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

<table>
<thead>
<tr>
<th>Multi-core</th>
<th>Many integrated core (MIC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++/Fortran, OMP/MPI/Cilk+/TBB</td>
<td>C/C++/Fortran, OMP/MPI/Cilk+/TBB</td>
</tr>
<tr>
<td>Bootable, native execution model</td>
<td>PCIe coprocessor, native and offload execution models</td>
</tr>
<tr>
<td>Up to 18 cores, 3 GHz, 36 threads ... until slide 16</td>
<td>Up to 61 cores, 1.2 GHz, 244 threads</td>
</tr>
<tr>
<td>Up to 768 GB, 68 GB/s, 432 GFLOP/s DP</td>
<td>Up to 16 GB, 352 GB/s, 1.2 TFLOP/s DP</td>
</tr>
<tr>
<td><strong>256-bit SIMD</strong>, FMA, gather (AVX2)</td>
<td><strong>512-bit SIMD</strong>, FMA, gather/scatter, EMU (IMCI)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Targeted at general purpose applications</strong></th>
<th><strong>Targeted at highly parallel applications</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Single thread performance (ILP)</td>
<td>High parallelism (DLP, TLP)</td>
</tr>
<tr>
<td>Memory capacity</td>
<td>High memory bandwidth</td>
</tr>
</tbody>
</table>
Intel® Xeon® processor architecture
Intel® Xeon® processors and platforms

Intel® Xeon® E3
- Memory
- PCIe3

Intel® Xeon® E5
- Memory
- 2x QPI
- PCIe3

Intel® Xeon® E7
- Memory
- 3x QPI
- PCIe3

Intel® Xeon® E5-1xxx
- E3-1xxx
- CPU/Socket

Intel® Xeon® E5-2xxx
- E5-4xxx
- QPI

Intel® Xeon® E5-2600 v3 (Haswell EP)
Intel® Xeon® E5-2600 v4 (Broadwell EP)

Intel® Xeon® E7-4xxx

Intel® Xeon® E7-xxxx

>4S
Haswell execution unit overview

Unified Reservation Station

- Port 0: Integer ALU & Shift, FMA FP Multiply, Vector Int Multiply, Vector Logicals, Branch
- Port 1: Integer ALU & LEA, FMA FP Multi FP Add, Vector Int ALU, Vector Logicals, Divide, Vector Shifts
- Port 2: Load & Store Address, Store Data
- Port 3: Integer ALU & LEA
- Port 4: Integer ALU & LEA
- Port 5: Integer ALU & Shift, Vector Shuffle, Vector Int ALU, Vector Logicals, Branch
- Port 6: Integer ALU & Shift, 4th ALU
- Port 7: Store Address, Additional AGU for Stores

Additional Branch Unit:
- Reduces Port0 Conflicts
- 2nd EU for high branch code

2xFMA:
- Doubles peak FLOPs
- Two FP multiplies
- Benefits legacy

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

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
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)
## 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

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

◊ 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)
High Performance Computing Performance

Intel® Xeon® E5-2699 v4 (22-core 2.2GHz) vs. Intel® Xeon® E5-2699 v3 (18-core 2.3GHz)

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

From Intel® Xeon® Processor E5-2600 v4 Product Family (“Broadwell”) Performance & Platform Solutions
Public 31 March 2016
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

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

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
Per-Core AVX Max Turbo Optimization on Intel® Xeon® processor E5-2600 v4 Product Family

Non-AVX Codes | AVX Workloads | Mixed Workloads
--- | --- | ---
(Non-AVX and AVX) | (AVX and AVX) | (Mixed AVX and non-AVX)

**Max Turbo Frequency**

- **Non-AVX workloads running on all cores have a max turbo frequency of $P_0n$.**
- **AVX workloads have a lower max turbo frequency of "$P_0n$ AVX".**
- **On E5-2600 v3 family, workloads with a mix of cores running AVX and non-AVX experienced lower max turbo frequency on all cores.**
- **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.**

From Intel® Xeon® Processor E5-2600 v4 Product Family ("Broadwell") Performance & Platform Solutions Public 31 March 2016
Intel® Xeon Phi™ (co)processor architecture

