



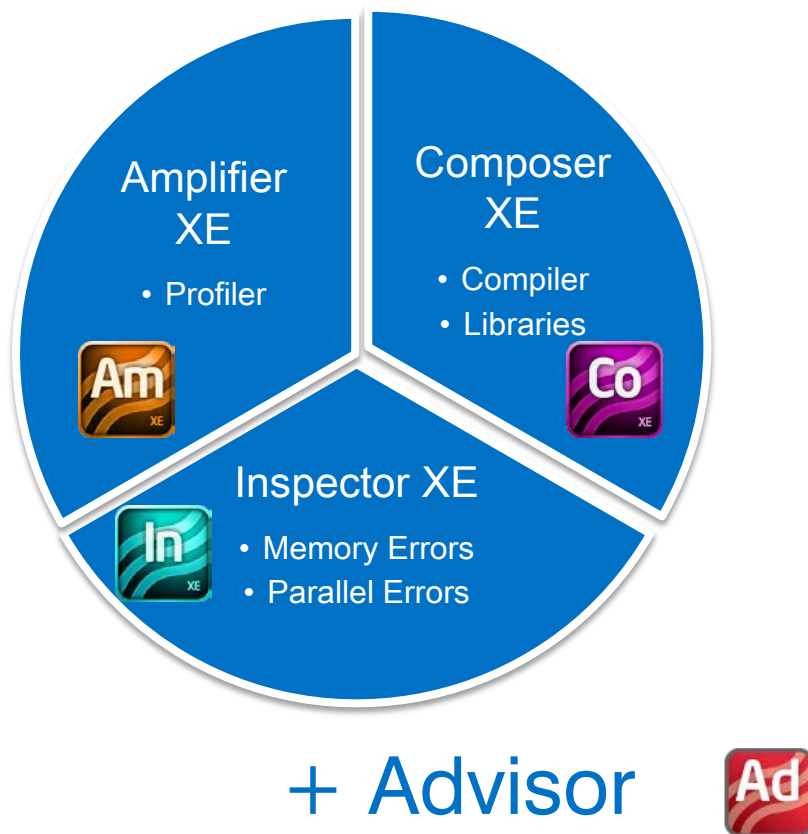
Programming for Intel® Xeon Phi™

Stephen Blair-Chappell, Intel

Essential Requirements for Programming on the Intel® Xeon Phi™ Coprocessor.

In this session, the ingredients for successful Xeon Phi™ Programming are discussed. We look at the experience and results of porting an application to run on the Intel® Xeon Phi™. We look at: What went well in the project; How difficult the project porting was; Useful tips and tricks; A comparison of performance on Xeon Phi™ and non-Xeon Phi™ platforms.

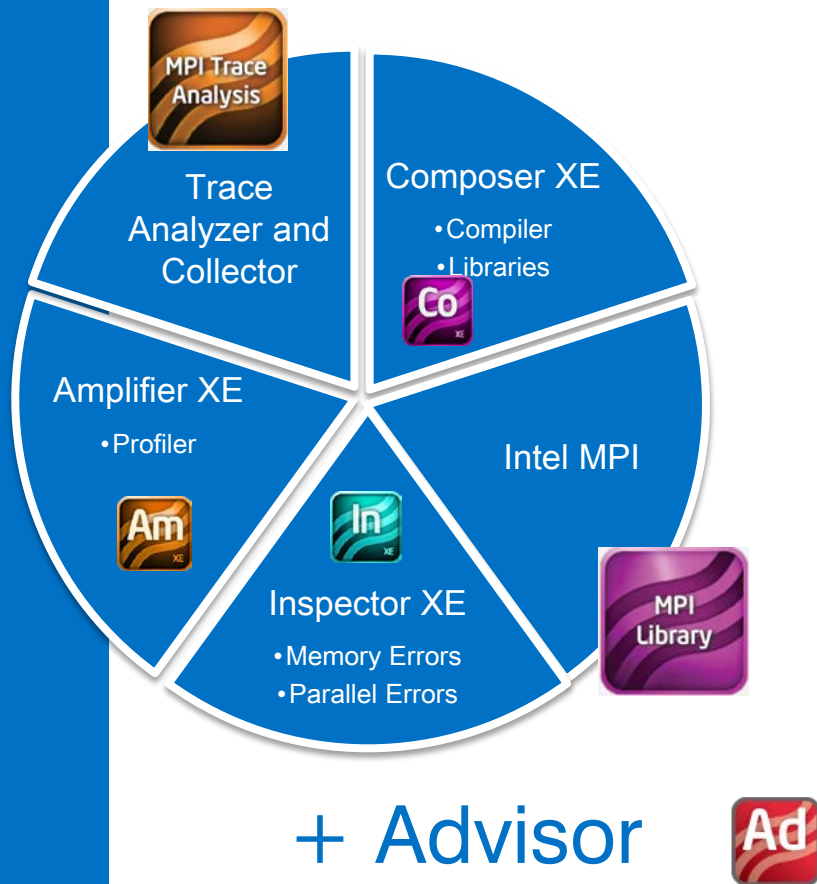
Intel® Parallel Studio XE



- **Intel® Parallel Advisor**
Use to model parallelism in your existing applications
- **Intel® Composer XE**
Use to generate fast, safe, parallel code (C/C++, Fortran)
- **Intel® VTune™ Amplifier XE**
Find hotspots and bottlenecks in your code.
- **Intel® Inspector XE**
Use to find memory and threading errors

Four Components

® Cluster Studio XE



Four Six Components

- **Intel® Parallel Advisor**
Use to model parallelism in your existing applications
- **Intel® Composer XE**
Use to generate fast, safe, parallel code (C/C++, Fortran)
- **Intel® VTune™ Amplifier XE**
Find hotspots and bottlenecks in your code.
- **Intel® Inspector XE**
Use to find memory and threading errors
- **Intel® MPI**
Industry standard message passing interface library for parallelism across clusters
- **Intel® Trace Analyzer and Collector (ITAC)**
Examine runtime behaviour of programs running on clusters.

