FUTURE TO LIFE

PLATFORM OPTIONS TRAINING KEY RESOURCES PRE-ORDER

DEVELOPER ACCESS PROGRAM (DAP) FOR INTEL® XEON PHI™ PROCESSOR CODENAMED KNIGHTS LANDING (KNL)

What is DAP for Intel® Xeon Phi™ Processor Codenamed Knights Landing (KNL)?

DAP is an early access program for developers worldwide to purchase an Intel® Xeon Phi™ Processor based system aka Ninja Developer Platform.

Highly-Parallel Performance

- Intel® Xeon Phi™ Processor, 512-bit SIMD vectors with 2 VPU/core, 16GB MCDRAM integrated memory,
- Binary-compatible with Intel® Xeon® processors

Software Tools & Libraries

- CentOS 7.2
- Includes Intel Parallel Studio XE 2016 1 year license
- Featuring the new Intel Vector Advisor for parallelization
- Access to Intel Libraries

Support & Training

- Online community access
- Support from Colfax/Local OEMs
- Training: Hands on Webinars, optimization guidance, whitepapers, videos, How to guides

Online resources

Intel® Software Development Products, performance tuning, etc.

• Documentation library	All available documentation about Intel software
• HPC webinars	Free technical webinars about HPC on Intel platforms
• Modern code	Intel resources about code modernization
• Forums	Public discussions about Intel SIMD, threading, ISAs, etc.

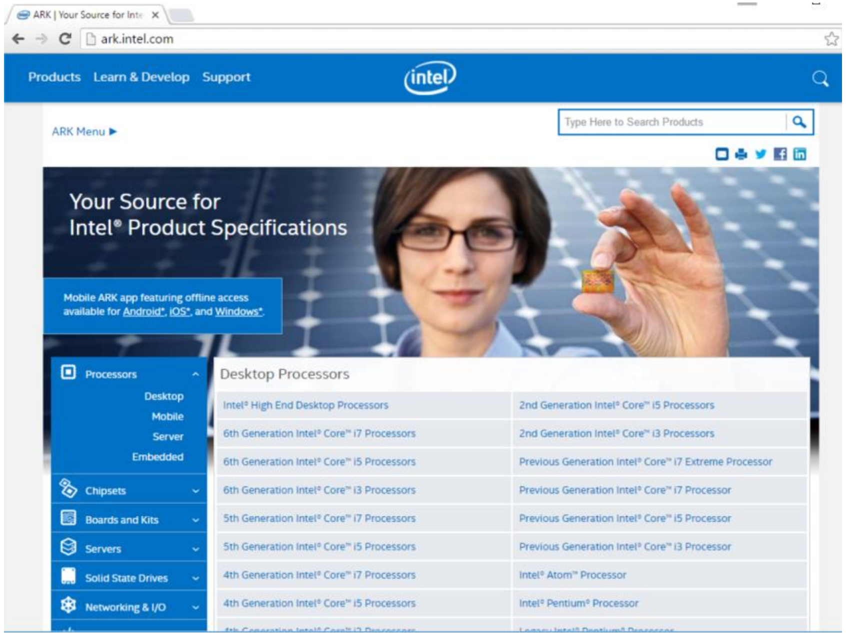
Intel® Xeon Phi$^{\text{TM}}$ resources

• Developer portal	Programming guides, tools, trainings, case studies, etc.
• Solutions catalog	Existing Intel® Xeon Phi$^{\text{TM}}$ solutions for known codes

Other resources (white papers, benchmarks, case studies, etc.)

• Go parallel	BKMs for Intel multi- and many-core architectures
• Colfax research	Publications and material on parallel programming
• Bayncore labs	Research and development activities (WIP)

Online resources



ARK | Your Source for Inte

ark.intel.com

Products Learn & Develop Support

ARK Menu ▶

Type Here to Search Products

Your Source for
Intel® Product Specifications

Mobile ARK app featuring offline access available for Android*, iOS*, and Windows*.