Intel® Many Integrated Core architecture (Intel® MIC)
# Intel® Xeon Phi™ architecture family

<table>
<thead>
<tr>
<th>Intel® Xeon Phi™ coprocessor product family</th>
<th>Upcoming generation of the Intel® MIC architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Knights Corner”</td>
<td>“Knights Hill”</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel® Xeon Phi™ coprocessor product family</td>
<td>Upcoming generation of the Intel® MIC architecture</td>
</tr>
<tr>
<td>“Knights Landing”</td>
<td>“Knights Hill”</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td>2017?</td>
</tr>
<tr>
<td>22 nm process</td>
<td>14 nm process</td>
</tr>
<tr>
<td></td>
<td>10 nm process</td>
</tr>
<tr>
<td>1 TeraFLOP DP peak</td>
<td>3+ TeraFLOP DP peak</td>
</tr>
<tr>
<td></td>
<td>?</td>
</tr>
<tr>
<td>57-61 cores</td>
<td>72 cores (36 tiles)</td>
</tr>
<tr>
<td>In-order core architecture</td>
<td>Out-of-order architecture based on Intel® Atom™ core</td>
</tr>
<tr>
<td>1 Vector Unit per core</td>
<td>2 Vector Units per core</td>
</tr>
<tr>
<td></td>
<td>Up to 3x single thread performance w.r.t. Knights Corner</td>
</tr>
<tr>
<td>6-16 GB GDDR5 memory</td>
<td>On package, 8-16 GB high bandwidth memory (HBM)</td>
</tr>
<tr>
<td></td>
<td>with flexible models: cache, flat, hybrid</td>
</tr>
<tr>
<td></td>
<td>Up to 768 GB DDR4 main memory</td>
</tr>
<tr>
<td>Intel® Initial Many Core Instructions (IMIC)</td>
<td>Intel® Advanced Vector Extensions (AVX-512)</td>
</tr>
<tr>
<td></td>
<td>Binary compatible with AVX2</td>
</tr>
<tr>
<td>PCIe coprocessor</td>
<td>Stand alone processor and PCIe coprocessor versions</td>
</tr>
<tr>
<td>Intel® True Scale fabric</td>
<td>Intel® Omni-Path™ fabric (integrated in some models)</td>
</tr>
<tr>
<td></td>
<td>2nd generation Intel® Omni-Path™ fabric</td>
</tr>
</tbody>
</table>
## Intel® Xeon Phi™ coprocessor product lineup

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

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

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™ uncore architecture

- 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: 2\textsuperscript{nd} 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
| High-performance on-package memory (MCDRAM) | Over 5x STREAM vs. DDR4 (Over \(\approx\)400 GB/s vs \(\approx\)90 GB/s)
| Over 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)
- 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) \(\rightarrow\) 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
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

$\$/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
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

<table>
<thead>
<tr>
<th>KNL</th>
<th>KNL w/ Omni-Path</th>
<th>KNL Card</th>
</tr>
</thead>
</table>
| 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

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

Projected KNL Performance (1 socket, 200W CPU TDP) vs. 2 Socket Intel® Xeon® processor E5-2697v3 (2x145W CPU TDP)
Best practices for SIMD vectorization
Exploiting the parallel universe
Three levels of parallelism supported by Intel hardware

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

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 \textit{vectorized}
• Vectorization usually on \textit{inner} loops
• Main and \textit{remainder} loops are generated

\begin{verbatim}
for (int i = 0; i < N; i++)
    c[i] = a[i] + b[i];

for (int i = 0; i < N; i += 4)
    c[i:4] = a[i:4] + b[i:4];
\end{verbatim}

Scalar loop
\begin{tabular}{cccc}
    x & y & z & w \\
    1.0 & 2.0 & 3.0 & 4.0 \\
\end{tabular}

SIMD loop (4 elements)
\begin{tabular}{cccc}
    1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 7.0 & 8.0 & 6.0 & 8.0 & 10.0 & 12.0 \\
\end{tabular}
Past, present, and future of Intel SIMD types