Code must be

highly Parallel

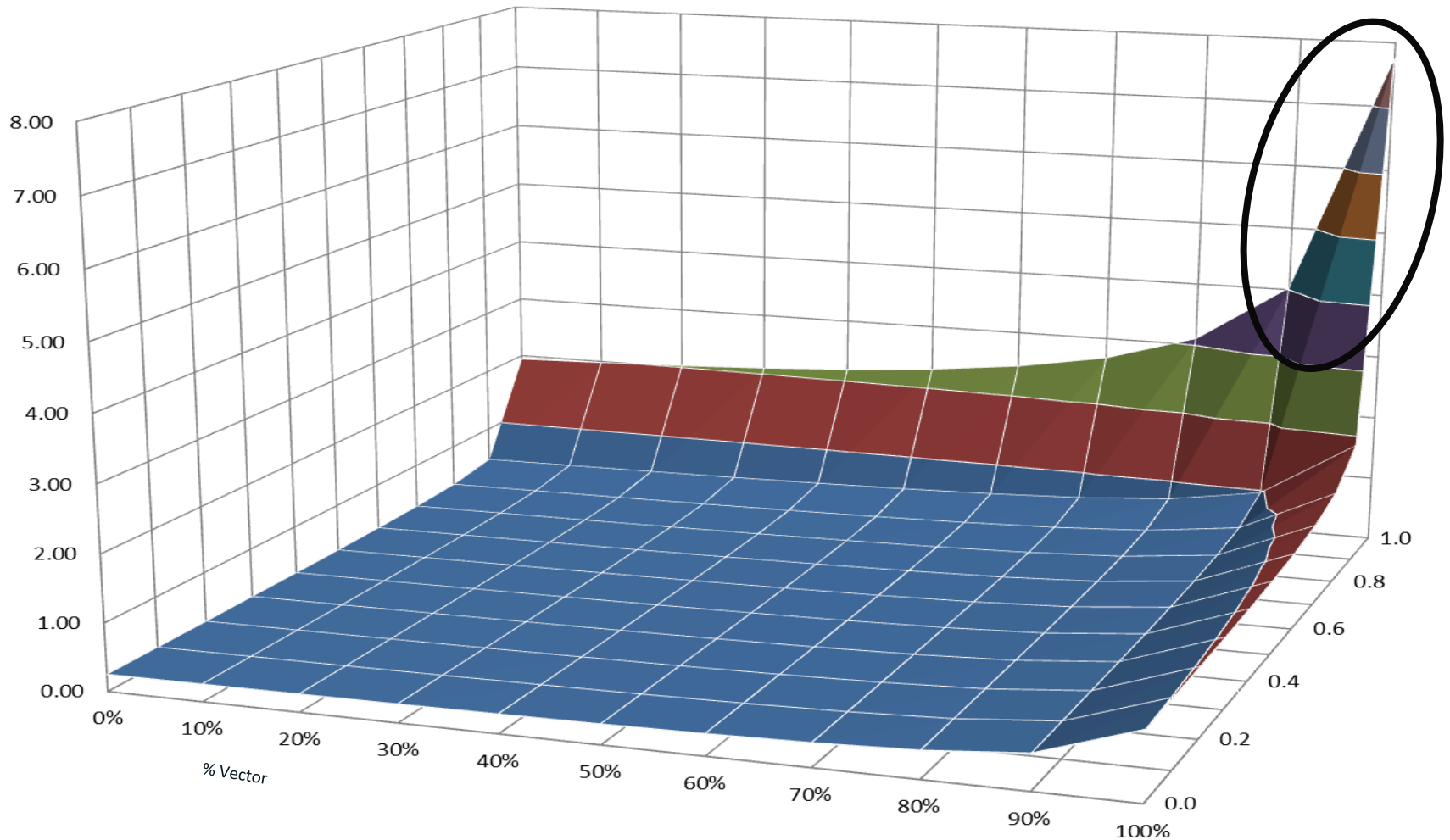
effectively Vectorised



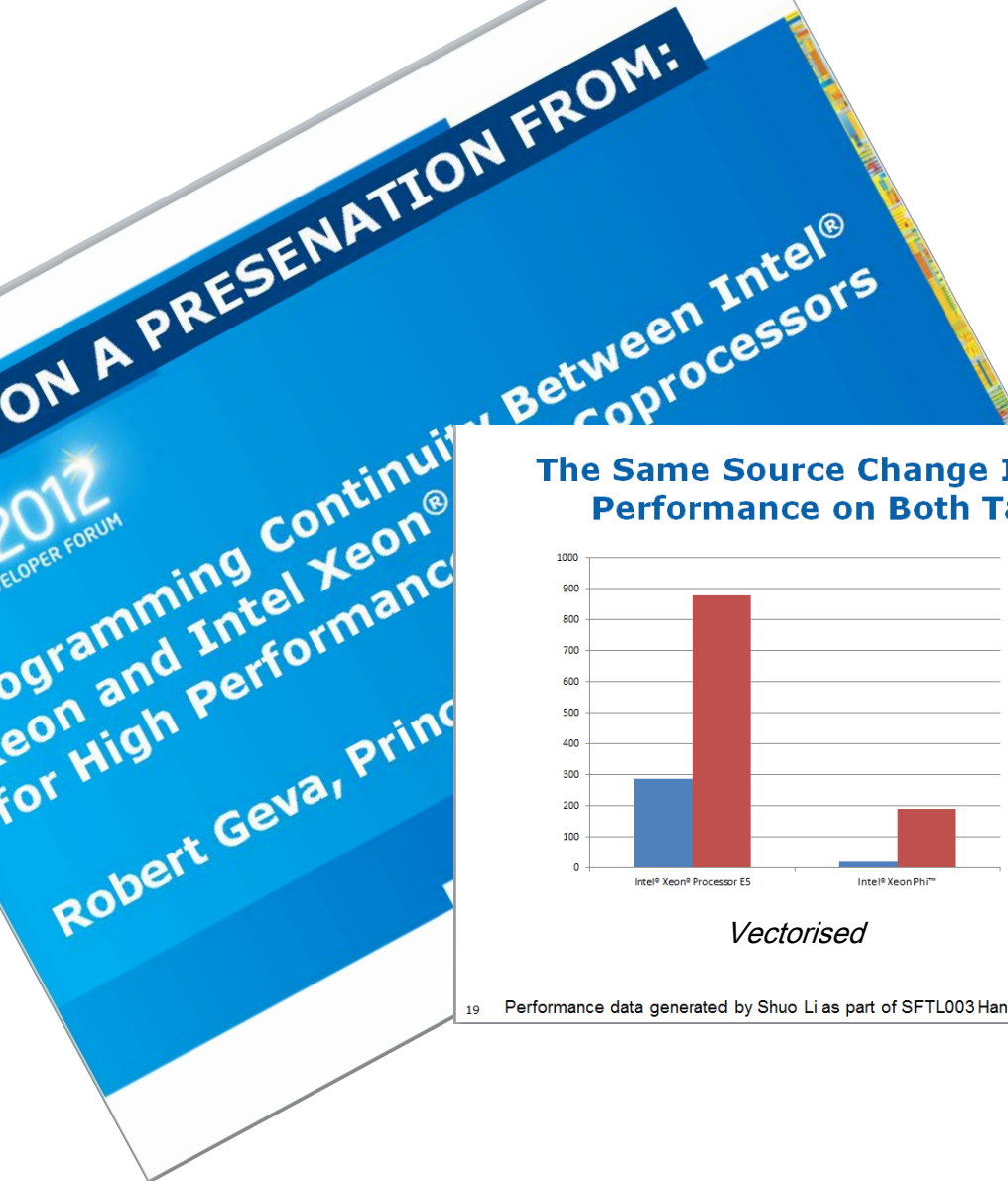
Application Performance: Intel® Xeon Phi™ Coprocessor

Ratio KNC/E5 Peak Performance (per processor)

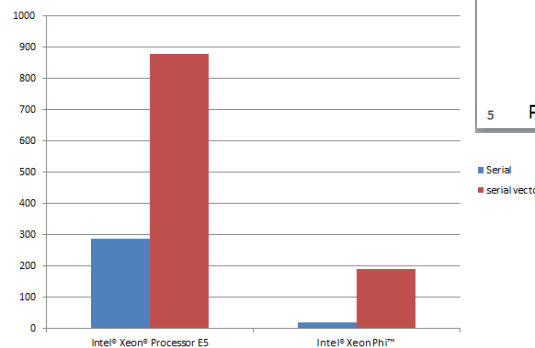
■ 0.00-1.00 ■ 1.00-2.00 ■ 2.00-3.00 ■ 3.00-4.00 ■ 4.00-5.00 ■ 5.00-6.00 ■ 6.00-7.00 ■ 7.00-8.00



<http://www.intel.com/performance>



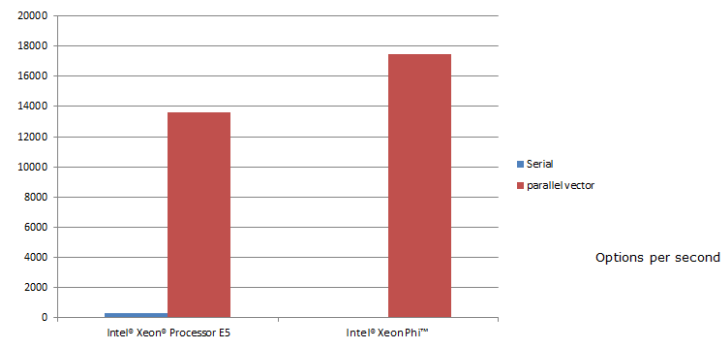
The Same Source Change Improves Performance on Both Targets



Vectorised

19 Performance data generated by Shuo Li as part of SFTL003 Hands On Lab

The Same Source Change Improves Performance on Both Targets



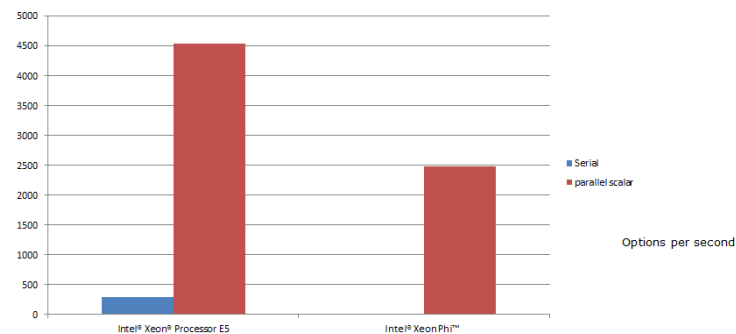
Parallelization and vectorization together improve option per second by > 800X and by >50X

HOW DO WE GET THERE?