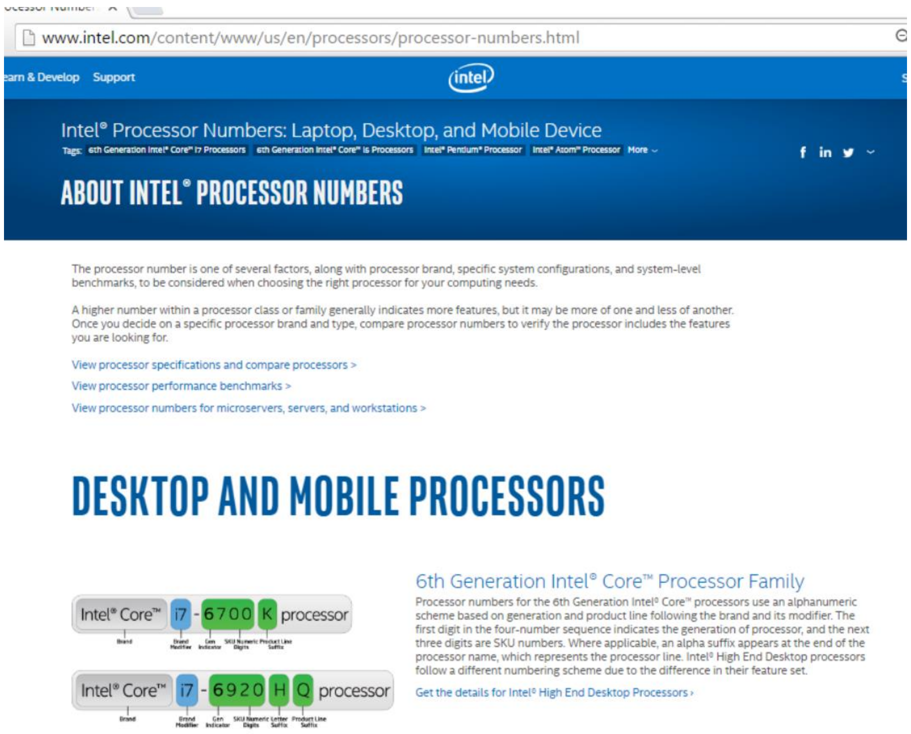
Processors
Desktop
Mobile
Server
Embedded
Chipsets
Boards and Kits
Servers
Solid State Drives
Networking & I/O

Desktop Processors

Intel® High End Desktop Processors	2nd Generation Intel® Core™ i5 Processors
6th Generation Intel® Core™ i7 Processors	2nd Generation Intel® Core™ i3 Processors
6th Generation Intel® Core™ i5 Processors	Previous Generation Intel® Core™ i7 Extreme Processor
6th Generation Intel® Core™ i3 Processors	Previous Generation Intel® Core™ i7 Processor
5th Generation Intel® Core™ i7 Processors	Previous Generation Intel® Core™ i5 Processor
5th Generation Intel® Core™ i5 Processors	Previous Generation Intel® Core™ i3 Processor
4th Generation Intel® Core™ i7 Processors	Intel® Atom™ Processor
4th Generation Intel® Core™ i5 Processors	Intel® Pentium® Processor
4th Generation Intel® Core™ i3 Processors	Legacy Intel® Pentium® Processor

Detailed information about available Intel products
http://ark.intel.com/

Encoding scheme of Intel processor numbers
www.intel.com/products/processor_number



www.intel.com/content/www/us/en/processors/processor-numbers.html

earn & Develop Support

Intel® Processor Numbers: Laptop, Desktop, and Mobile Device

Tags: 6th Generation Intel® Core™ i7 Processors 6th Generation Intel® Core™ i5 Processors Intel® Pentium® Processor Intel® Atom™ Processor More

ABOUT INTEL® PROCESSOR NUMBERS

The processor number is one of several factors, along with processor brand, specific system configurations, and system-level benchmarks, to be considered when choosing the right processor for your computing needs.

