# HPC codes **modernization** using **vector** and threading **parallelism – part I (intro)**

**Zakhar A. Matveev, PhD,**

**Intel Russia, Intel Software and Services Group**

**May' 2015**

# Acknowledgments

**This foildeck uses some content created by:**

- Kevin O'Leary, Dick Kaiser, Mike Lee (from Parallel Studio and Vectorization Advisor marketing and support teams)
- Geoff Lowney and Victor Lee (SIMD conference keynotes)
- James Reinders and Arch D. Robison
- Intel® Compiler architects

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Motivation

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Performance is a Proven Game Changer

## It is driving disruptive change in multiple industries

### Protecting buildings from extreme events

Sophisticated mechanics simulations are performed to identify innovative ways to protect infrastructure from extreme events, such as natural disasters.

### Solving Austin, Texas's traffic problem

Running advanced traffic simulations to improve the models used to plan infrastructure and traffic control changes

### New possible treatments for Parkinson's

Extensive calculations performed at supercomputer helped researchers to learn more about the protein structure's evolution

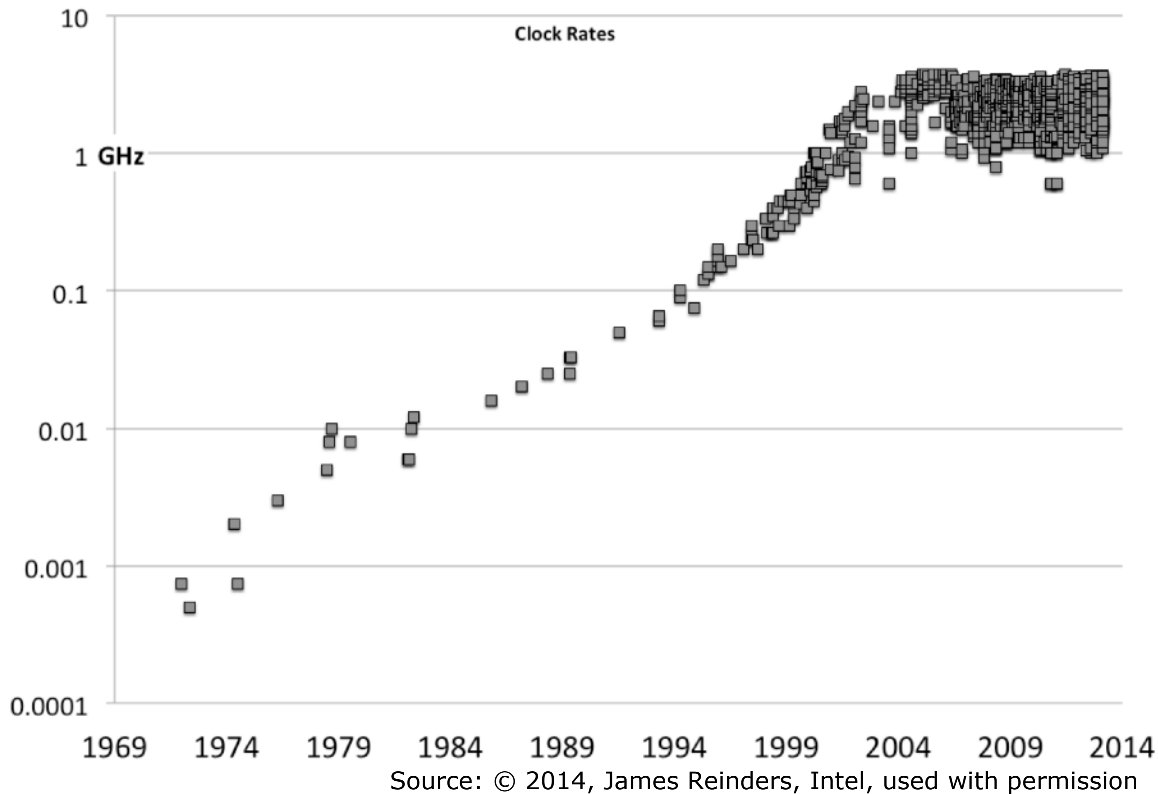
Click on a picture for details

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Optimization Notice

# The “Free Lunch” is over, really

## Processor clock rate growth halted around 2005



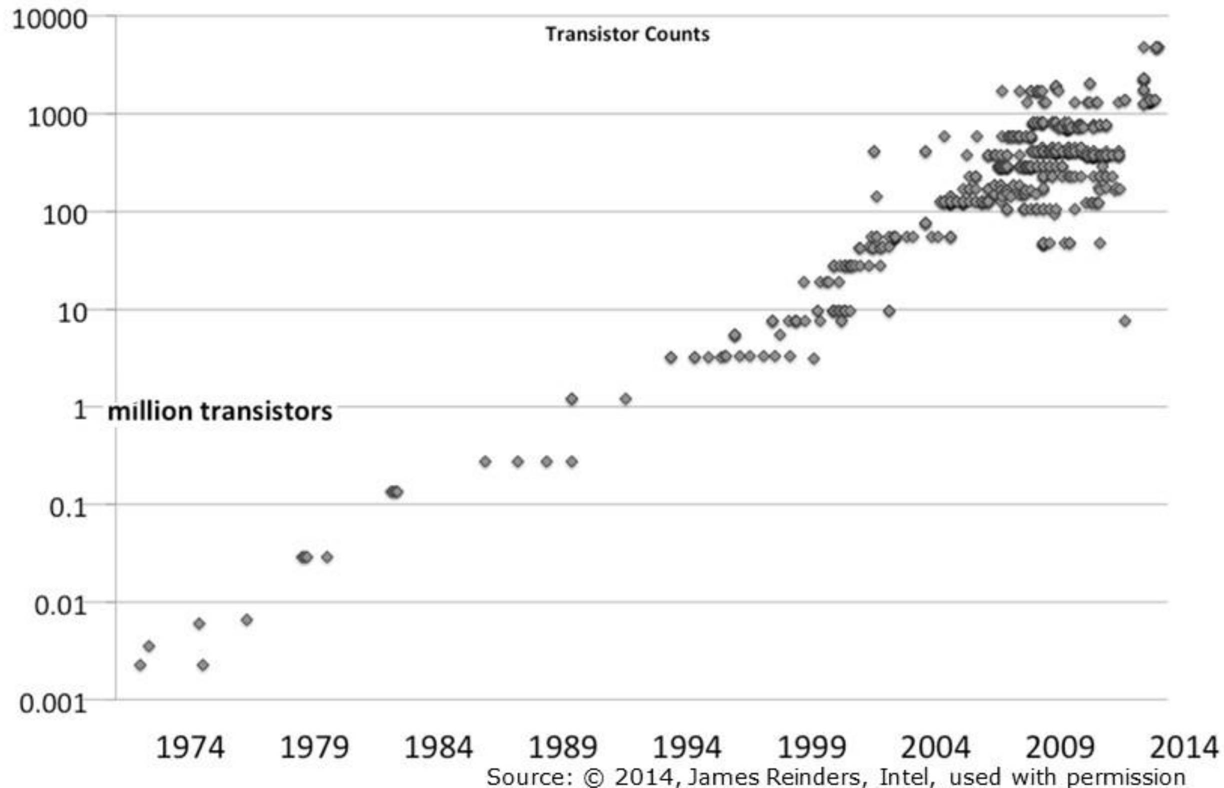

Source: © 2014, James Reinders, Intel, used with permission

Software must be parallelized to realize all the potential performance

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Moore's Law Is Going Strong

## Hardware performance continues to grow exponentially

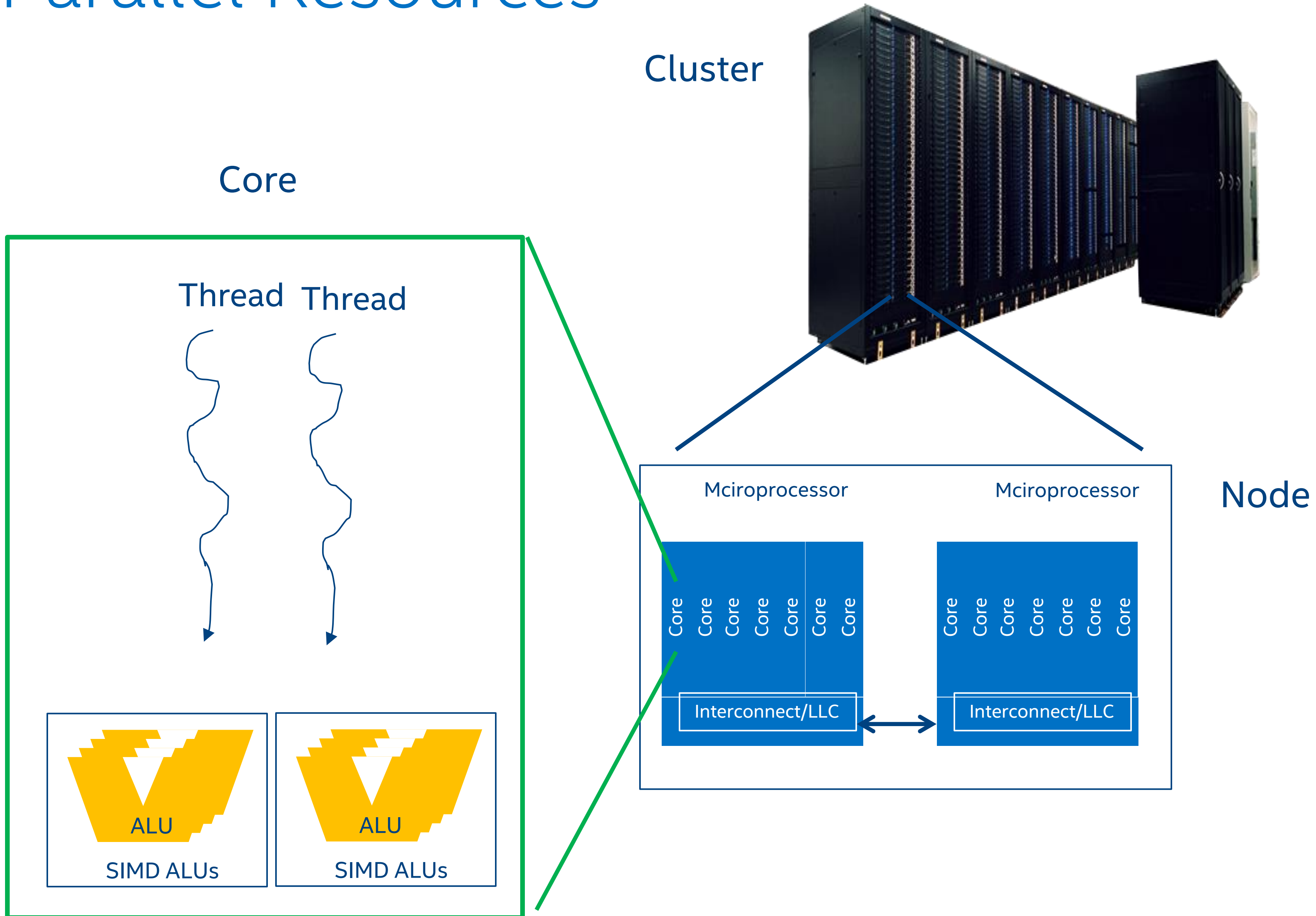



*"We think we can continue Moore's Law for at least another 10 years."*

Intel Senior Fellow Mark Bohr, 2015

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

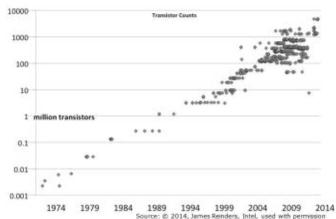
# Parallel Resources

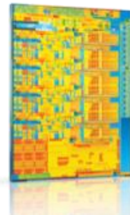
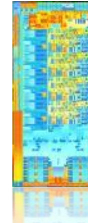



**Cluster → Node → Sockets → Processor/Co-processor → Core → Thread → SIMD (Vector)**

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# More cores . More Threads . Wider vectors



| | Intel® Xeon® processor 64-bit | Intel® Xeon® processor 5100 series | Intel® Xeon® processor 5500 series | Intel® Xeon® processor 5600 series | Intel® Xeon® processor code-named Sandy Bridge EP | Intel® Xeon® processor code-named Ivy Bridge EP | Intel® Xeon® processor code-named Haswell EP | Intel® Xeon Phi™ coprocessor Knights Corner | Intel® Xeon Phi™ processor & coprocessor Knights Landing[1] |
|---|---|---|---|---|---|---|---|---|---|
| Core(s) | 1 | 2 | 4 | 6 | 8 | 12 | 18 | 61 | 60+ |
| Threads | 2 | 2 | 8 | 12 | 16 | 24 | 36 | 244 | |
| SIMD Width | 128 | 128 | 128 | 128 | 256 | 256 | 256 | 512 | |

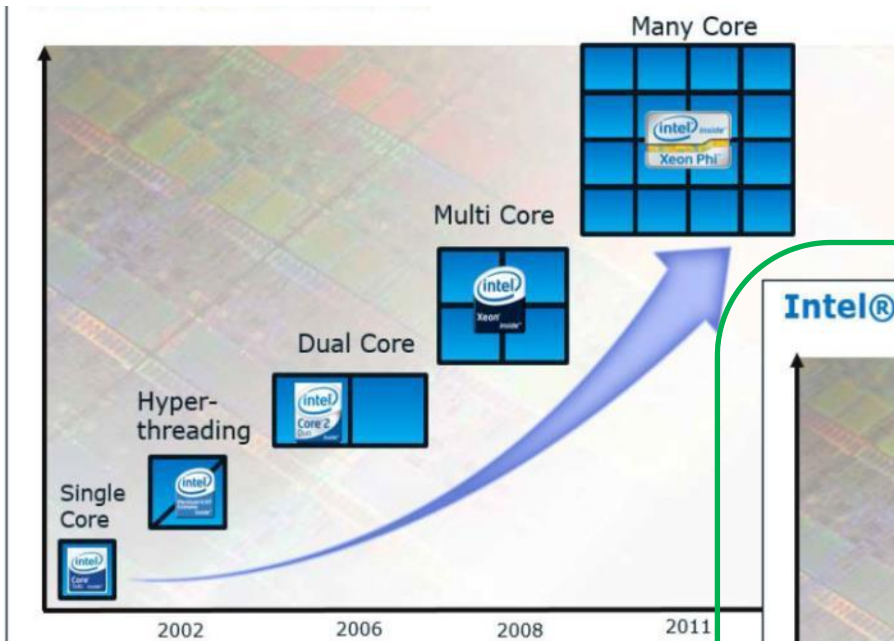
*Product specification for launched and shipped products available on ark.intel.com. 1. Not launched or in planning.