5 Performance data generated by Shuo Li as part of SFTL003 Hands On Lab

IDF2012
INTEL DEVELOPER FORUM

The Same Source Change Improves Performance on Both Targets



Parallel

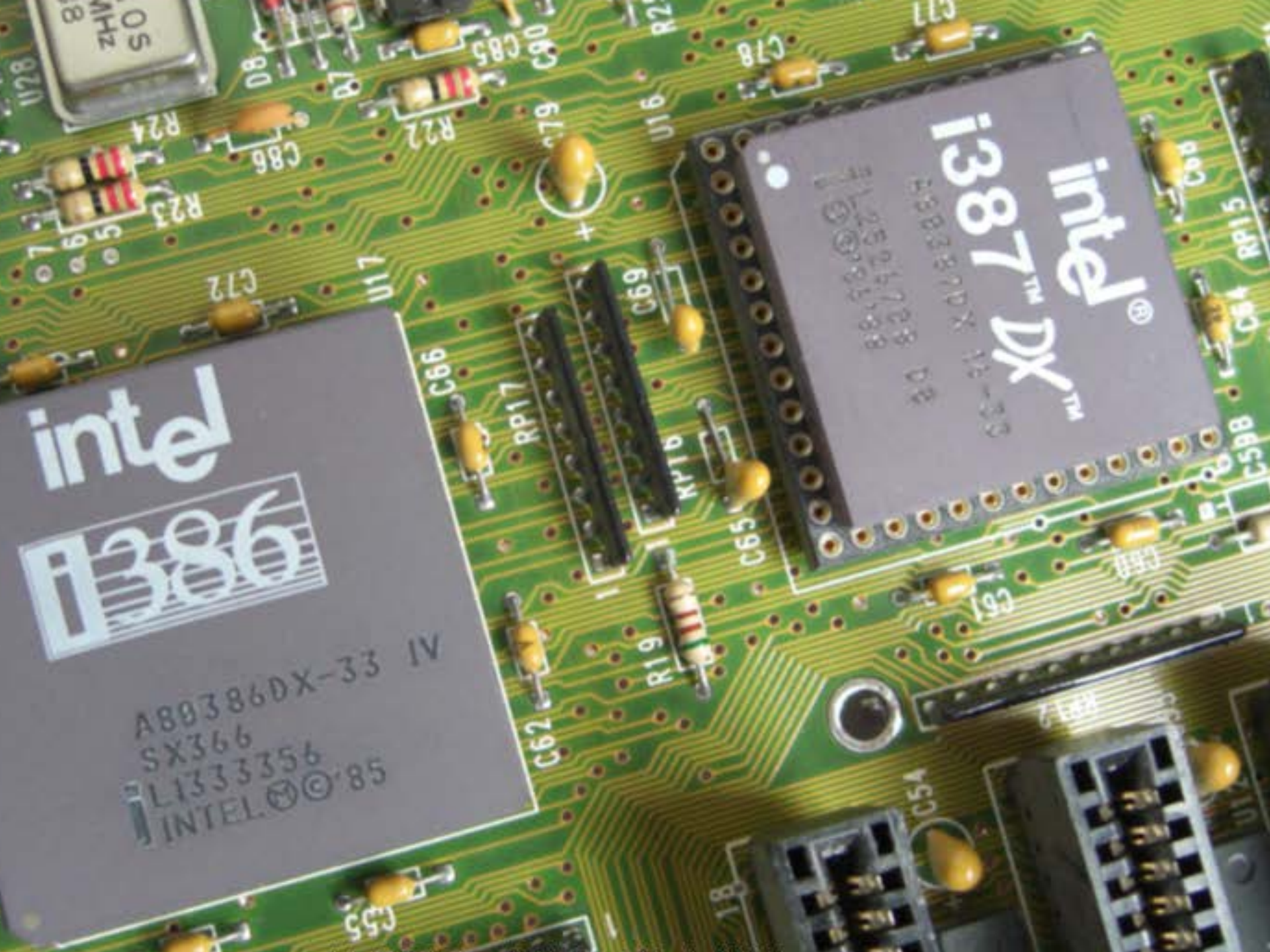
7 Performance data generated by Shuo Li as part of SFTL003 Hands On Lab