- Current Intel® Xeon® processors
  - Streaming SIMD Extensions (SSE*)
  - MultiMedia eXtensions (MMX)
  - Advanced Vector eXtensions (AVX)
  - AVX2
  - AVX-512
  - 64-bit SIMD
  - 128-bit SIMD
  - 256-bit SIMD
  - 512-bit SIMD

- Future Intel® Xeon Phi™ coprocessors (including Knights Landing)
  - 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® processors
  - Initial Many Core Instructions (IMCI)

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

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

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];

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

**Auto Vectorization**
- Libraries
- Compiler knobs

**Guided Vectorization**
- Compiler hints/pragmas
- Array notation

**Low level Vectorization**
- C/C++ vector classes
- Intrinsics/Assembly

Easy of use
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](https://software.intel.com/en-us/mkl-community) (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
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

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)

<table>
<thead>
<tr>
<th>Opt. level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>–O0</td>
<td>Disables all optimizations.</td>
</tr>
<tr>
<td>–O1</td>
<td>Enables optimizations for speed which are known to not cause code size increase.</td>
</tr>
</tbody>
</table>
| –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

Vectorization won’t be efficient
  • Compiler estimates how better the vectorized version will be
  • Affected by data alignment, data layout, etc.

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
Auto vectorization on Intel compilers

Vectorization breakdown for loop candidates in Polyhedron benchmark suite

<table>
<thead>
<tr>
<th>Percentage of all candidate loops at compile time</th>
<th>Speedup vs. non vectorized version (higher is better)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel2</td>
<td>100x</td>
</tr>
<tr>
<td>fatigue2</td>
<td>90%</td>
</tr>
<tr>
<td>protein</td>
<td>80%</td>
</tr>
<tr>
<td>aermod</td>
<td>70%</td>
</tr>
<tr>
<td>mp_prop_design</td>
<td>60%</td>
</tr>
<tr>
<td>doduc</td>
<td>50%</td>
</tr>
<tr>
<td>tft2</td>
<td>40%</td>
</tr>
<tr>
<td>capacita</td>
<td>30%</td>
</tr>
<tr>
<td>ac</td>
<td>20%</td>
</tr>
<tr>
<td>nf</td>
<td>10%</td>
</tr>
<tr>
<td>air</td>
<td>0%</td>
</tr>
<tr>
<td>nflow</td>
<td>100%</td>
</tr>
<tr>
<td>mbx</td>
<td>90%</td>
</tr>
<tr>
<td>linpk</td>
<td>80%</td>
</tr>
<tr>
<td>test_fpu2</td>
<td>70%</td>
</tr>
<tr>
<td>induct2</td>
<td>60%</td>
</tr>
<tr>
<td>gas_dyn2</td>
<td>50%</td>
</tr>
</tbody>
</table>

Vectorized loops (including memset/memcpy)
Non-standard, non-canonical, or too complex loop
Outer loop not vectorizable (inner loop already was)
Vectorization possible but seems inefficient
Vector dependence prevents vectorization
Other

Polyhedron benchmark suite
Intel® Xeon Phi™ 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 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
```

```
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
```

Vectorized loop

Non-vectorized loop

Loop nest optimizations
- `loop`
- `par`
- **vec** Vectorization
- `openmp`
- `offload`
- `ipo`
- `pgo`
- `cg`
- `tcollect`
- `all` All optimizations (default)
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`

Ignore assumed vector dependencies with Intel-specific compiler directive
- **C/C++**: `#pragma ivdep`
- **Fortran**: `!dir$ ivdep`

```c
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];
}
```

```c/
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?