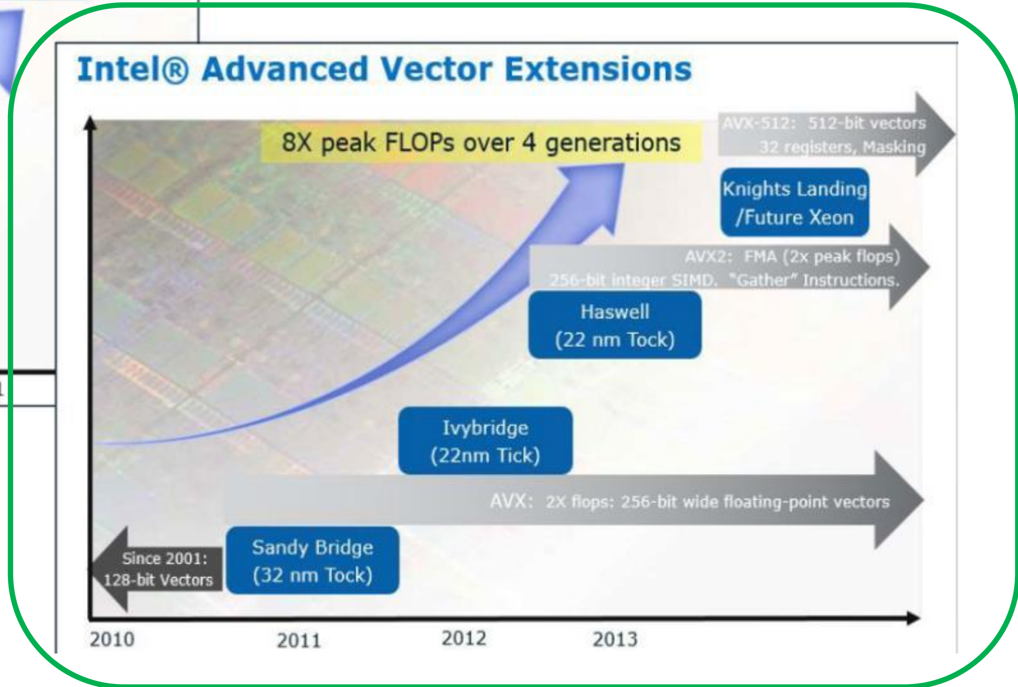
# High performance software must exploit both:

- Threading parallelism
- Vector data parallelism

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Why should we care about Vector SIMD parallelism at all?

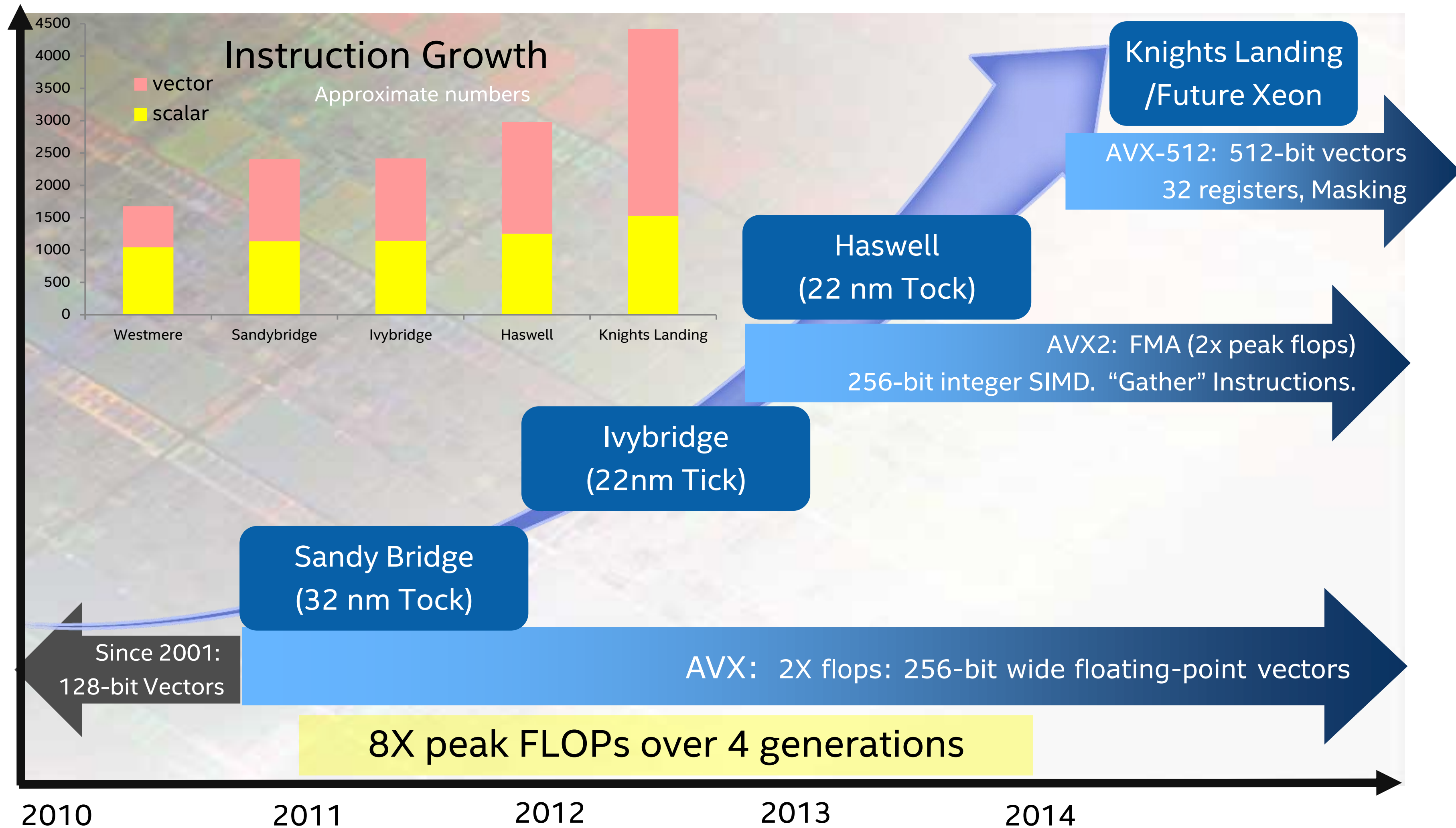



| 4C | 4C | 6C | 8C | 12C | 14C | 60C+ |
|---|---|---|---|---|---|---|
| 4 * SP | 4*SP | 4*SP | 8*SP | 8*SP | 16*SP | 32*SP |

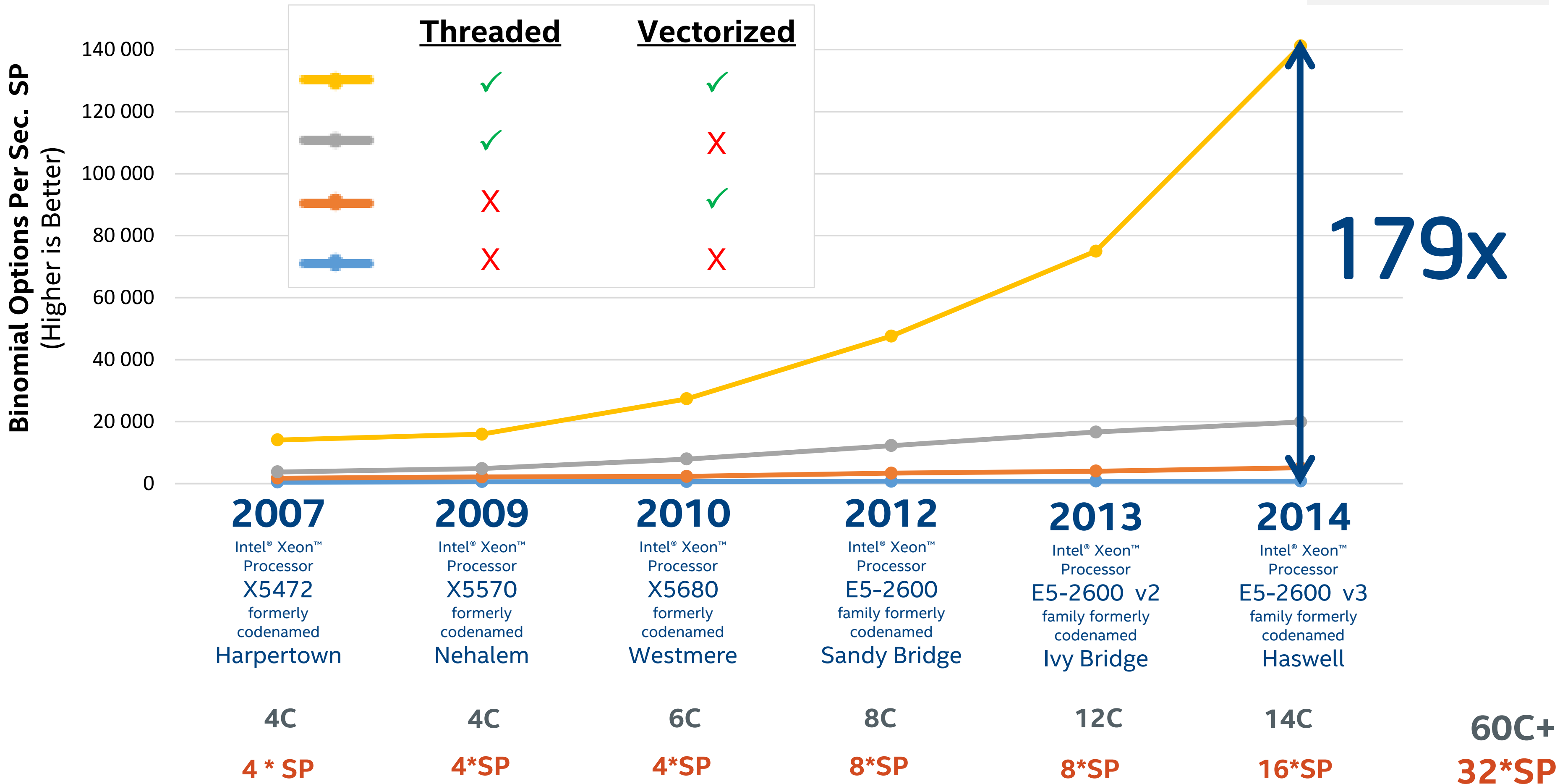
Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Intel® Advanced Vector Extensions




Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Untapped Potential Can Be Huge!



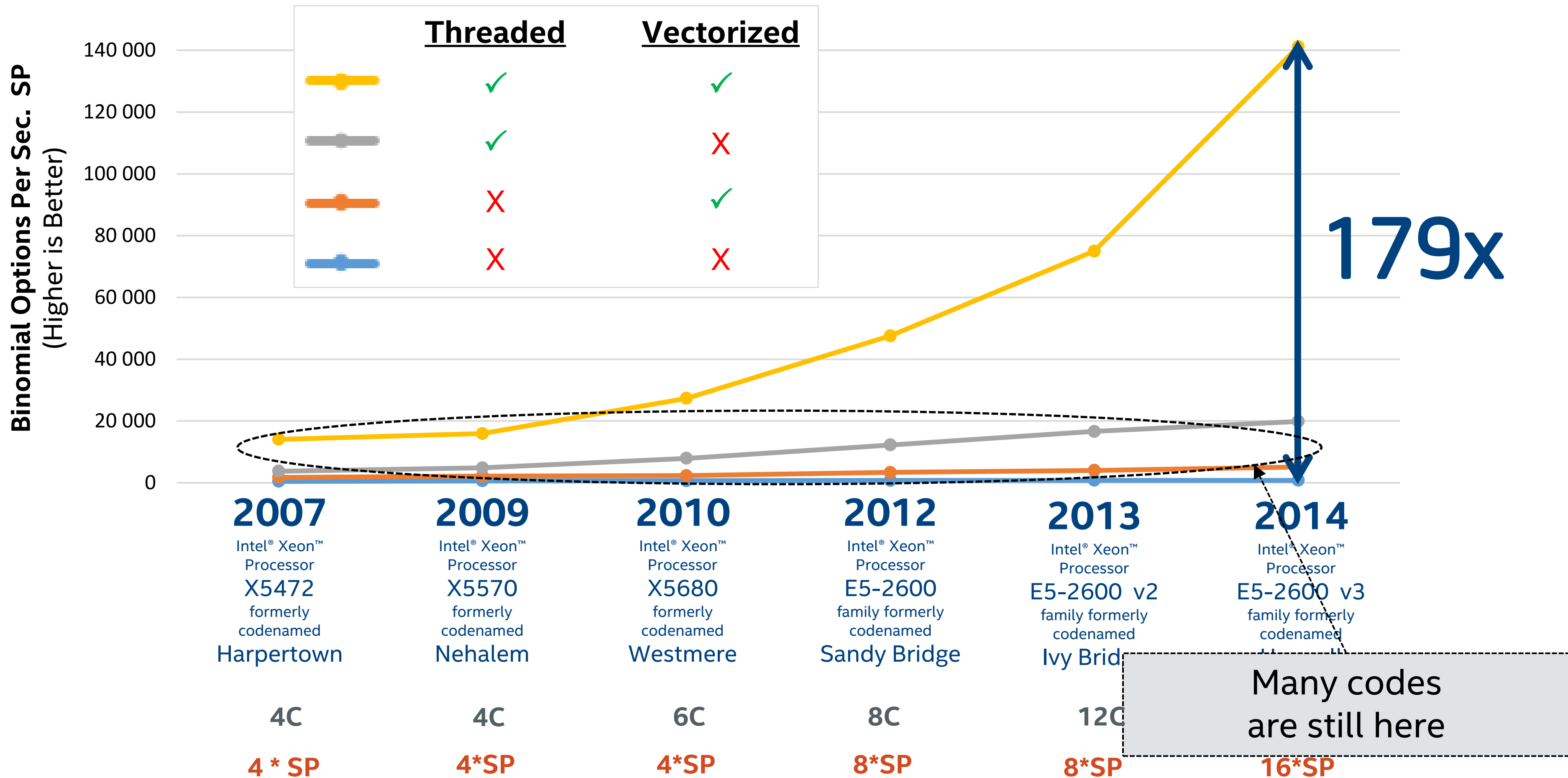

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may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

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Optimization Notice

# The Gap ~~Untapped Potential~~ Can Be Huge!

## Threaded + Vectorized can be much faster than either one alone




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



Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Optimization Notice

# Multi-Threading and Vectorization = Huge Potential

*Let's do some accounting..*

## Current Intel Xeon processor

12 cores
x 2 hyper-threads
x 8 lane (SP) vector unit per thread (x2 for FMA)

---

= **384**-folds parallelism for single socket

## Intel Many Integrated Core architecture

> 60 cores
x ?? independent threads per core
x 16 lane (SP) vector unit per thread (x2 for FMA)

---

= **parallel heaven**

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Don’t use a single Vector lane/thread!