IDF2012
INTEL DEVELOPER FORUM

On the graphs, bigger is better



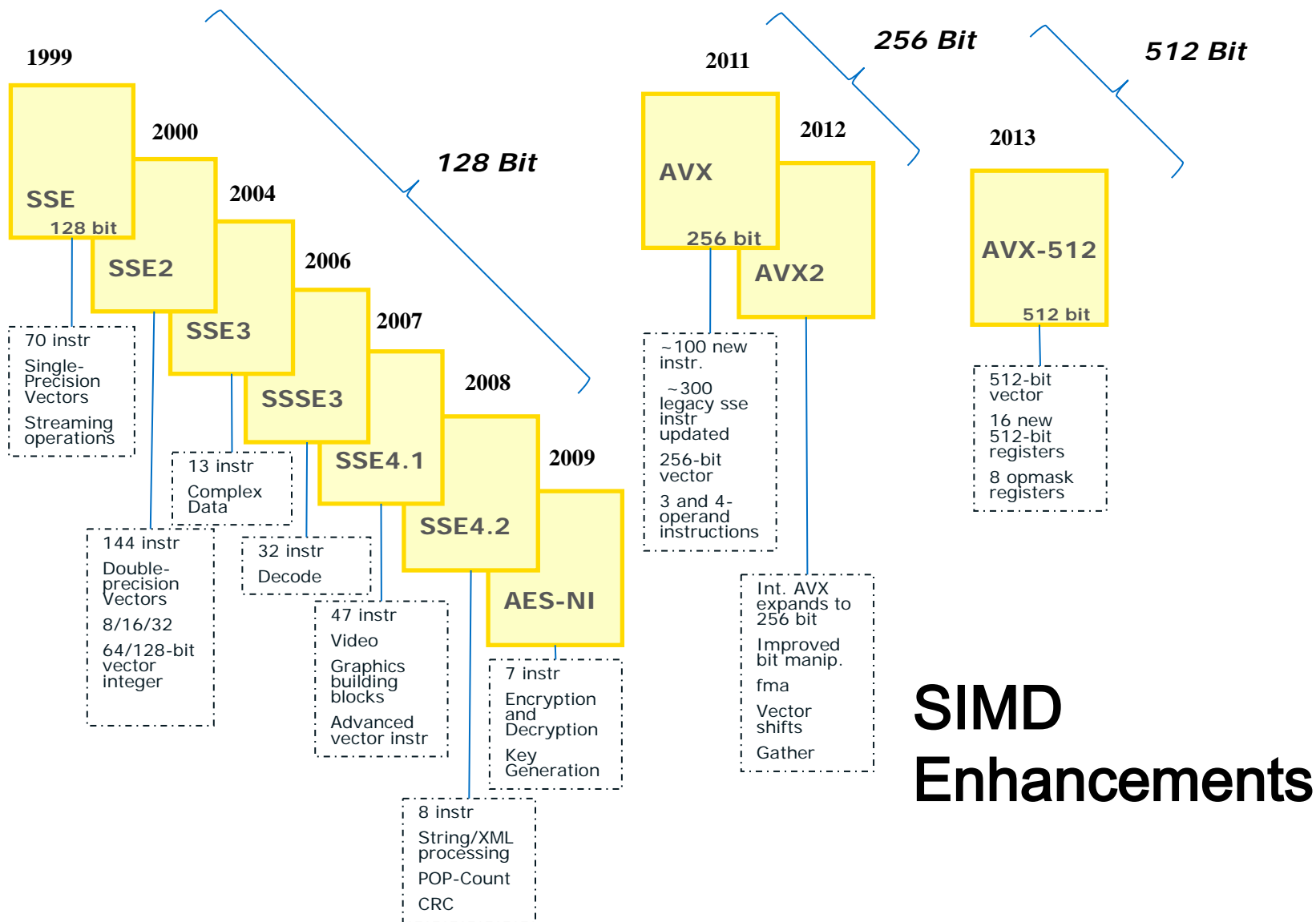
Vectorisation



intel
386

A80386DX-33 IV
SX366
L1333356
INTEL © '85

intel
387DX
883875612
883875612
883875612
883875612

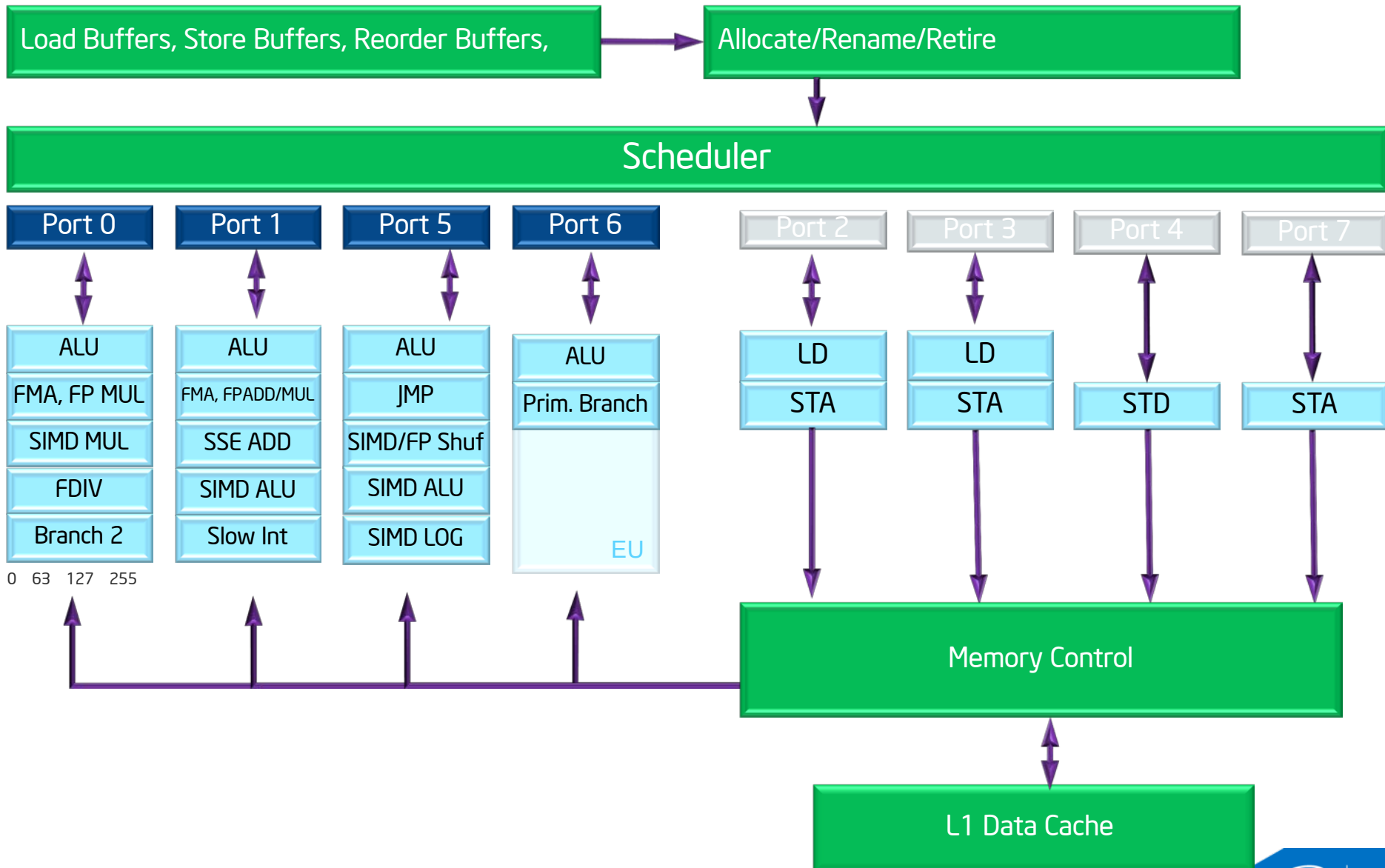


SIMD Enhancements

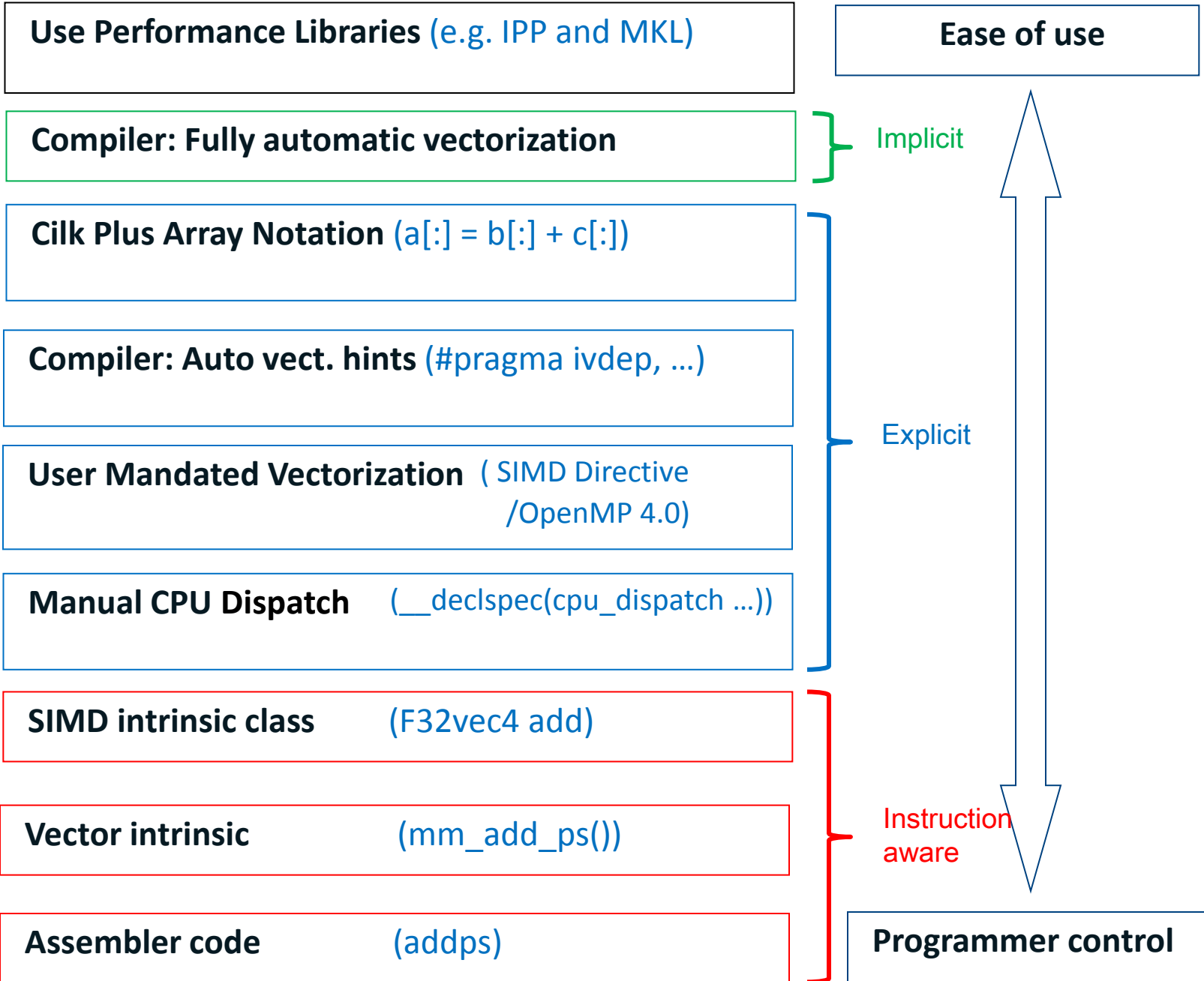


4th Generation Intel® Core™ processor family

Execution Units



Ways of Inserting Vectorised Code



Overview of Writing Vector Code

Array Notation

```
A[:] = B[:] + C[];
```

SIMD Directive

```
#pragma simd  
for (int i = 0; i < N; ++i) {  
    A[i] = B[i] + C[i];  
}
```