A higher number within a processor class or family generally indicates more features, but it may be more of one and less of another. Once you decide on a specific processor brand and type, compare processor numbers to verify the processor includes the features you are looking for.

View processor specifications and compare processors >

View processor performance benchmarks >

View processor numbers for microservers, servers, and workstations >

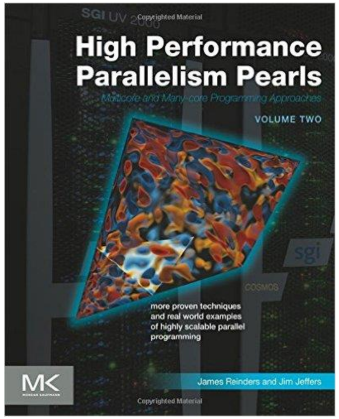
DESKTOP AND MOBILE PROCESSORS

6th Generation Intel® Core™ Processor Family

Processor numbers for the 6th Generation Intel® Core™ processors use an alphanumeric scheme based on generation and product line following the brand and its modifier. The first digit in the four-number sequence indicates the generation of processor, and the next three digits are SKU numbers. Where applicable, an alpha suffix appears at the end of the processor name, which represents the processor line. Intel® High End Desktop processors follow a different numbering scheme due to the difference in their feature set.

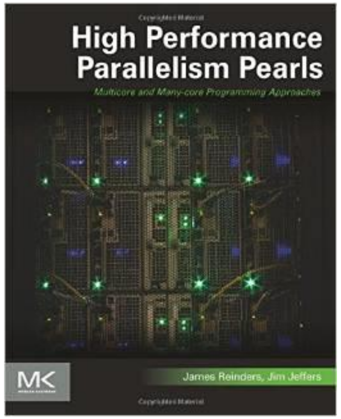
Get the details for Intel® High End Desktop Processors ›

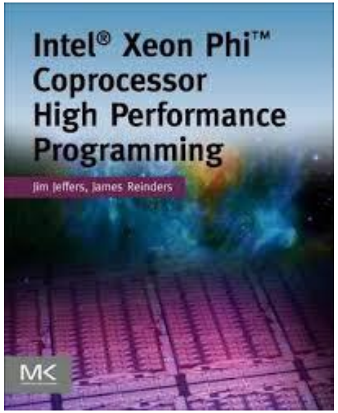
Recommended books


High Performance Parallelism Pearls
Multicore and Many-core Programming Approaches
VOLUME TWO
more proven techniques and real world examples of highly scalable parallel programming
MK
James Reinders and Jim Jeffers

High performance parallelism pearls: multi-core and many-core approaches (Vol. 2), by James Reinders and Jim Jeffers, Morgan Kaufmann, 2015

High performance parallelism pearls: multi-core and many-core approaches, by James Reinders and Jim Jeffers, Morgan Kaufmann, 2014

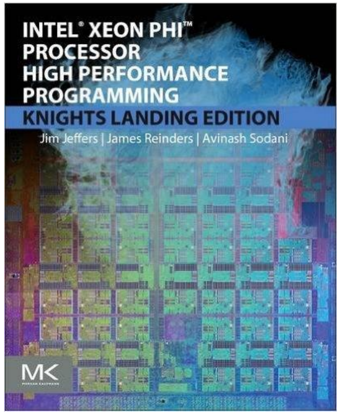

High Performance Parallelism Pearls
Multicore and Many-core Programming Approaches
MK
James Reinders, Jim Jeffers


Intel® Xeon Phi™ Coprocessor High Performance Programming
Jim Jeffers, James Reinders
MK

Intel® Xeon Phi^{TM} Coprocessor High Performance Programming, by Jim Jeffers and James Reinders, Morgan Kaufmann, 2013

Coming up!

Intel® Xeon Phi^{TM} High Performance Programming: Knights Landing Edition 2nd Edition 2nd Edition, 2016


INTEL® XEON PHI™ PROCESSOR HIGH PERFORMANCE PROGRAMMING
KNIGHTS LANDING EDITION
Jim Jeffers | James Reinders | Avinash Sodani
MK