## Un-vectorized and un-threaded software will under perform

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Permission to Design for All Lanes

Threading and Vectorization needed to fully utilize modern hardware

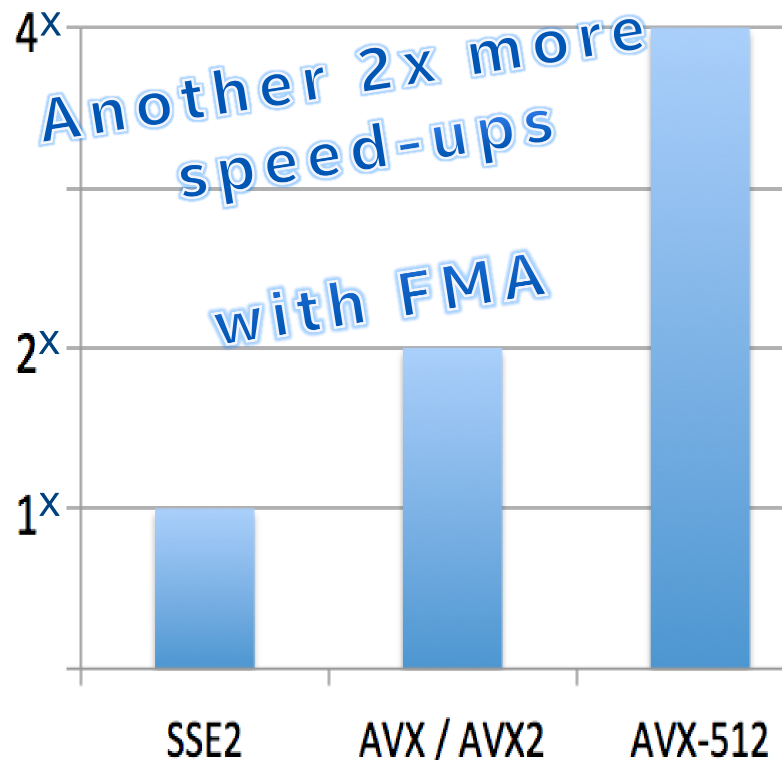
Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# SIMD vector parallelism for x86.

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Up to **8x Double-Precision** Performance

## with Intel® Advanced Vector Extensions 512 (Intel® AVX-512) Support




- Significant leap to 512-bit SIMD support for processors
- Intel® Compilers and Intel® Math Kernel Library include AVX-512 support
- Strong compatibility with AVX
- Added EVEX prefix enables additional functionality
- Appears first in future Intel® Xeon Phi™ coprocessor, code named Knights Landing

**Higher performance for the most demanding computational tasks**

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Why SIMD vector parallelism?

*Goal is higher performance and lower power*

$C_{dynamic}$ is roughly a product of area and activity
"how many bits" * "how much do they toggle"

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

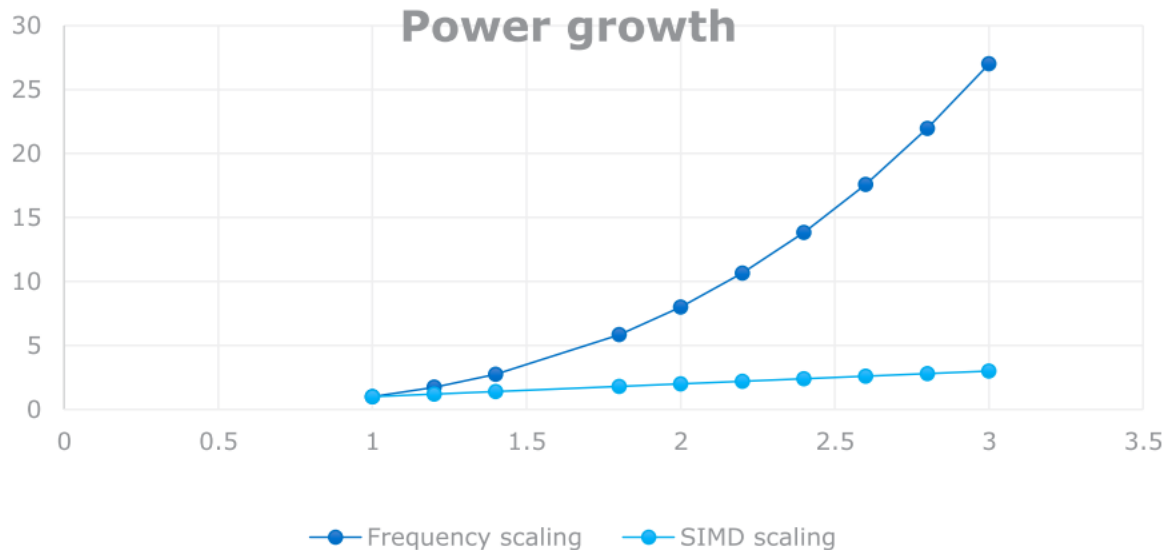
# Why SIMD vector parallelism?

Frequency is proportional to voltage.   Frequency reduction gives ***cubic reduction in power.***

$C_{dynamic}$ is roughly a product of area and activity
"how many bits" * "how much do they toggle"

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Why SIMD vector parallelism?

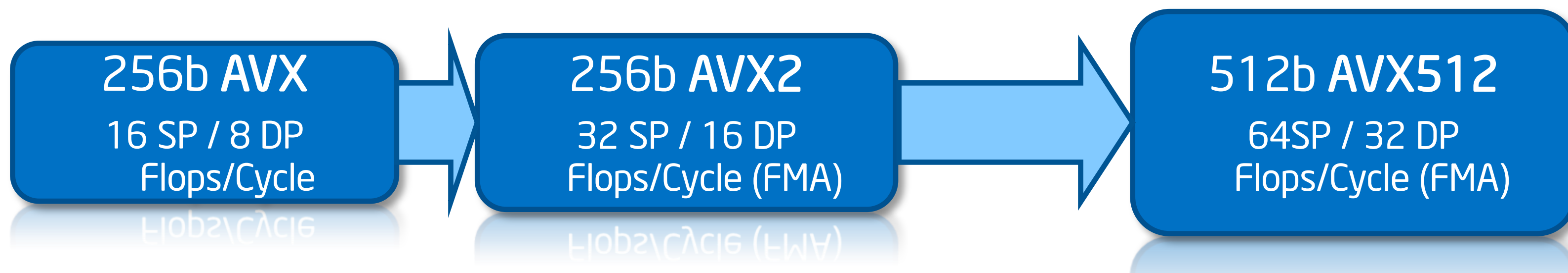



Wider SIMD -- Linear increase in area and power
Wider superscalar – Quadratic increase in area and power
Higher frequency – Cubic increase in power

**With SIMD we can go faster with less power**

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Intel® AVX Technology




| AVX | AVX2 |
|---|---|
| 256-bit basic FP | Float16 (IVB 2012) |
| **16 registers** | 256-bit FP FMA |
| NDS (and AVX128) | 256-bit integer |
| Improved blend | PERMD |
| MASKMOV | Gather |
| Implicit unaligned | |

| AVX512 |
|---|
| 512-bit FP/Integer |
| **32 registers** |
| **8 mask registers** |
| Embedded rounding |
| Embedded broadcast |
| Scalar/SSE/AVX "promotions" |
| HPC additions |
| Transcendental support |
| Gather/Scatter |

**SNB**
2011

**HSW**
2013

**Future Processors (KNL & future Xeon)**

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Optimization Notice

# What is a Vector?

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Vector of numbers

| 4.4 | 1.1 | 3.1 | -8.5 | -1.3 | 1.7 | 7.5 | 5.6 | -3.2 | 3.6 | 4.8 |
|---|---|---|---|---|---|---|---|---|---|---|

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Vector addition

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 4.4 | 1.1 | 3.1 | -8.5 | -1.3 | 1.7 | 7.5 | 5.6 | -3.2 | 3.6 | 4.8 |
| + | -0.3 | -0.5 | 0.5 | 0 | 0.1 | 0.8 | 0.9 | 0.7 | 1 | 0.6 | -0.5 |
| = | 4.1 | 0.6 | 3.6 | -8.5 | -1.2 | 2.5 | 8.4 | 6.3 | -2.2 | 4.2 | 4.3 |

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# …and Vector multiplication

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 4.4 | 1.1 | 3.1 | -8.5 | -1.3 | 1.7 | 7.5 | 5.6 | -3.2 | 3.6 | 4.8 |
| + | -0.3 | -0.5 | 0.5 | 0 | 0.1 | 0.8 | 0.9 | 0.7 | 1 | 0.6 | -0.5 |
| = | 4.1 | 0.6 | 3.6 | -8.5 | -1.2 | 2.5 | 8.4 | 6.3 | -2.2 | 4.2 | 4.3 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 4.4 | 1.1 | 3.1 | -8.5 | -1.3 | 1.7 | 7.5 | 5.6 | -3.2 | 3.6 | 4.8 |
| × | -0.3 | -0.5 | 0.5 | 0 | 0.1 | 0.8 | 0.9 | 0.7 | 1 | 0.6 | -0.5 |
| = | -1.32 | -0.55 | 1.55 | 0 | -0.13 | 1.36 | 6.75 | 3.92 | -3.2 | 2.16 | -2.4 |

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# An example

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# vector data operations: data operations done in parallel

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

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# vector data operations: data operations done in parallel

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

Loop:

1. LOAD a[i] -> Ra
2. LOAD b[i] -> Rb
3. ADD Ra, Rb -> Rc
4. STORE Rc -> c[i]
5. ADD i + 1 -> i

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# vector data operations: data operations done in parallel

```
void v_add (float *c,
```

Loop:
1. LOADv4 a[i:i+3] -> Rva
2. LOADv4 b[i:i+3] -> Rvb
3. ADDv4 Rva, Rvb -> Rvc
4. STOREv4 Rvc -> c[i:i+3]
5. ADD i + 4 -> i

Loop:
1. LOAD a[i] -> Ra
2. LOAD b[i] -> Rb
3. ADD Ra, Rb -> Rc
4. STORE Rc -> c[i]
5. ADD i + 1 -> i

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# vector data operations: data operations done in parallel

## We call this “vectorization”

```
void v_add (float *c,
            float *a,
            float *b)
...
  for (int i=0; i<= MAX; i++)
...
}
```

Loop:

1. LOADv4 a[i:i+3] -> Rva
2. LOADv4 b[i:i+3] -> Rvb
3. ADDv4 Rva, Rvb -> Rvc
4. STOREv4 Rvc -> c[i:i+3]
5. ADD i + 4 -> i

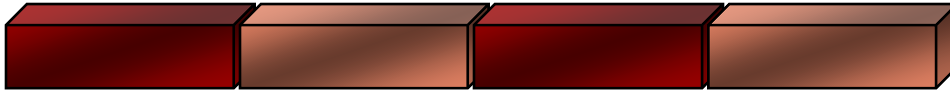
Loop:

1. LOAD a[i] -> Ra
2. LOAD b[i] -> Rb
3. ADD Ra, Rb -> Rc
4. STORE Rc -> c[i]
5. ADD i + 1 -> i

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

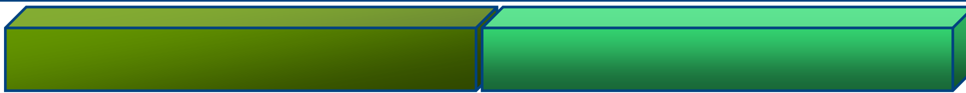
# Intel® SSE and AVX-128 Data Types

SSE

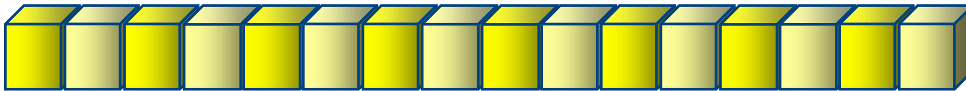


4x floats

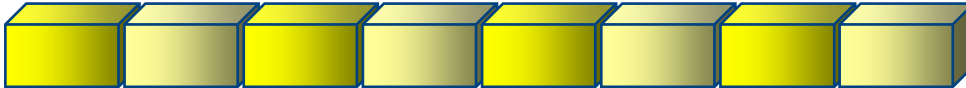
SSE-2



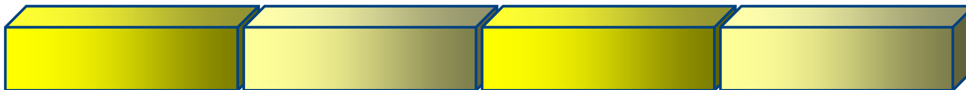
2x doubles



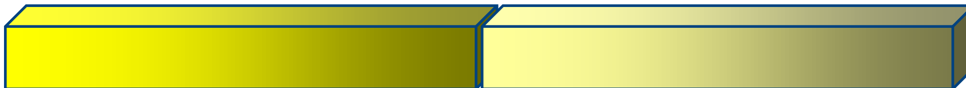
16x bytes



8x 16-bit shorts



4x 32-bit integers



2x 64-bit integers

1x 128-bit(!) integer

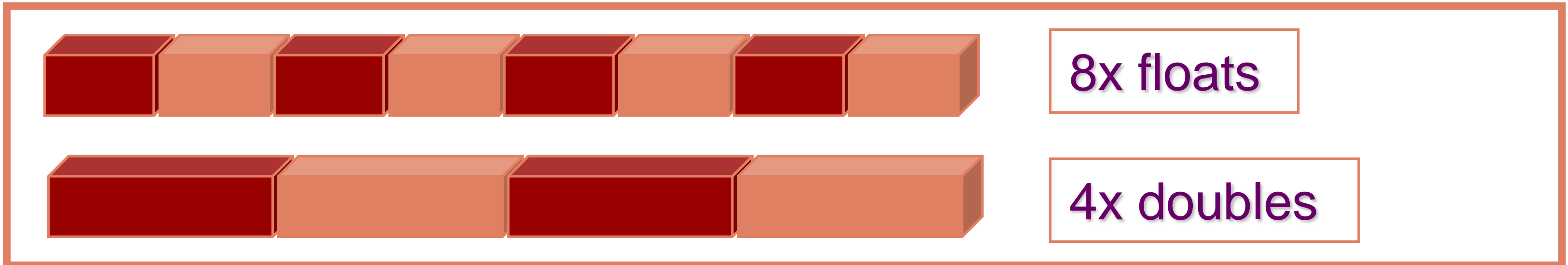


Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Optimization Notice

# AVX-256 Data Types

Intel® AVX




Intel® AVX2

32x bytes

16x 16-bit shorts

8x 32-bit integers

4x 64-bit integers

2x 128-bit(!) integer



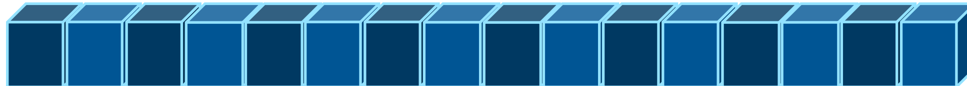
Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Optimization Notice

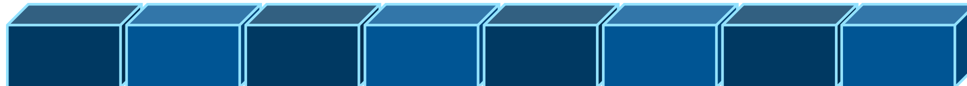
intel

# Data Types for Intel® MIC Architecture

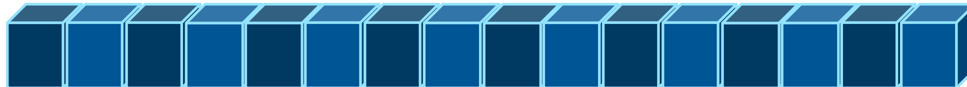
MIC



16x floats



8x doubles



16x 32-bit integers

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Next generation Intel® Xeon Phi™

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Optimization Notice

# Next Intel® Xeon Phi™ Product Family

## *(Codenamed Knights Landing)*

- Available in Intel cutting-edge 14 nanometer process
- Stand alone CPU or PCIe coprocessor – not bound by 'offloading' bottlenecks
- Integrated Memory - balances compute with bandwidth

**Parallel is the path forward, Intel is your roadmap!**

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

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Knights Landing: Next Generation Intel® Xeon Phi™ Product Family

*Designed using Intel's cutting-edge*
***14nm Transistor Technology***

Intel leads the industry in transistor technology by about three years; 14nm technology will deliver more compute density and power efficiency than any previous Intel processor.[1]

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

As a host processor directly installed on the motherboard, Knights Landing will deliver a leap in compute density, power efficiency & reliability.

*Common instruction set architecture*
***Intel® Advanced Vector Extensions 512***

Commonality & backward compatibility of next-gen 512-bit instruction set architectures; supported on future Intel® Xeon® processors to be introduced after Knights Landing.

*Leadership compute & memory bandwidth*
***Integrated on-package Memory***

On-package memory will significantly increase memory bandwidth, delivering greater performance for memory-bound workloads and help scale the Exascale memory wall. Learn More

[1] http://newsroom.intel.com/community/intel_newsroom/blog/2013/09/10/new-intel-ceo-president-outline-product-plans-future-of-computing-vision-to-mobilize-intel-and-developers

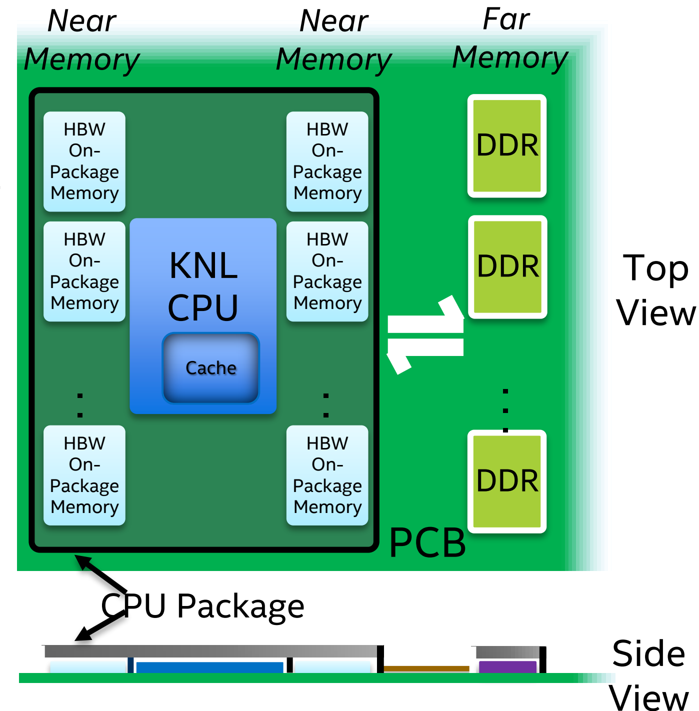
Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Knights Landing Integrated On-Package Memory

**Cache Model** — Let the hardware automatically manage the integrated on-package memory as an "L3" 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[1]

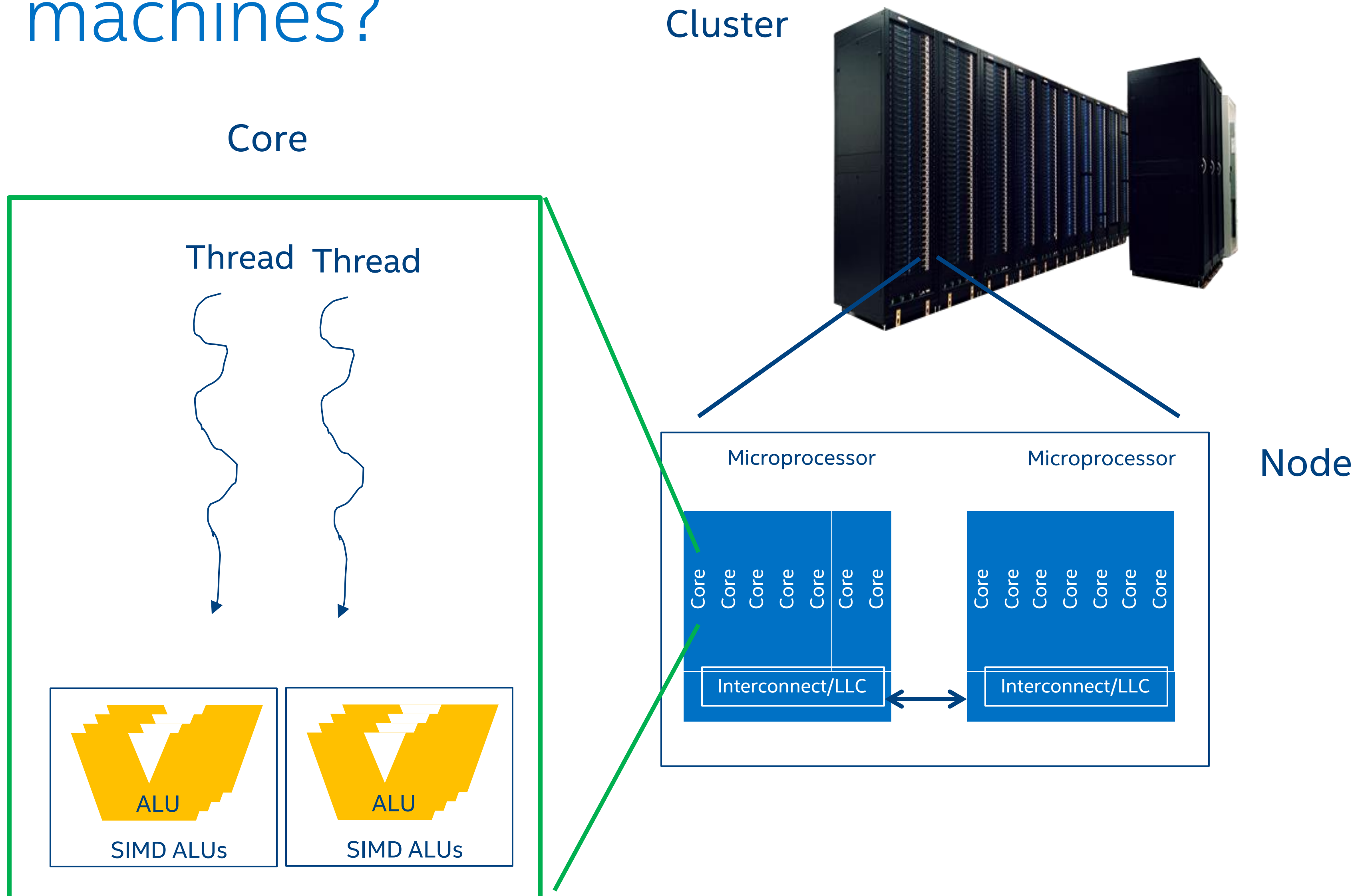
[1] As compared with Intel® Xeon Phi™ x100 Coprocessor Family
Diagram is for conceptual purposes only and only illustrates a CPU and memory – it is not to scale, and is not representative of actual component layout.

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Parallel programming models. “3 layer cake” with OpenMP4.x examples

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

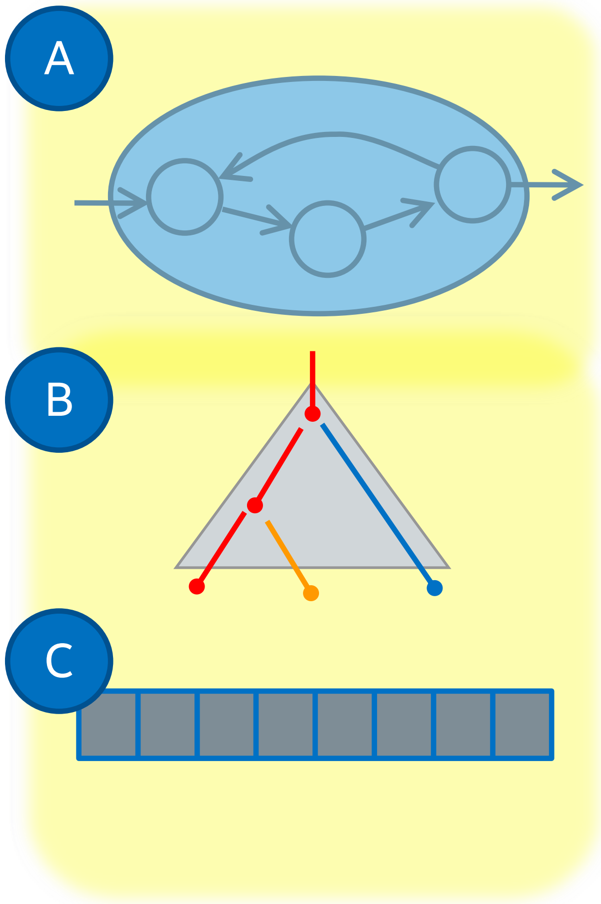
# How could we program these parallel machines?




**Cluster → Node → Sockets → Processor/Co-processor → Core → Thread → SIMD (Vector)**

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# How could we program these parallel machines?