Elemental Function

```
__declspec(vector)  
float ef(float a, float b) {  
    return a + b;  
}  
  
A[:] = ef(B[:], C[:]);
```

Auto-Vectorization

```
for (int i = 0; i < N; ++i) {  
    A[i] = B[i] + C[i];  
}
```

Explicit Vector Programming with SIMD Pragma/Directive

Programmer asserts:

**p* is loop invariant

A[] does not overlap with *B[]* or *C[]*

sum not aliased with *B[]* or *C[]*

sum should be treated as a reduction

Allow compiler to reorder for better vectorization

Vector code should be generated even if efficiency heuristic does not indicate a gain in performance

```
#pragma omp simd reduction(+:sum)
for(i = 0; i < *p; i++) {
    A[i] = B[i] * C[i];
    sum = sum + A[i];
}
```

Explicit vector programming
lets you express what you mean!

How do I know if a loop is vectorised?

- -vec-report

```
> icl /Qvec-report MultArray.c  
MultArray.c(92): (col. 5) remark: LOOP WAS VECTORIZED.
```

Qvec-report1 (default)

Qvec-report2

Qvec-report3

Qvec-report4

Qvec-report5

Qvec-report6

```
[sbc@snbws3 lab-01]$ vecanalysis.py --annotate vec7.txt
```

```
Writing pi_vr.c ... done
```

Statistics for all files

Message

```
vector loop cost: 32.000000.
```

```
type converts: 2.
```

```
unroll factor set to 4.
```

LOOP WAS VECTORIZED.

```
loop inside vectorized loop at nesting level: 1
```

```

loop inside vectorized loop at position 1
remainder loop was not vectorized: 1

```

```
vector loop cost: 59.000000.
```

```
medium-overhead vector operations: 2
```

heavy-overhead vector operations: 2

conversion from int to float will be emulated.

```
scalar loop cost: 52.
```

```
loop was vectorized (no peel/with remainder)
```

```
estimated potential speedup: 6.010000.
```

lightweight vector operations: 32.

lightweight vector operations: 34.

```
estimated potential speedup: 3.150000.
```

divides: 1.

REMAINDER LOOP WAS VECTORIZED.

```
remainder loop was vectorized (masked)
```

Total Source Locations:

Source Locations

Count	%
-------	---

2 66.7%

1	66.7%
2	66.7%

2 66.7%

1	66.7%
2	66.7%

2	66.7%
---	-------

2 66.7%

2 66.7%

1	66.7%
2	66.7%

2 66.7%

1	66.7%
2	66.7%

2 66.7%

2 66.7%

2 66.7%

2 66.7%

2 66.7%

2 66.7%

2 66.7%

2 66.7%

2 66.7%

3

Lightweight Hotspots - Hardware Issues with Vectorization Info

Analysis Target Analysis Type Summary Bottom-up Top

Grouping: Function / Call Stack

Function / Call Stack	Hardware Ev ... CPU_CL...	Hardware Ev ... INST_RETIRE...	CP ... Rate	Source File	Vector Instruction Set	Vector Instruction Class
[Loop@0x40da56 at line 581 in grid_intersect]	1,402,220,698	1,461,822,066	0.9	...he 581 in g ... grid.cpp	SSE2	MOVSD_XMM
[Loop@0x40d9a0 at line 559 in grid_intersect]	227,291,360	221,977,048	1.0	...he 559 in g ... grid.cpp	SSE2	ADDSD; COMISD; MOVSD_XMM
[Loop@0x40dad0 at line 600 in grid_intersect]	67,278,616	49,820,136	1.0	...he 600 in g ... grid.cpp	SSE2	MOVSD_XMM
[Loop@0x40d9d0 at line 562 in grid_intersect]	53,850,098	45,107,993	1.0	...he 562 in g ... grid.cpp	SSE2	MOVSD_XMM
[Loop@0x4078c4 at line 111 in shader]	20,371,513	22,348,421	0.9	...e 111 in s ... shade.cpp	SSE2	ADDSD; COMISD; DIVSD; MOVAPD; MOVQ; MOVSD_XMM
[Loop@0x40d0af at line 193 in cellbound]	4,757,400	2,802,192	1.698	find_hotspots.exe [Loop@0x40d0af at line 193 in ce ... grid.cpp	SSE2	COMISD; MOVAPD; MOVSD_XMM
[Loop@0x6ff5a048 in func@0x6ff5a02e]	4,253,395	2,068,814	2.056	msvcrt.dll [Loop@0x6ff5a048 in func@0x6ff ... [Unknown source...	SSE2	MOVDQA
[Loop@0x6ff59c7f in func@0x6ff59c67]	0	0	0.000	msvcrt.dll [Loop@0x6ff59c7f in func@0x6ff ... [Unknown source...	SSE2	MOVDQA
[Loop@0x40ce99 at line 155 in globalbound]	0	0	0.000	find_hotspots.exe [Loop@0x40ce99 at line 155 in gl ... grid.cpp	SSE2	COMISD; MOVAPD; MOVSD_XMM
[Loop@0x4031c0 at line 165 in shadow_intersection]	0	2,142,765	0.000	find_hotspots.exe [Loop@0x4031c0 at line 165 in s ... intersect.cpp	SSE2	COMISD

Select viewpoint:

Hardware Event Counts

Hardware Event Sample Counts

Hardware Issues

Hotspots

Extended Sleep States

Hardware Issues with Vectorization Info

Hotspots with Vectorization Info

0.5s 1s 1.5s 2s 2.5s 3s 3.5s 4s 4.5s 5s 5.5s 6s 6.5s 7s 7.5s 8s 8.5s 9s 9.5s 10s 10.5s 11s

Thread (0x2154)
Thread (0x478c)

Hardware Events

Thread

Context

Hardware

Hardware

Hardware

Hardware

Hardware

Hardware

Hardware

Hardware

Hardware

Hardware

Hardware

Hardware

Hardware

Hardware

Hardware

Hardware

Hardware

Hardware

Hardware

Hardware

Hardware

Filter: 10.4% is shown

Process: Any Process

Thread: Any Thread

Any Vector Instruction Class

Any Module

Timeline Hardware Event: BR_INST_RETIRED.NEAR_TAKEN

Call Stack Mode: User/system functions

Inline Mode: on

Loop Mode: Loops only

[10.4%] SSE2

Any Vector Instruction Set

[89.6%] [Unknown]

[10.4%] SSE2

[0.0%] SSE

Analysis Target Analysis Type Summary Bottom-up Top

Grouping: Function / Call Stack

Function / Call Stack	Hardware Ev ...	Hardware Ev ...	CP	Source File	Vector Instruction Set	Vector Instruction Class
[Loop@0x40da56 at line 581 in grid_intersect]	1,402,220,698	1,461,822,066	0.9	he 581 in g ... grid.cpp	SSE2	MOVSD_XMM
[Loop@0x40d9a0 at line 559 in grid_intersect]	227,291,360	221,977,048	1.0	he 559 in g ... grid.cpp	SSE2	ADDSD; COMISD; MOVSD_XMM
[Loop@0x40da56 at line 581 in grid_intersect]	67,338,616	40,939,136	1.0	he 581 in g ... grid.cpp	SSE2	MOVSD_XMM