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-mmic</td>
<td>Builds an application that runs natively on Intel® MIC Architecture.</td>
</tr>
<tr>
<td>-xfeature -xHost</td>
<td>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™ architecture). Values for <code>feature</code> are:</td>
</tr>
<tr>
<td></td>
<td>• COMMON-AVX512 (includes AVX512 FI and CDI instructions)</td>
</tr>
<tr>
<td></td>
<td>• MIC-AVX512 (includes AVX512 FI, CDI, PFI, and ERI instructions)</td>
</tr>
<tr>
<td></td>
<td>• CORE-AVX512 (includes AVX512 FI, CDI, BWI, DQI, and VLE instructions)</td>
</tr>
<tr>
<td></td>
<td>• CORE-AVX2</td>
</tr>
<tr>
<td></td>
<td>• CORE-AVX-I (including RDRND instruction)</td>
</tr>
<tr>
<td></td>
<td>• AVX</td>
</tr>
<tr>
<td></td>
<td>• SSE4.2, SSE4.1</td>
</tr>
<tr>
<td></td>
<td>• ATOM_SSE4.2, ATOM_SSSE3 (including MOVBE instruction)</td>
</tr>
<tr>
<td></td>
<td>• SSSE3, SSE3, SSE2</td>
</tr>
<tr>
<td></td>
<td>When using <code>-xHost</code>, the compiler will generate instructions for the highest instruction set available on the compilation host processor.</td>
</tr>
</tbody>
</table>

| -axfeature | 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**

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

```c
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**

```c
__declspec(vector)
void vadd(float c, float a, float b) {
    c = a + b;
}
...```

**SIMD function**

Fortran

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

Layout your data as Structure of Arrays (SoA)
- As opposite to Array of Structures (AoS)

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
Improving vectorization: data alignment (cont’d)

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

Padding might be necessary to guarantee aligned access to matrices

$n=64$ for Intel® Xeon Phi™ coprocessors, $n=32$ for AVX, $n=16$ for SSE
Vectorization with multi-version loops

Peel loop
Alignment purposes
Might be vectorized

Main loop
Vectorized
Unrolled by x2 or x4

Remainder loop
Remainder iterations
Might be vectorized

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 #15487: --- 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™ 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
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
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

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?
Source code and assembly integration

Source code with compiler annotations

Assembly code
Recommendations
Get specific advice for improving vectorization

Click the “light bulb” to see recommendations

Set of “how do I fix” recommendations
Intel® VTune™ 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

Check release notes and “What’s new in 2016?” for product updates
## Intel® VTune™ Amplifier XE analysis types

**Software user mode sampling and tracing collector**

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

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Description</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic hotspots</strong></td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td><strong>Concurrency</strong></td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td><strong>Locks and waits</strong></td>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>
## Intel® VTune™ Amplifier XE analysis types

### Hardware Event-Based Sampling (EBS)
Higher resolution, system wide, lower overhead (about 2% overhead)

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Description</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Advanced hotspots</strong></td>
<td>This analysis is a fast and <strong>easy way to identify performance-critical code sections (hotspots)</strong>. 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></td>
</tr>
<tr>
<td><strong>General exploration</strong></td>
<td>This analysis is a good starting point to <strong>triage hardware issues in your application</strong> by understanding how efficiently your code is passing through the core pipeline. It calculates a set of predefined ratios used for the metrics and facilitates identifying hardware-level performance problems. The list of events and metrics collected during the General Exploration analysis depends on your microarchitecture.</td>
<td></td>
</tr>
<tr>
<td><strong>Memory access</strong></td>
<td>Use this analysis to <strong>identify memory-related issues</strong>, 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></td>
</tr>
</tbody>
</table>
## Intel® Parallel Studio XE 2016 components

<table>
<thead>
<tr>
<th>Component</th>
<th>Full Licensing (including Intel® Premier Support)</th>
<th>Free Licensing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® C/C++ Compiler</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(including Intel® Cilk™ Plus)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel® Fortran Compiler</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OpenMP 4.0</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intel® Threading Building Blocks (C++ only)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intel® IPP Library (C/C++ only)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intel® Math Kernel Library</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intel® Data Analytics Acceleration Library</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intel® MPI Library</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Rogue Wave IMSL Library (Fortran only)</td>
<td>Bundled and Add-on</td>
<td>Add-on</td>
</tr>
<tr>
<td>Intel® Advisor XE</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intel® Inspector XE</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intel® VTune™ Amplifier XE</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intel® ITAC + MPI Performance Snapshot</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
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:
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

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™ resources

- **Developer portal** Programming guides, tools, trainings, case studies, etc.
- **Solutions catalog** Existing Intel® Xeon Phi™ 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

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

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


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

Coming up!