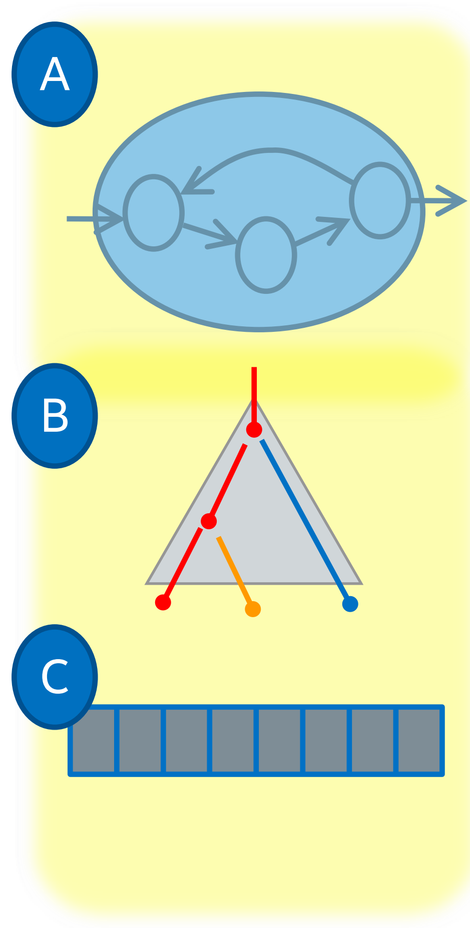

“Three Layer Cake”

“abstracts” common hybrid parallelism programming approaches

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Optimization Notice

# How could we program these parallel machines?



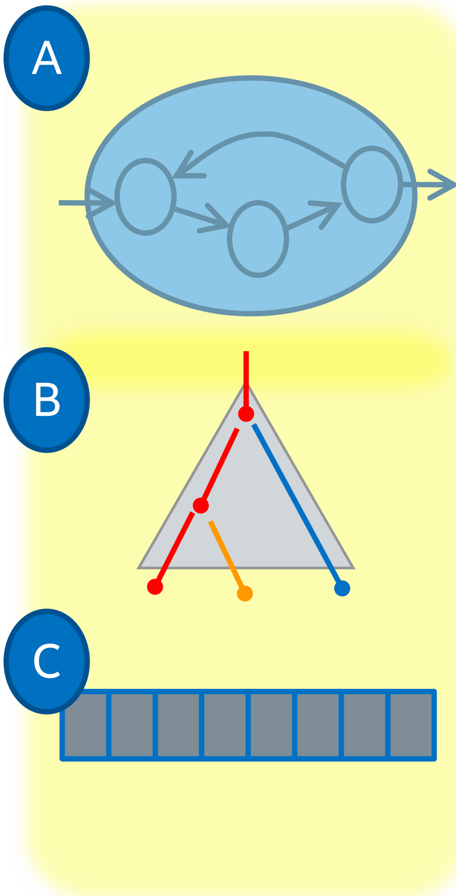

| **Parallelism type** | **Exploiting hardware* :** |
|---|---|
| **A – Message Passing** | **A:** exploit multiple **nodes**, distributed memory |
| **B – Fork-Join** | B – exploit multiple **cores**, hardware threads |
| **C- SIMD** | C- exploit **vector units** |

* - alternate hardware mappings also possible

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# How could we program these parallel machines?

Implementing *the Cake*




**Programming models**

A – MPI, tbb::flow, PGAS

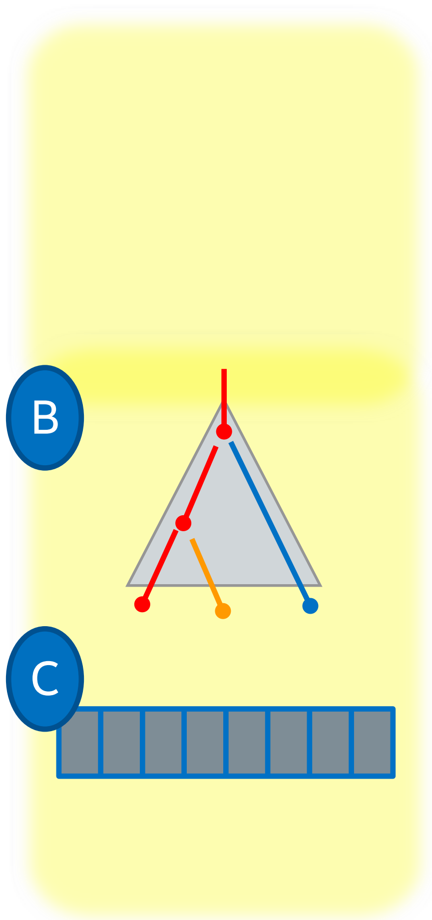
B – **OpenMP4.x,** Cilk Plus, TBB

C - **OpenMP4.x,** Cilk Plus

**Software tools**

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

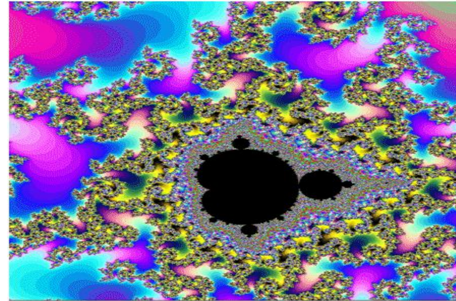
# How could we program these parallel machines?

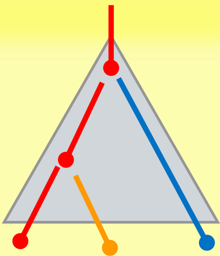



- Different methods exist
- OpenMP4.x:
  - Industry standard
  - C/C++ and Fortran
  - Supported by Intel Compiler (14, 15, 16), GCC 4.9, …
  - Both levels of hardware parallelism: threading and vector

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# 2 level parallelism decomposition with **OpenMP4.x**: image processing example

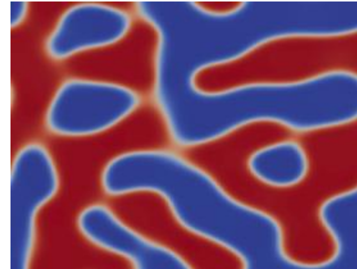


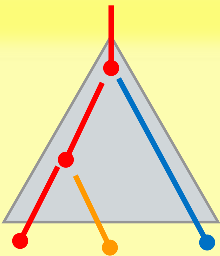


```
#pragma omp parallel for
 for (int y = 0; y < ImageHeight; ++y){
#pragma omp simd
    for (int x = 0; x < ImageWidth; ++x){
        count[y][x] = mandel(in_vals[y][x]);
    }
 }
```

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# 2L parallelism decomposition with **OpenMP4.x**: fluid dynamics example





B

C

```
#pragma omp parallel for
 for (int i = 0; i < X_Dim; ++i){
#pragma omp simd
    for (int m = 0; x < n_velocities; ++m){
       next_i = f(i, velocities(m));
       X[i] = next_i;
    }
 }
```

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Many Ways to Vectorize

**Compiler:**
**Auto-vectorization (no change of code)**

**Compiler:**
**Auto-vectorization hints (`#pragma vector`, ...)**

Explicit Vector Programming (OpenMP4.x, Intel Cilk Plus)

**SIMD intrinsic class**
**(e.g.: `F32vec`, `F64vec`, ...)**

**Vector intrinsic**
**(e.g.: `_mm_fmadd_pd(...)`, `_mm_add_ps(...)`, ...)**

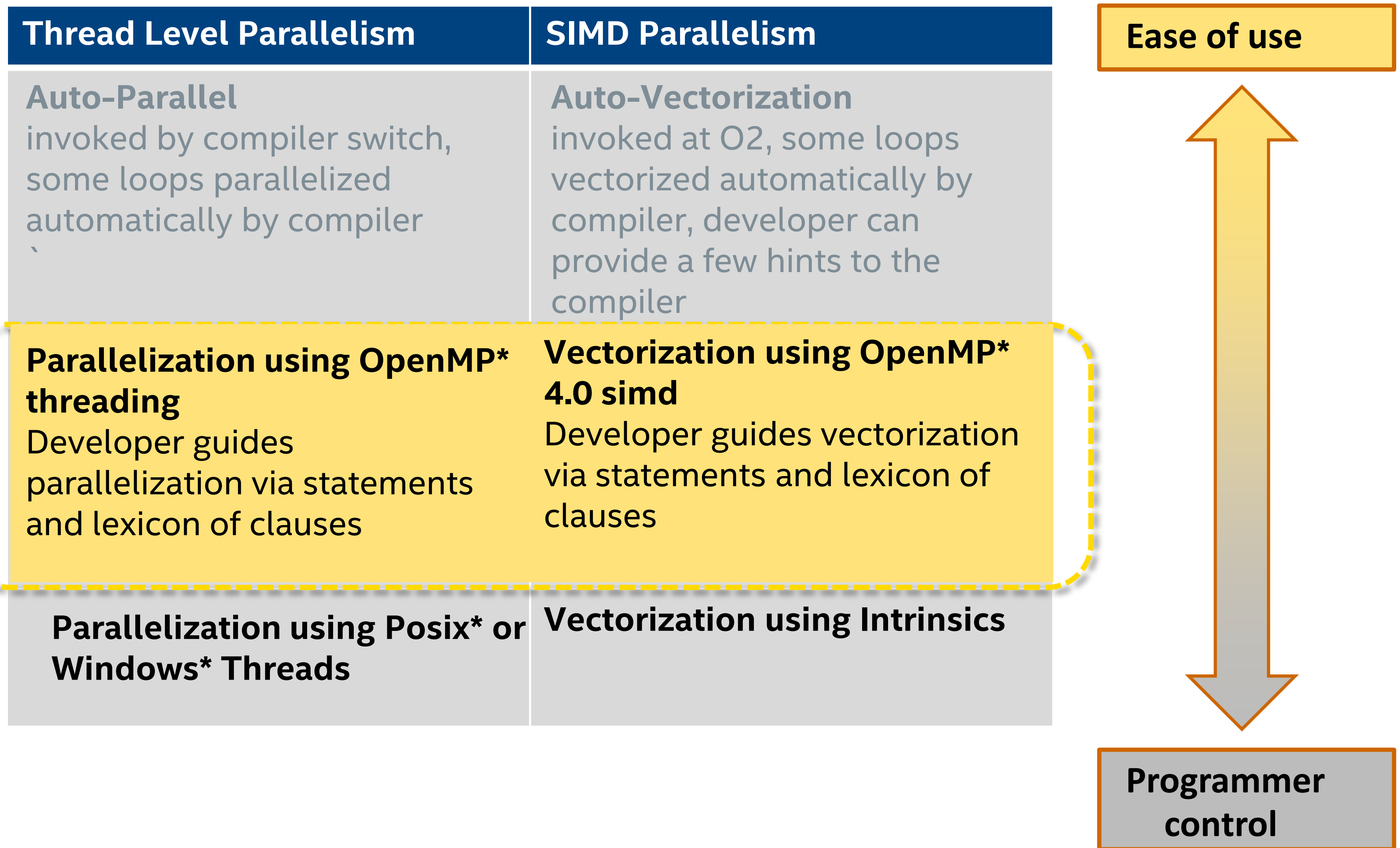
**Assembler code**
**(e.g.: `[v]addps`, `[v]addss`, ...)**

**Ease of use**

**Programmer control**

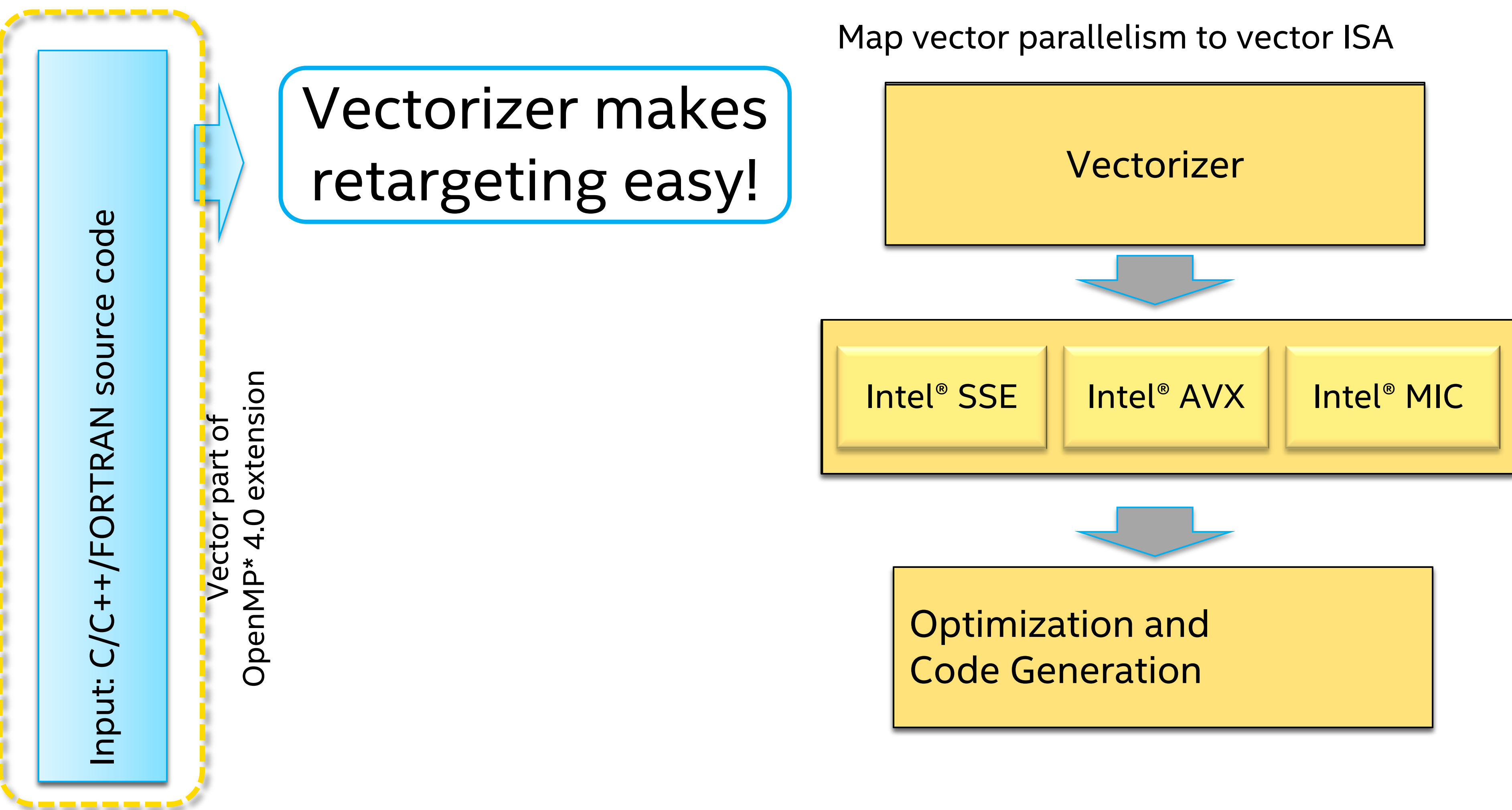
Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# OpenMP4.x: threading and vectors




Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Explicit Vector Programming with OpenMP 4.0




Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Compiling for Intel® AVX(2)

## Compile with –xavx (Intel® AVX; Sandy Bridge etc)

## Compile with –xcore-avx2 (Intel® AVX2; Haswell)

- Intel processors only (Use -mavx, -march=core-avx2 for non-Intel)
- Vectorization works just as for SSE
  - Best if 32 byte aligned
- More loops can be vectorized than with SSE
  - Individually masked data elements
  - More powerful data rearrangement instructions

## -axavx (-axcore-avx2) gives both SSE2 and newer ISA code paths

- (!) but use –x or –m switches to modify the default SSE2 code path
  - Eg –axcore-avx2 –xavx to target both Haswell and Sandy Bridge (/Qaxcore-avx2 /Qxavx on Windows*)

## Math libraries may target AVX and/or AVX2 automatically at runtime

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# SIMD Pragma Notation

## OpenMP 4.0: #pragma omp simd [clause  [,clause] …]

- **Targets loops**
  - Can target inner or outer canonical loops
- **Developer asserts loop is suitable for SIMD**
  - The Intel Compiler will vectorize if possible (will ignore dependency or efficiency concerns)
  - Use when you **KNOW** that a given loop is safe to vectorize
  - Can choose from lexicon of clauses to modify behavior of SIMD directive
- **Developer should validate results (correctness)**
  - Just like for race conditions in OpenMP* threading loops
- **Minimizes source code changes needed to enforce vectorization**

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# OMP SIMD Pragma Clauses

**`reduction(operator:v1, v2, …)`**

- v1 etc are reduction variables for operation “operator”
- Examples include computing averages or sums of arrays into a single scalar value : *reduction (+:sum)*

**`linear(v1:step1, v2:step2, …)`**

- declares one or more list items to be private to a SIMD lane and to have a linear relationship with respect to the iteration space of a loop : *linear (i:2)*

**`safelen (`*`length`*`)`**

- no two iterations executed concurrently with SIMD instructions can have a greater distance in the logical iteration space than this value
- *Typical values are* 2, 4, 8, 16

Refer to OpenMP 4.0 Specification.
http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf


Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.
Optimization Notice

# OMP SIMD Pragma Clauses cont…

**`aligned(v1:alignment, v2:alignment)`**

- declares that the object to which each list item points is aligned to the number of bytes expressed in the optional parameter of the aligned clause.

**`collapse(number of loops)`**

- Nested loop iterations are collapsed into one loop with a larger iteration space.

**`private(v1, v2, …), lastprivate (v1, v2, …)`**

- declares one or more list items to be private to an implicit task or to a SIMD lane, lastprivate causes the corresponding original list item to be updated after the end of the region..

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# SIMD-enabled functions

Write a function for one element and add **pragma** as follows

```
#pragma omp declare simd
float foo(float a, float b, float c, float d)
{
  return a * b + c * d;
}
```

Call the scalar version:

```
e = foo(a, b, c, d);
```

Call vector version via SIMD loop:

```
#pragma omp simd
for(i = 0; i < n; i++) {
  A[i] = foo(B[i], C[i], D[i], E[i]);
}
```

```
A[:] = foo(B[:], C[:], D[:], E[:]);
```

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# What's New in Beta

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Intel® C/C++ and Fortran Compilers 16.0
## Get best performance with latest standards

- More of C++14, generic lambdas, member initializers and aggregates
- More of C11, _Static_assert, _Generic, _Noreturn, and more
- OpenMP 4.0 C++ User Defined Reductions, Fortran Array Reductions
- OpenMP 4.1 asynchronous offloading, simdlen, simd ordered
- F2008 Submodules, Impure Elemental Functions
- F2015 TYPE(*), DIMENSION(..), RANK intrinsic, attributes for args with BIND
- Significant improvement in alignment analysis, vectorization robustness
- Much improved Neighboring Gather optimization

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Intel® Advisor XE – New! Vectorization Advisor

## Data Driven Vectorization Design

### Have you:

- Recompiled with AVX2, but seen little benefit?
- Wondered where to start adding vectorization?
- Recoded intrinsics for each new architecture?
- Struggled with cryptic compiler vectorization messages?

### Breakthrough for vectorization design

- What vectorization will pay off the most?
- What is blocking vectorization and why?
- Are my loops vector friendly?
- Will reorganizing data increase performance?
- Is it safe to just use pragma simd?

**Threading and Vectorization Survey** Intel Advisor XE 2016

Summary | Survey Report | Suitability Report | Correctness Report | Memory Access Patterns

Filter by Loop Type: Vectorized | Not Vectorized — Filter by Source: All — Filter by Module: All

| Function Call Sites and Loops | Self Time | Total Time | Memory analysis | Compiler Vectorization: Loop Type | Gain Estimate | Vectorized Loops |
|---|---|---|---|---|---|---|
| **[loop at mmult_se...** | **10.040s** | **10.040s** | ☐ | **Vectorized ...** | **2.19727** | **SSE2** |
| [loop at mmult_serial.cp... | 0.000s | 10.100s | | Scalar | | SSE2 |
| [loop at mmult_serial.cp... | 0.000s | 10.100s | | Scalar | | |
| [loop in __libc_start_mai... | 0.000s | 10.100s | | Scalar | | |

Top Down | Source | Assembly | Assistance | Recommendations

| Function Call Sites and Loops | Total Time % | Total Time | Self Time | Hot Loops | Vector... Loops | Location: Source Loc... | Module |
|---|---|---|---|---|---|---|---|
| Total | 100.0% | 10.100s | 0s | | | | |
| __libc_start_main | 100.0% | 10.100s | 0s | | | | libc-2.12.so |
| [loop in __libc_st... | 100.0% | 10.100s | 0s | | | | libc-2.12.so |
| main | 100.0% | 10.100s | 0s | | | mmult_seri... | 1_mmult_serial |
| [loop at mm... | 100.0% | 10.100s | 0s | | | mmult_seri... | 1_mmult_serial |
| [loop at m... | 100.0% | 10.100s | 0s | | SSE2 | mmult_seri... | 1_mmult_serial |
| multiply_d | 100.0% | 10.100s | 0.0600s | | | mmult_seri... | 1_mmult_serial |
| [loop ... | 99.4% | 10.040s | 10.0400s | | SSE2 | mmult_seri... | 1_mmult_serial |

**More Performance**
**Fewer Machine Dependencies**

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Optimization Notice

# Intel® Advisor XE – Vectorization Advisor

## Provides the data you need for high impact vectorization

Compiler diagnostics + Performance Data = All the data you need in one place

- Find “hot” un-vectorized or “under vectorized” loops.
- Trip counts

Recommendations – How do I fix it?

Correctness via dependency analysis

- Is it safe to vectorize?

Memory Access Patterns analysis

- Unit stride vs Non-unit stride access, Unaligned memory access, etc.

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Intel® Math Kernel Library (Intel® MKL) 11.3

## Better performance with new two-stage API for Sparse BLAS routines

### Additional Sparse Matrix Vector Multiplication API

- new two-stage API for Sparse BLAS level 2 and 3 routines

### MKL MPI wrappers

- all MPI implementations are API-compatible but MPI implementations are **not ABI-compatible**
- MKL MPI wrapper solves this problem by providing an MPI-independent ABI to MKL

### Support For Batched Small Matrix multiplication

- a single call executes multiple independent ?GEMM operation simultaneously

### Support for Philox4x35 and ARS5 RNG

- two new pseudorandom number generators with a period of 2^128 are highly optimized for multithreaded environment

### Sparse Solver SMP improvements

- significantly improved overall scalability for Intel Xeon Phi coprocessors and Intel Xeon processors

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Intel® Data Analytics Acceleration Library 2016

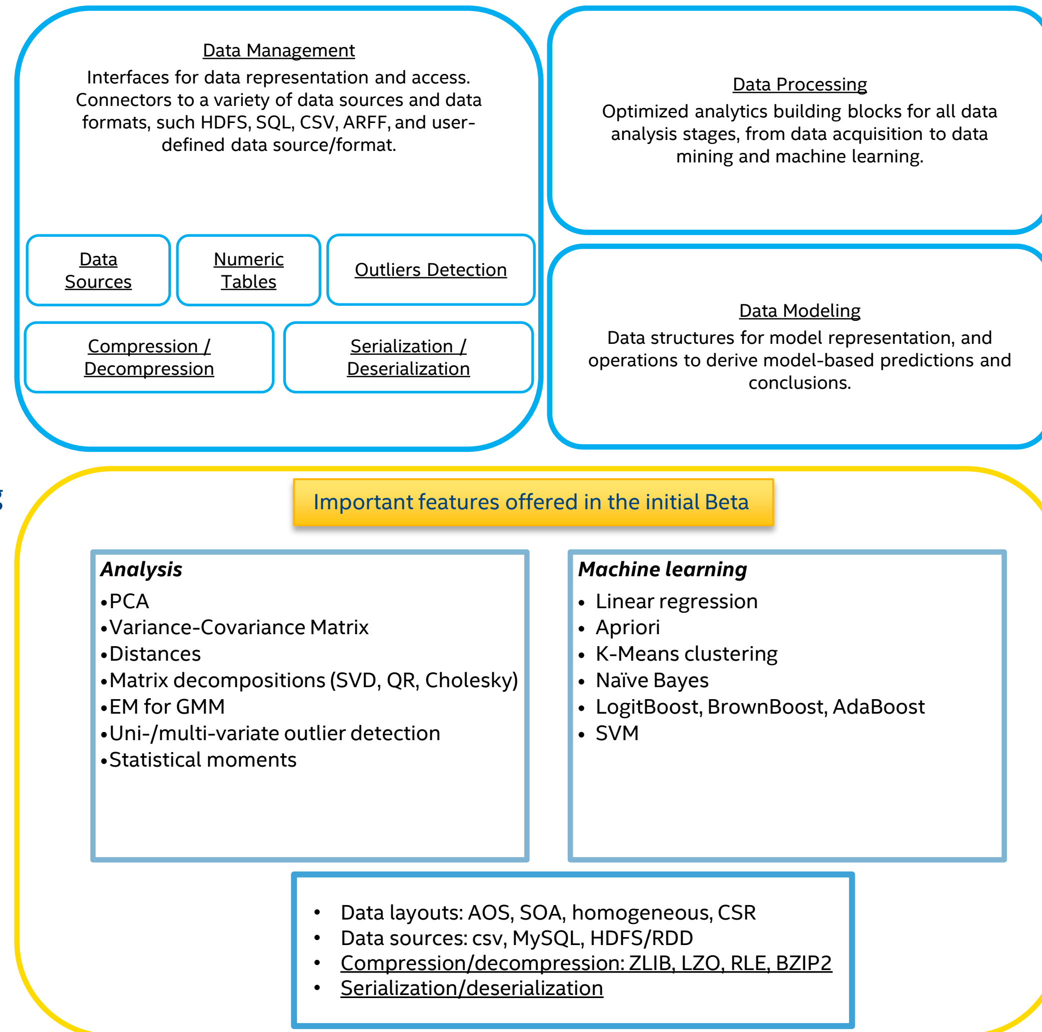
A C++ and Java API library of optimized analytics building blocks for all data analysis stages, from data acquisition to data mining and machine learning. Essential for engineering high performance Big Data applications.