Select viewpoint:

- Hardware Event Counts
- Hardware Event Sample Counts
- Hardware Issues
- Hotspots
- Extended Sleep States

The feature is enabled in AXE 2013 Update 4

```
set AMPLXE_EXPERIMENTAL=vectinfo
amplxe-cl -collect lightweight-hotspots -knob enable-stack-collection=true -- application.exe
amplxe-cl -R hw-events-vect
```

Thread

Hardware Events

Filter: 10.4% is shown

Process: Any Process Thread: Any Thread Any Vector Instruction Class Any Module [10.4%] SSE2

Timeline Hardware Event: BR_INST_RETIRED.NEAR_TAKEN Call Stack Mode: User/system functions Inline Mode: on Loop Mode: Loops only

[89.6%] [Unknown]

[10.4%] SSE2

[0.0%] SSE

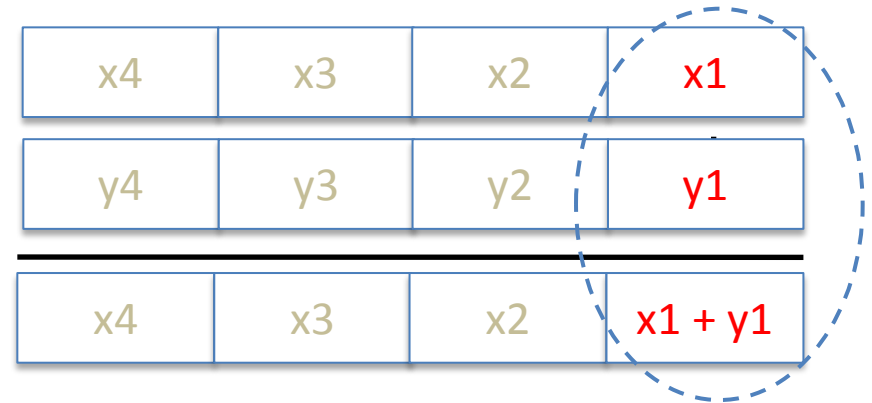
Scalar and Packed Instructions

add ss Scalar Single-FP Add



Single precision FP data

Scalar execution mode

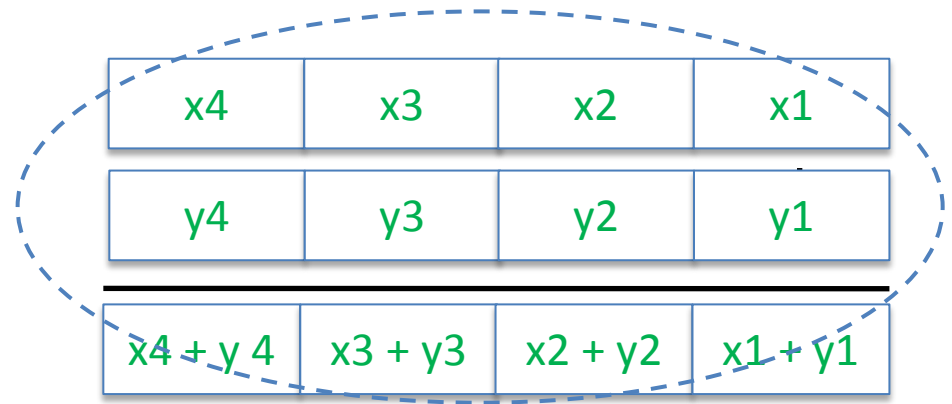


add ps Packed Single-FP Add



Single precision FP data

Packed execution mode



Examples of Code Generation

```
static double A[1000], B[1000],  
              C[1000];  
void add() {  
    int i;  
    for (i=0; i<1000; i++)  
        if (A[i]>0)  
            A[i] += B[i];  
        else  
            A[i] += C[i];  
}
```

```
.B1.2::  
    vmovaps    ymm3, A[rdx*8]  
    vmovaps    ymm1, C[rdx*8]  
    vcmpgtpd   ymm2, ymm3, ymm0  
    vblendvpd  ymm4, ymm1, B[rdx*8], ymm2  
    vaddpd     ymm5, ymm3, ymm4  
    vmovaps    A[rdx*8], ymm5  
    add        rdx, 4  
    cmp        rdx, 1000  
    jl         .B1.2
```

AVX

```
.B1.2::  
    movaps     xmm2, A[rdx*8]  
    xorps      xmm0, xmm0  
    cmpltprd   xmm0, xmm2  
    movaps     xmm1, B[rdx*8]  
    andps      xmm1, xmm0  
    andnps     xmm0, C[rdx*8]  
    orps       xmm1, xmm0  
    addpd      xmm2, xmm1  
    movaps     A[rdx*8], xmm2  
    add        rdx, 2  
    cmp        rdx, 1000  
    jl         .B1.2
```

SSE2

```
.B1.2::  
    movaps     xmm2, A[rdx*8]  
    xorps      xmm0, xmm0  
    cmpltprd   xmm0, xmm2  
    movaps     xmm1, C[rdx*8]  
    blendvpd   xmm1, B[rdx*8], xmm0  
    addpd      xmm2, xmm1  
    movaps     A[rdx*8], xmm2  
    add        rdx, 2  
    cmp        rdx, 1000  
    jl         .B1.2
```

SSE4.1

“Loop was not vectorized” because:

- “Existence of vector dependence”
- “Non-unit stride used”
- “Mixed Data Types”
- “Condition too Complex”
- “Condition may protect exception”
- “Low trip count”
- “Subscript too complex”
- “Unsupported Loop Structure”
- “Contains unvectorizable statement at line XX”
- “Not Inner Loop”
- “vectorization possible but seems inefficient”
- “Operator unsuited for vectorization”



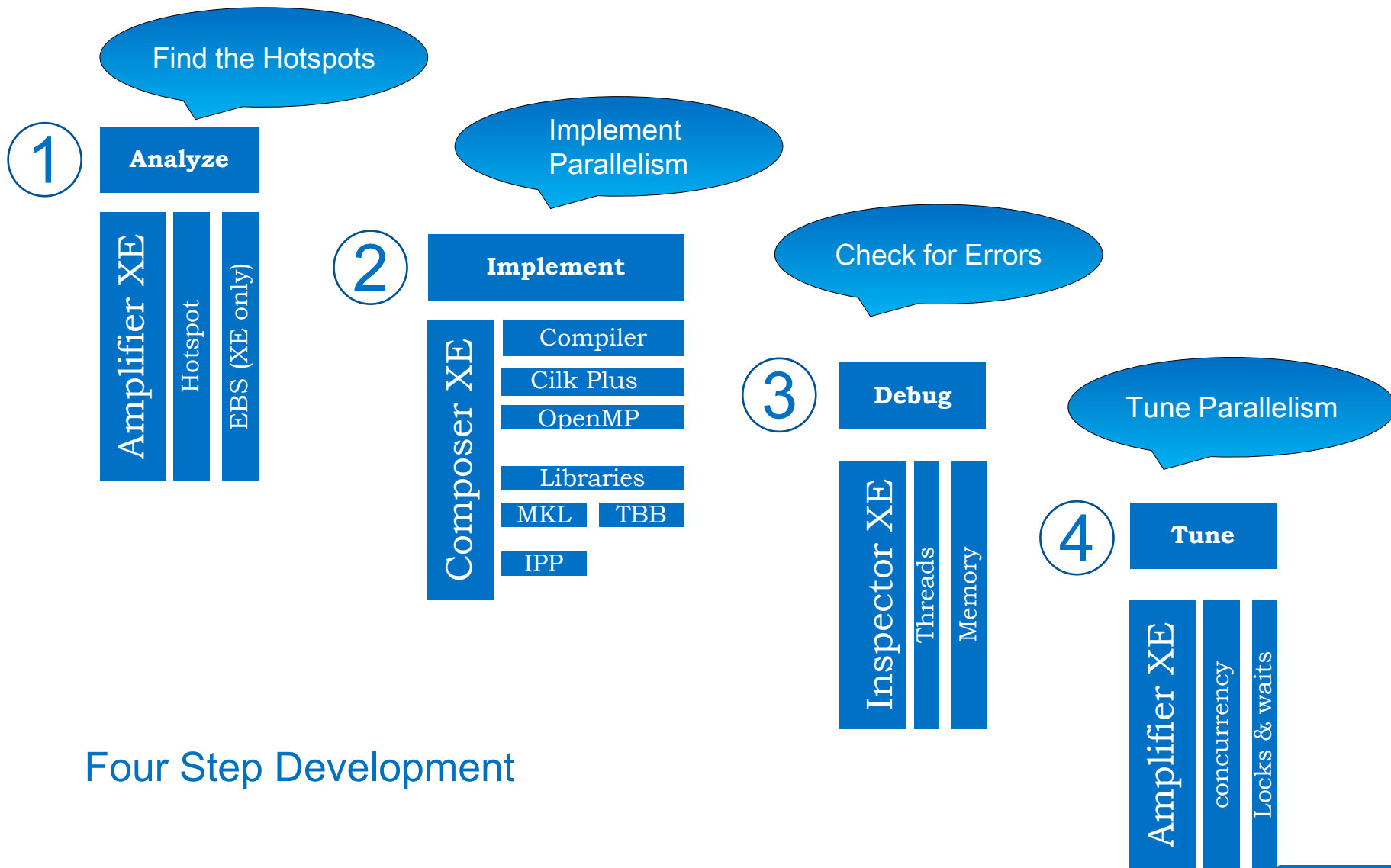
e.g. function calls



Parallelism

Speedup using parallelism

Parallel Code



Four Step Development

Language to help parallelism

Parallel Code

Intel® Cilk™ Plus

OpenMP

```
#pragma omp parallel for
for(i=1;i<=4;i++) {
    printf("Iter: %d", i);
}
```

Intel® Threading Building Blocks

Intel® MPI

Fortran Coarrays

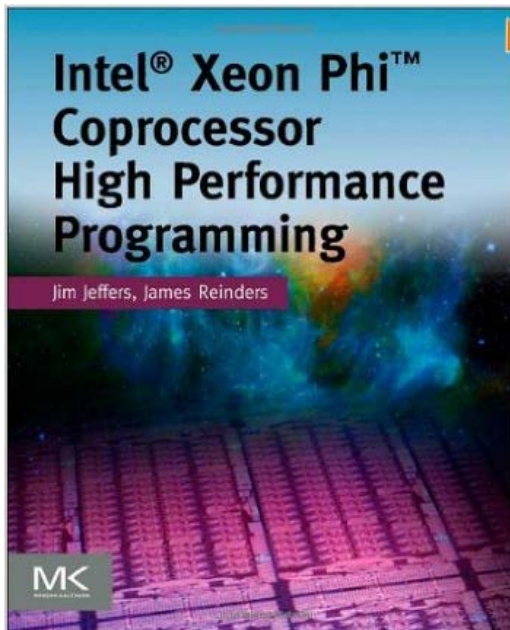
OpenCL

Native Threads

```
cilk_for (int i = 0; i < max_row; i++)
{
    for (int j = 0; j < max_col; j++ )
    {
        p[i][j] = mandel( complex(scale(i), scale(j)));
    }
}
```

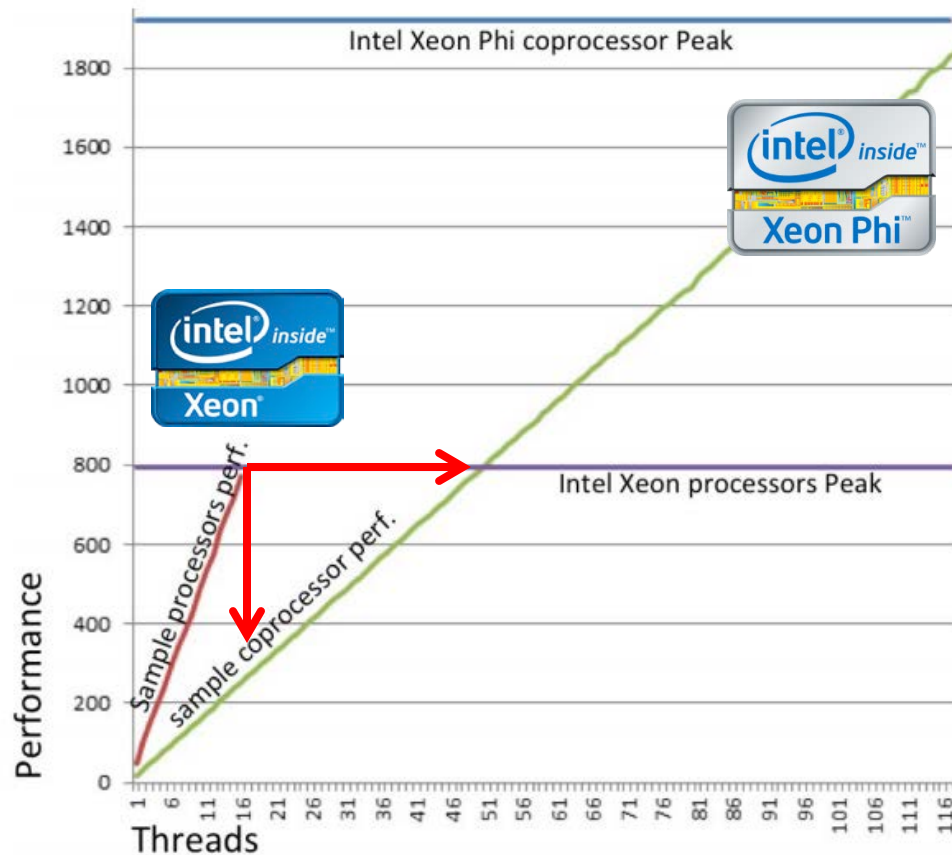
How many threads ?

“An application must scale well past one hundred threads to qualify as highly parallel”



Jim Jeffers
James Reinders.
ISBN: 978-0124104143

Parallel Performance Potential



If your performance needs are met by an Intel Xeon® processor, they will be achieved with fewer threads than on a coprocessor

On a coprocessor:

- Need more threads to achieve same performance
- Same thread count can yield less performance