## New library targeting data analytics market

- **Customers:** analytics solution providers, system integrators, and application developers (FSI, Telco, Retail, Grid, etc.)
- **Key benefits:** improved time-to-value, forward-scaling performance and parallelism on IA, advanced analytics building blocks

## Key features

- Building blocks highly optimized for IA to support all data analysis stages
- Support batch, streaming, and distributed processing with easy connectors to popular platforms (Hadoop, Spark) and tools (R, Python, Matlab)
- Flexible interfaces for handling different data sources (CSV, MySQL, HDFS, RDD (Spark))
- Rich set of operations to handle sparse and noisy data
- C++ and Java APIs




Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Optimization Notice

# Intel® VTune™ Amplifier XE 2016 Beta
# Enhanced GPU and Microarchitecture Profiling

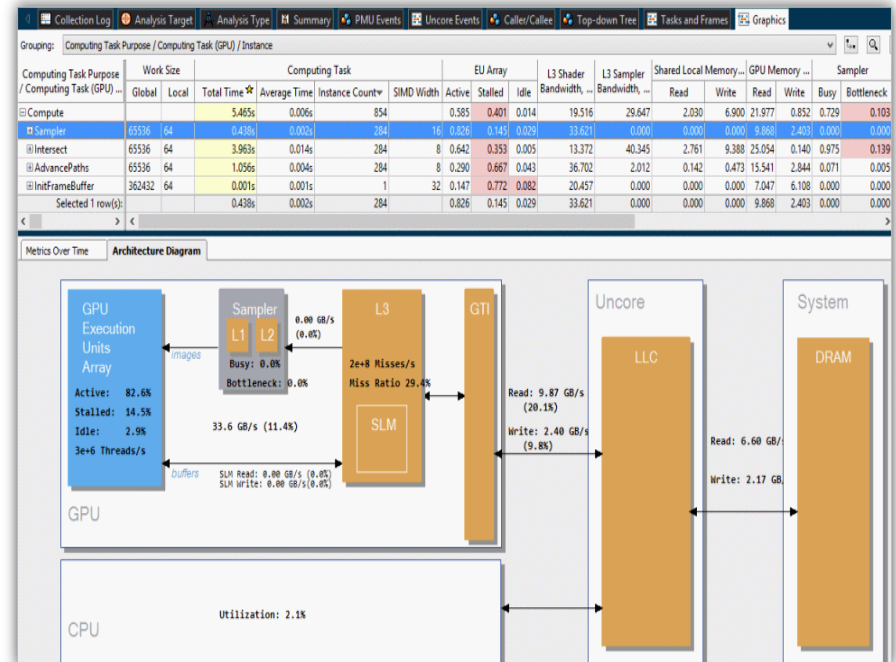
## New OS and IDE support: Visual Studio* 2015 & Windows* 10 Threshold

## Intel® HD Graphics (GPU) profiling

- GPU Architecture Annotation Diagram
- GPU profiling on Linux (OpenCL, Media SDK)

## Microarchitecture tuning

- General Exploration analysis with confidence indication
- Driverless ‘perf’ EBS with stacks




| Computing Task Purpose / Computing Task (GPU) ... | Work Size | | Computing Task | | | | EU Array | | | L3 Shader Bandwidth, ... | L3 Sampler Bandwidth, ... | Shared Local Memory... | | GPU Memory ... | | Sampler | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Global | Local | Total Time | Average Time | Instance Count | SIMD Width | Active | Stalled | Idle | | | Read | Write | Read | Write | Busy | Bottleneck |
| Compute | | | 5.465s | 0.006s | 854 | | 0.585 | 0.401 | 0.014 | 19.516 | 29.647 | 2.030 | 6.900 | 21.977 | 0.852 | 0.729 | 0.103 |
| Sampler | 65536 | 64 | 0.438s | 0.002s | 284 | 16 | 0.826 | 0.145 | 0.029 | 33.621 | 0.000 | 0.000 | 0.000 | 9.868 | 2.403 | 0.000 | 0.000 |
| Intersect | 65536 | 64 | 3.963s | 0.014s | 284 | 8 | 0.642 | 0.353 | 0.005 | 13.372 | 40.345 | 2.761 | 9.388 | 25.054 | 0.140 | 0.975 | 0.139 |
| AdvancePaths | 65536 | 64 | 1.056s | 0.004s | 284 | 8 | 0.290 | 0.667 | 0.043 | 36.702 | 2.012 | 0.142 | 0.473 | 15.541 | 2.844 | 0.071 | 0.005 |
| InitFrameBuffer | 362432 | 64 | 0.001s | 0.001s | 1 | 32 | 0.147 | 0.772 | 0.082 | 20.457 | 0.000 | 0.000 | 0.000 | 7.047 | 6.108 | 0.000 | 0.000 |
| Selected 1 row(s): | | | 0.438s | 0.002s | 284 | | 0.826 | 0.145 | 0.029 | 33.621 | 0.000 | 0.000 | 0.000 | 9.868 | 2.403 | 0.000 | 0.000 |



Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Intel® VTune™ Amplifier XE 2016 Beta
# Improved OpenMP* and Hybrid Support

## Intel OpenMP analysis enhancements

- Precise trace-based imbalance calculation that is especially useful for profiling of small region instances
- Classification and issue highlighting of potential gains, e.g., imbalance, lock contention, creation overhead, etc.
- Detailed analysis of barrier-to-barrier region segments

Dynamic scheduling overhead

## MPI+OpenMP: multi-rank analysis on a compute node

- Per-rank OpenMP potential gain and serial time metrics
- Per-rank Intel MPI communication busy wait time detection

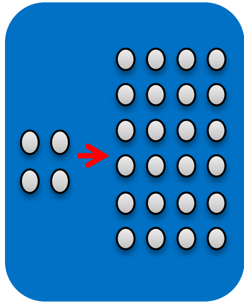
Imbalance

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

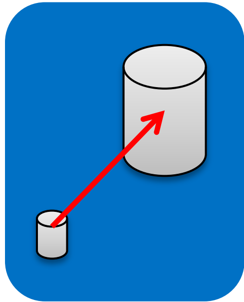
Optimization Notice

# MPI Performance Snapshot

## Scalable profiling for MPI and Hybrid



**Lightweight** – Low overhead profiling for 32K+ Ranks



**Scalability-** Performance variation at scale can be detected sooner

**Identifying Key Metrics –** Shows PAPI counters and MPI/OpenMP imbalances

### MPI Performance Snapshot Summary

Application: ./hybrid
Ranks: 4
Used statistics: pcs_r4_f020.txt

#### Overview

- MPI Time: 14.70 sec — 48.99%
  - MPI Imbalance: 14.69 sec — 48.98%
- Computation Time: 15.30 sec — 50.98%
  - OpenMP Time: 14.90 sec — 49.66%
    - OpenMP Imbalance: 7.37 sec — 24.57%
  - Serial Time: 0.40 sec — 1.32%

#### Memory Usage

- Peak memory consumption (rank 1): 0.79 MB
- Mean memory consumption: 0.75 MB

Per process memory usage affects the application scalability.

#### Performance by Metric

WallClock time: 30.00 sec
Total application lifetime. The time is elapsed time for the slowest process. This metric is the sum of the MPI Time and the Computation time below.

MPI Time: 14.70 sec — 48.99%
Time spent inside the MPI library. High values are usually bad.
This value is HIGH. The application is Communication-bound. More details...

MPI Imbalance: 14.69 sec — 48.98%
Mean unproductive wait time per-process spent in the MPI library calls when a process is waiting for data. This time is part of the MPI time above. High values are usually bad.
This value is HIGH. The application workload is NOT well balanced between MPI ranks. More details...

Computation Time: 15.30 sec — 50.98%
Mean time per-process spent in the application code. This is the sum of the OpenMP Time and the Serial time. High values are usually good.
This value is AVERAGE. The application is Computation-bound. More details...

OpenMP Time: 14.90 sec — 49.66%
Mean time per process spent in the OpenMP parallel regions. High values are usually good and indicate that the application is well-threaded.
This value is AVERAGE.

OpenMP Imbalance: 7.37 sec — 24.57%
Mean unproductive wait time per-process spent in OpenMP parallel regions (normally at synchronization barriers). High values are usually bad.
This value is HIGH. The application's OpenMP work sharing is NOT well load-balanced. More details...

Serial Time: 0.40 sec — 1.32%
Mean application time per-process spent outside OpenMP parallel regions. High values may be good or bad depending on the application algorithm.
This value is NEGLIGIBLE. This application is well parallelized via OpenMP directives.

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# ISA (AVX, AVX2, AVX512) : basic insight

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Intel® AVX

# Intel® Advanced Vector Extensions (Intel® AVX)

| *KEY FEATURES* | *BENEFITS* |
| --- | --- |
| • **Wider Vectors**<br>  ▪ **Increased from 128 bit to 256 bit** | • Up to 2x peak FLOPs output with good power efficiency |
| • Enhanced Data Rearrangement<br>  – Use the new 256 bit primitives to broadcast, mask loads and permute data | • Organize, access and pull only necessary data more quickly and efficiently |
| • Flexible unaligned memory access support | • More opportunities to fuse load and compute operations |

Intel® AVX is a general purpose architecture, expected to supplant SSE in all applications used today

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Applications likely to benefit from rebuilding for Intel® AVX

- Significant time spent in floating-point vectorizable loops with iteration count > vector length (8 floats, 4 doubles)
  - Vectorization with SSE is a good initial indication
- Calls to optimized performance libraries, e.g. MKL
  - Might not even need rebuilding

**Less likely to benefit:**

- Scalar or integer code;
- Heavy use of double precision division or square root
- Applications that are memory bound

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Intel® AVX2

# Intel® AVX2: Key Features

1. Extends 128-bit <u>integer</u> vector instructions to 256-bit
   - Including*: Intel® SSE2, Intel SSE3, SSSE3 and Intel SSE4 (some special instructions excepted)
2. Floating Point Fused Multiply Add: A*B + C
   - Increased FLOPS potential
   - Increased accuracy – Only a single rounding
3. Enhanced vectorization with Gather, Shifts and powerful permutes

Uses the same 256-bit YMM registers as Intel AVX

**Intel AVX2 completes the 256-bit extensions started with Intel AVX: 256-bit integer , cross-lane permutes, gather, FMA**

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Other Features of Haswell (wrt AVX2)

Improved cache bandwidth to feed wide vector units and FMAs

- 32-byte load/store for L1 -> 2X bandwidth
- 2x L2 bandwidth

2 new ports:

- additional ALU and new branch unit
- new AGU (address generation unit) for stores;

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Optimizing for Intel® AVX2

## Additional speedups come from:

- Wider SIMD integer instructions
- Fused multiply-add instructions (e.g. linear algebra)
- **Gather & permute instructions enable more vectorization for indirect referencing**

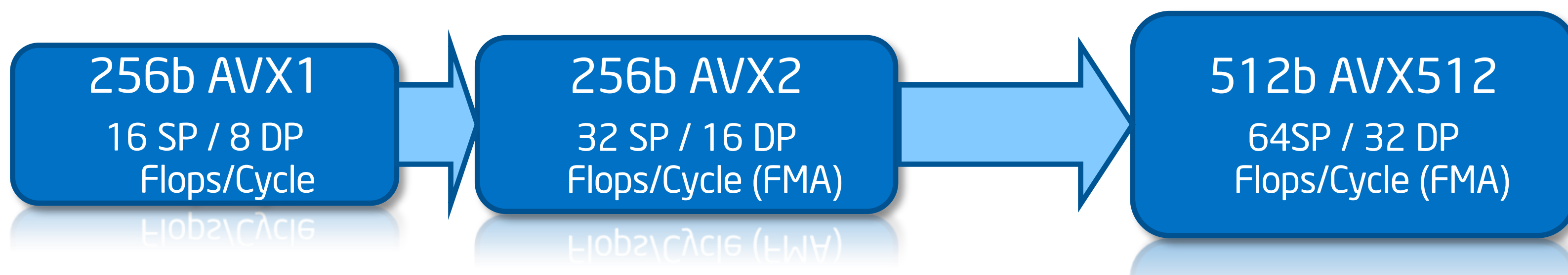
```
float *a, *b;
    for (i=0; i<imax; i++)
        if (ind[i]>=0 && ind[i]<
indmax)
            a[i] = b[ind[i]]
```

- More efficient general 32 byte loads (but still try to align data to 32 byte boundary)

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Intel® AVX512

# Intel® AVX Technology




| AVX | AVX2 |
|---|---|
| 256-bit basic FP | Float16 (IVB 2012) |
| 16 registers | 256-bit FP FMA |
| NDS (and AVX128) | 256-bit integer |
| Improved blend | PERMD |
| MASKMOV | Gather |
| Implicit unaligned | |

| AVX512 |
|---|
| 512-bit FP/Integer |
| 32 registers |
| 8 mask registers |
| Embedded rounding |
| Embedded broadcast |
| Scalar/SSE/AVX "promotions" |
| Transcendental support |
| Gather/Scatter |

**SNB**
2011

**HSW**
2013

**Future Processors (KNL & Future Xeon)**

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Math Support

Package to aid with Math library writing
- Good value upside in financial applications
- Available in PS, PD, SS and SD data types
- Great in combination with embedded RC

| Instruction (30) | | |
|---|---|---|
| $VGETXEXP_{\{PS,PD,SS,SD\}}$ | zmm1 {k1}, zmm2 | Obtain exponent in FP format |
| $VGETMANT_{\{PS,PD,SS,SD\}}$ | zmm1 {k1}, zmm2 | Obtain normalized mantissa |
| $VRNDSCALE_{\{PS,PD,SS,SD\}}$ | zmm1 {k1}, zmm2, imm8 | Round to scaled integral number |
| $VSCALEF_{\{PS,PD,SS,SD\}}$ | zmm1 {k1}, zmm2, zmm3 | $X*2^{y}$, X <= getmant, Y <= getexp |
| $VFIXUPIMM_{\{PS,PD,SS,SD\}}$ | zmm1, zmm2, zmm3, imm8 | Patch output numbers based on inputs |
| $VRCP14_{\{PS,PD,SS,SD\}}$ | zmm1 {k1}, zmm2 | Approx. reciprocal() with rel. error $2^{-14}$ |
| $VRSQRT14_{\{PS,PD,SS,SD\}}$ | zmm1 {k1}, zmm2 | Approx. rsqrt() with rel. error $2^{-14}$ |
| $VDIV_{\{PS,PD,SS,SD\}}$ | zmm1 {k1}, zmm2, zmm3 | IEEE division |
| $VSQRT_{\{PS,PD,SS,SD\}}$ | zmm1 {k1}, zmm2 | IEEE square root |

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Why True Masking?

## Memory fault suppression

- Vectorize code without touching memory that the correspondent scalar code would not touch
  - Typical examples are if-conditional statements or loop remainders
  - AVX is forced to use VMASKMOV* (risc)

## MXCSR flag updates and fault handlers

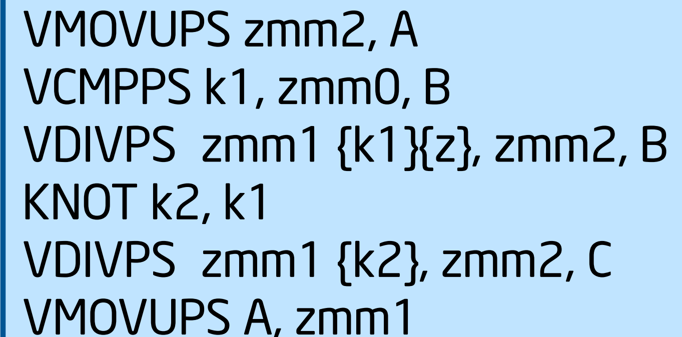
- Avoid spurious floating-point exceptions without having to inject neutral data

## Zeroing/merging

- Use zeroing to avoid false dependencies in OOO architecture
- Use merging to avoid extra blends in if-then-else clauses (predication) for great code density

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

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




Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Back-up

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Support for data divergence

```
for (j = 0; j < atom.ncnt; j++ ) {    // neighbors
       neigh = X[atom.nlist[j]];      // indirect
       t = compute_interactions(atom, neigh)
       force += t           I
}
```

| | |
|---|---|
| 4th Generation Intel© Core™ (Haswell) | Gather instruction, register shuffles |
| Intel© Phi™ Coprocessor (Knights Corner) | Gather scatter engine, register shuffles |
| Intel© Atom™ Processor Z300 (Silvermont) | Register shuffles |
| Intel© HD Graphics (GEN) | Full gather/scatter to register file, local memory, globabl memory |

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Support for control flow divergence

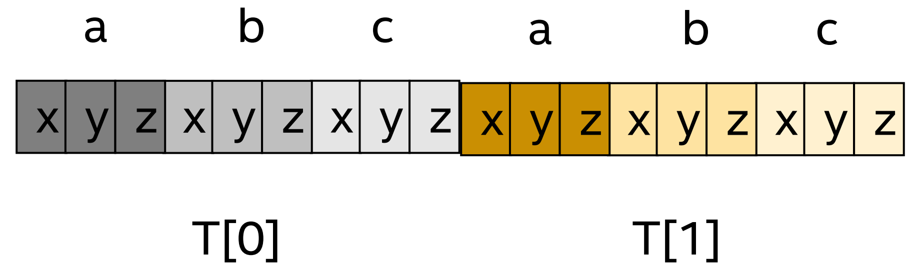
```
for (int i = 0; i < n; i++ ){
        int c = count[i];
         while (c-- > 0) values[i] *= values[i];
}
```

| | |
|---|---|
| 4th Generation Intel© Core™ (Haswell) | blending |
| Intel© Phi™ Coprocessor (Knights Corner) | Mask registers |
| Intel© Atom™ Processor Z300 (Silvermont) | Blending |
| Intel© HD Graphics (GEN) | "channel masking" (control flow per lane) Predication |

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Object-oriented programming

```
Class Point {float x,y,z;}
Class Triangle {Point a,b,c;}
Triangle T[100];
```




```
Point Cross( const Point& a, const Point& b ) {
    return Point( a.y*b.z-a.z*b.y, a.z*b.x-a.x*b.z,
a.x*a.y-a.y-b.x );
}

void ComputeNormals( Point normal[__restrict], const
Triangle p[], size_t n )
    for( size_t i=0; i<n; ++i )
        normal[i] = Cross( p[i].b-p[i].a, p[i].c-p[i].a );
}
```

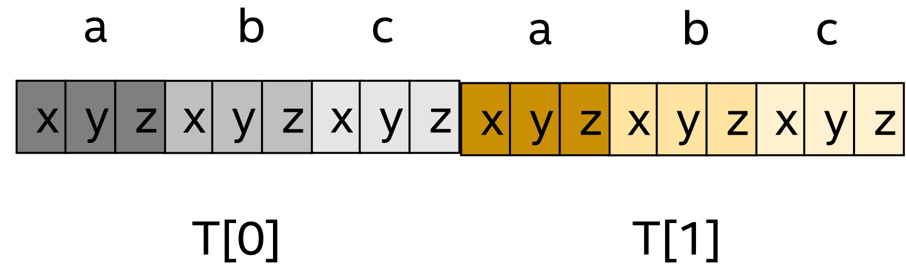
**Object oriented programming may inhibit SIMD code generation**



Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

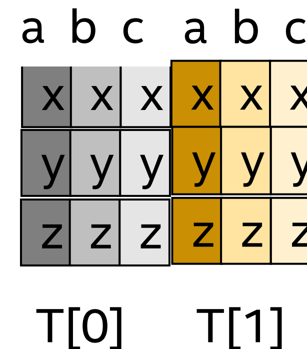
# Object-oriented programming, 2

```
Class Point {float
x,y,z;}
Class Triangle {Point
a,b,c;}
Triangle T[100];
```




Limited language support for "structure of arrays"

```
Struct {
      float x[100];
      float y[100];
      float z[100];
} T;
```




**Problem: OOP is easier for the programmer and more efficient if access is not-linear (cache locality)**
**Solution - consider "AoSoA"**


Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Example of Outer Loop Vectorization

```
#pragma omp declare simd
int lednam(float c)
{   // Compute n >= 0 such that c^n > LIMIT
    float z = 1.0f; int iters = 0;
    while (z < LIMIT) {
        z = z * c; iters++;
    }
    return iters;
}
```

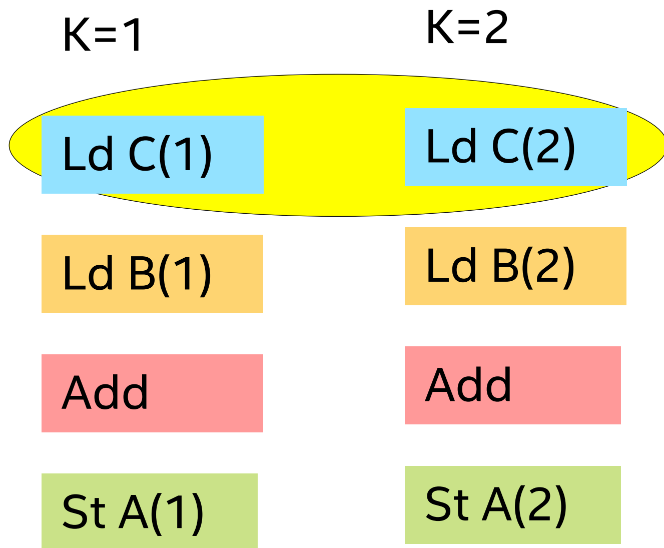
```
float in_vals[];
#pragma omp simd
for(int x = 0; x < Width; ++x) {
    count[x] = lednam(in_vals[x]);
}
```

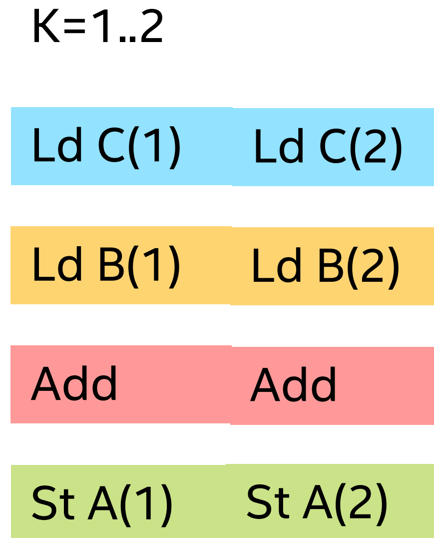
| x = 0 | x = 1 | x = 2 | x = 3 |
|---|---|---|---|
| z = z * c | z = z * c | z = z * c | z = z * c |
| z = z * c | z = z * c | z = z * c | z = z * c |
| | .... | .................. | ........ |
| iters = 2 | iters = 23 | iters = 255 | iters = 37 |


Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.
Optimization Notice

# Vectorization yesterday

```
      DO 1 k = 1,n
1       A(k) = B(k) + C(k)
```




Scalar code

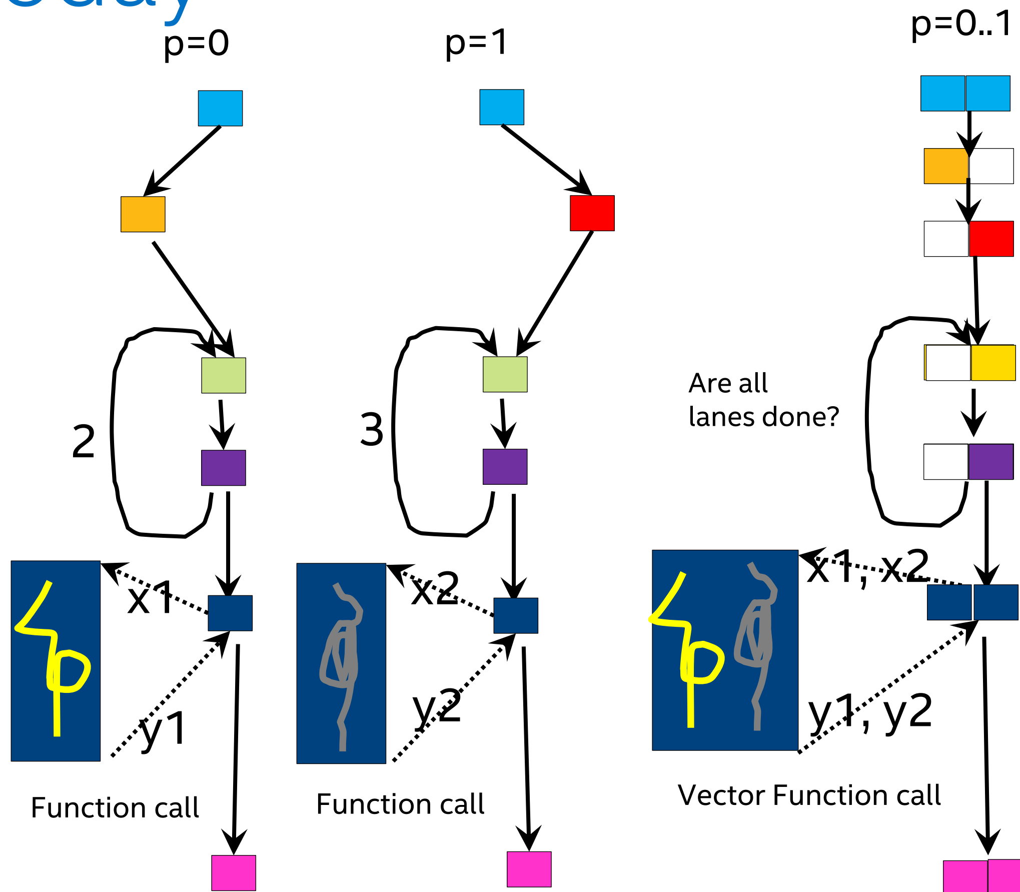
Vector code

Vector code generation was straightforward
Emphasis on analysis and disambiguation

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Vectorization today

```
#pragma omp simd reduction(+:....)
for(p=0; p<N; p++) {
  // Blue work
  if(...) {
     // Green work
  } else {
     // Red work
  }
  while(...) {
     // Gold work
     // Purple work
  }
  y = foo (x);
  Pink work
}
```

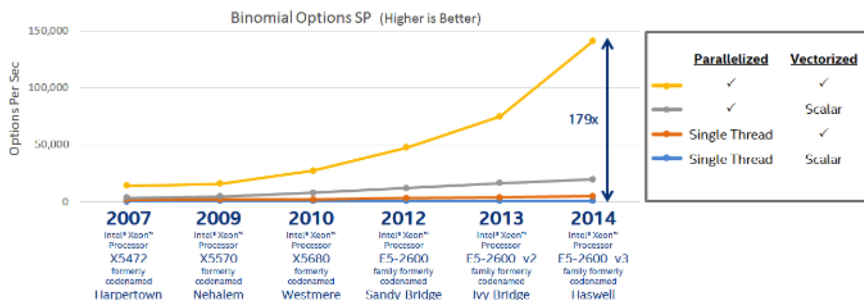



Two fundamental problems
- Data divergence
- Control divergence

Vector code generation has become a more difficult problem
Increasing need for user guided explicit vectorization
Explicit vectorization maps threaded execution to simd hardware

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Configurations for Binomial Options SP




**Optimization Notice**
Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Performance measured in Intel Labs by Intel employees

## Platform Hardware and Software Configuration

| Platform | Unscaled Core Frequency | Cores/ Socket | Num Sockets | L1 Data Cache | L1 I Cache | L2 Cache | L3 Cache | Memory | Memory Frequency | Memory Access | H/W Prefetchers Enabled | HT Enabled | Turbo Enabled | C States | O/S Name | Operating System | Compiler Version |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Intel® Xeon™ 5472 Processor | 3.0 GHZ | 4 | 2 | 32K | 32K | 12 MB | None | 32 GB | 800 MHZ | UMA | Y | N | N | Disabled | Fedora 20 | 3.11.10-301.fc20 | icc version 14.0.1 |
| Intel® Xeon™ X5570 Processor | 2.93 GHZ | 4 | 2 | 32K | 32K | 256K | 8 MB | 48 GB | 1333 MHZ | NUMA | Y | Y | Y | Disabled | Fedora 20 | 3.11.10-301.fc20 | icc version 14.0.1 |
| Intel® Xeon™ X5680 Processor | 3.33 GHZ | 6 | 2 | 32K | 32K | 256K | 12 MB | 48 MB | 1333 MHZ | NUMA | Y | Y | Y | Disabled | Fedora 20 | 3.11.10-301.fc20 | icc version 14.0.1 |
| Intel® Xeon™ E5 2690 Processor | 2.9 GHZ | 8 | 2 | 32K | 32K | 256K | 20 MB | 64 GB | 1600 MHZ | NUMA | Y | Y | Y | Disabled | Fedora 20 | 3.11.10-301.fc20 | icc version 14.0.1 |
| Intel® Xeon™ E5 2697v2 Processor | 2.7 GHZ | 12 | 2 | 32K | 32K | 256K | 30 MB | 64 GB | 1867 MHZ | NUMA | Y | Y | Y | Disabled | Fedora 20 | 3.11.10-301.fc20 | icc version 14.0.1 |
| Codename Haswell | 2.2 GHz | 14 | 2 | 32K | 32K | 256K | 35 MB | 64 GB | 2133 MHZ | NUMA | Y | Y | Y | Disabled | Fedora 20 | 3.13.5-202.fc20 | icc version 14.0.1 |

Copyright © 2015 Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2015v, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

**Optimization Notice**

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

Notice revision #20110804


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