Intel Xeon Phi excels on *highly parallel applications*



Intel® Xeon Phi™

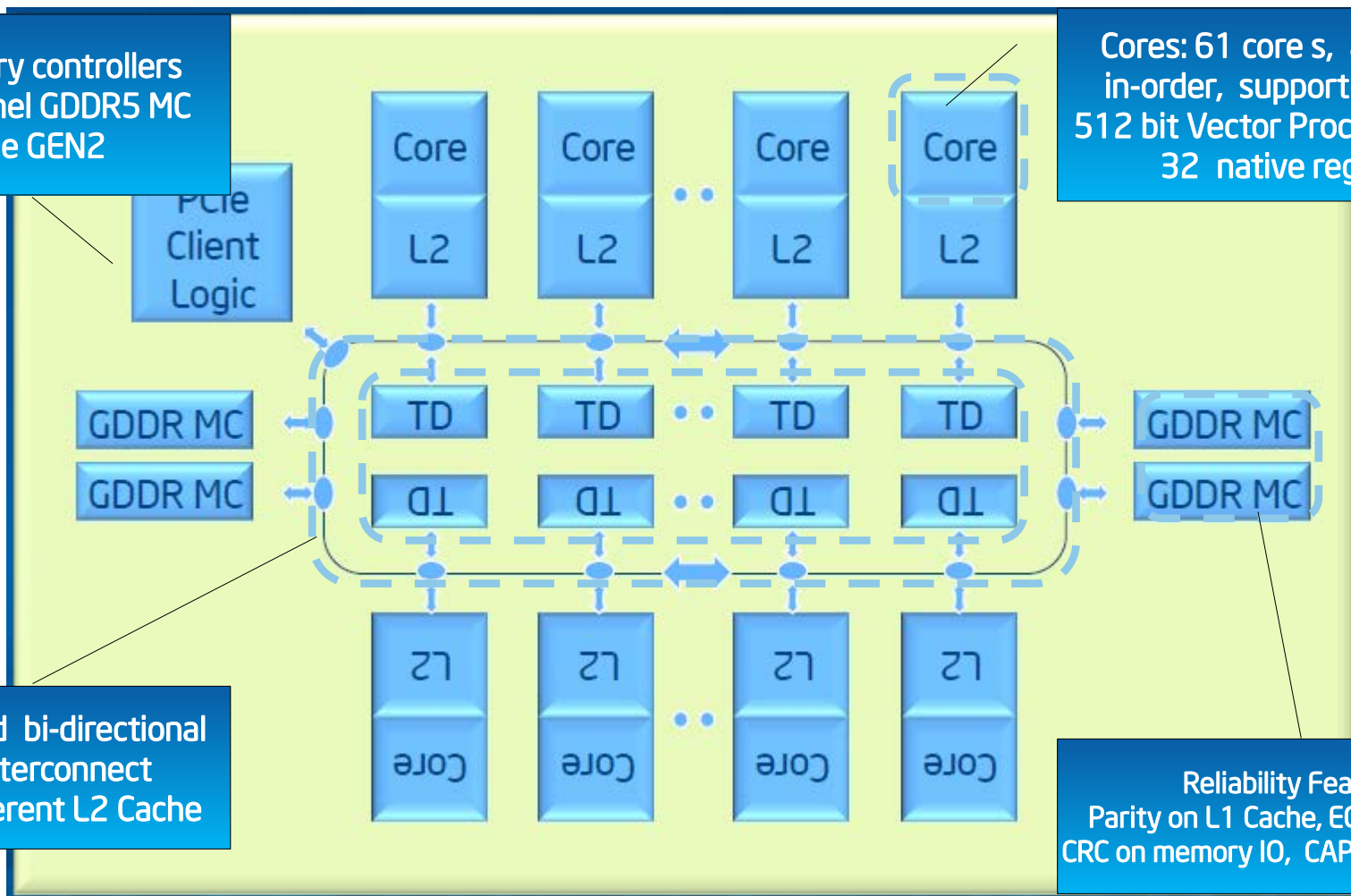
What is it?

- Co-processor
 - PCI Express card
 - Stripped down Linux operating system (busybox/dash)
- Dense, simplified processor
 - Simplifications for power savings **In-order**
 - Wider vector unit
 - Wider hardware thread count
- Lots of names
 - Many Integrated Core architecture, aka MIC
 - Knights Corner (code name)
 - Intel Xeon Phi Co-processor SE10P (product name)

Intel® Xeon Phi™ Architecture Overview

8 memory controllers
16 Channel GDDR5 MC
PCIe GEN2

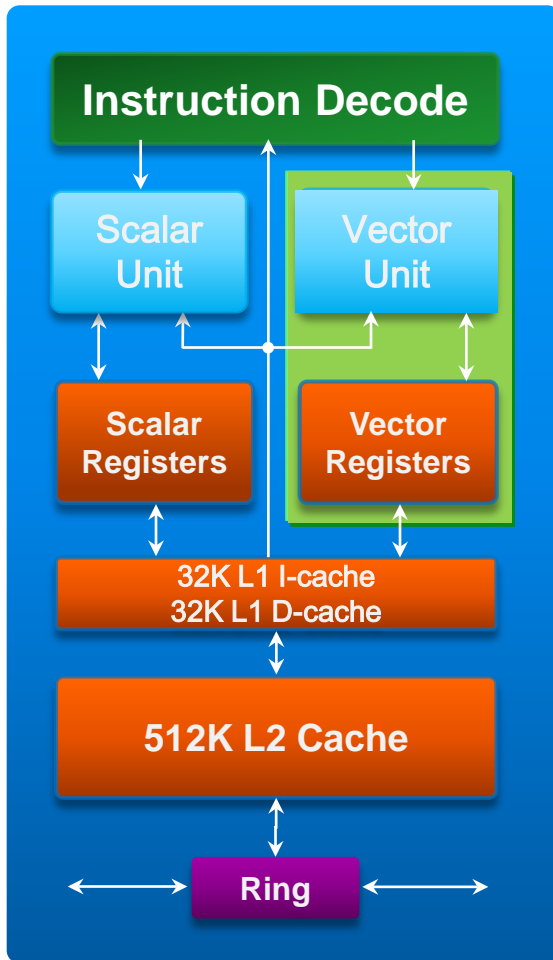
Cores: 61 core s, at 1.1 GHz
in-order, support 4 threads
512 bit Vector Processing Unit
32 native registers



Reliability Features
Parity on L1 Cache, ECC on memory
CRC on memory IO, CAP on memory IO

High-speed bi-directional
ring interconnect
Fully Coherent L2 Cache

Core Architecture Overview



60+ in-order, low power IA cores in a ring interconnect

Two pipelines

- Scalar Unit based on Pentium® processors
- Dual issue with scalar instructions
- Pipelined one-per-clock scalar throughput

SIMD Vector Processing Engine

4 hardware threads per core

- 4 clock latency, hidden by round-robin scheduling of threads
- Cannot issue back to back inst in same thread

Coherent 512KB L2 Cache per core

Key Differentiators

Xeon Phi vs Workstation

More **Cores**

Slower **Clock** Speed

Wider **SIMD** registers

Faster **Bandwidth**

In-order **pipeline**

A Tale of Two Architectures

	Intel® Xeon® processor	Intel® Xeon Phi™ Coprocessor
Sockets	2	1
Clock Speed	2.6 GHz	1.1 GHz
Execution Style	Out-of-order	In-order
Cores/socket	8	Up to 61
HW Threads/Core	2	4
Thread switching	HyperThreading	Round Robin
SIMD widths	8SP, 4DP	16SP, 8DP
Peak Gflops	692SP, 346DP	2020SP, 1010DP
Memory Bandwidth	102GB/s	320GB/s
L1 DCache/Core	32kB	32kB
L2 Cache/Core	256kB	512kB
L3 Cache/Socket	30MB	none

Theoretical Peak Flops Performance Example

Frequency * Num Sockets * Num Cores * Vector Width * FP Ops

Two socket Intel® Xeon® E5-2670 Processor

Freq	Sockets	Num Cores	Vector Width	FP Ops	GFlops
2.6	2	8	4	2	666

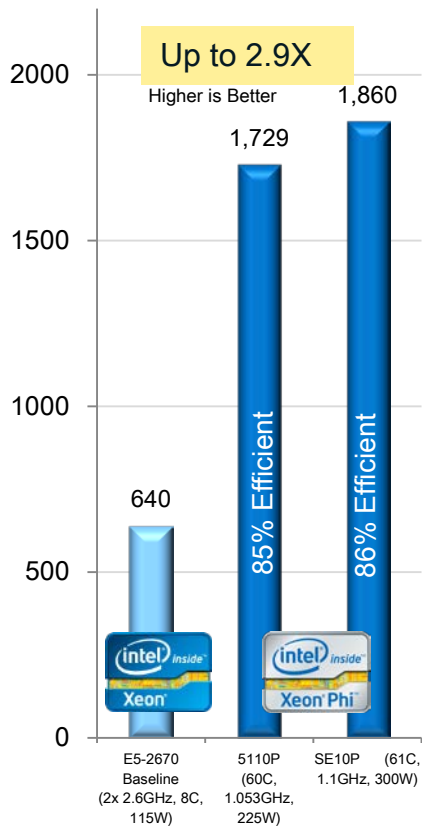
Single card Xeon Phi Coprocessor (B0)

Freq	Sockets	Num Cores	Vector Width	FP Ops	GFlops
1.091	1	61	16	2 (using FMA)	2,128

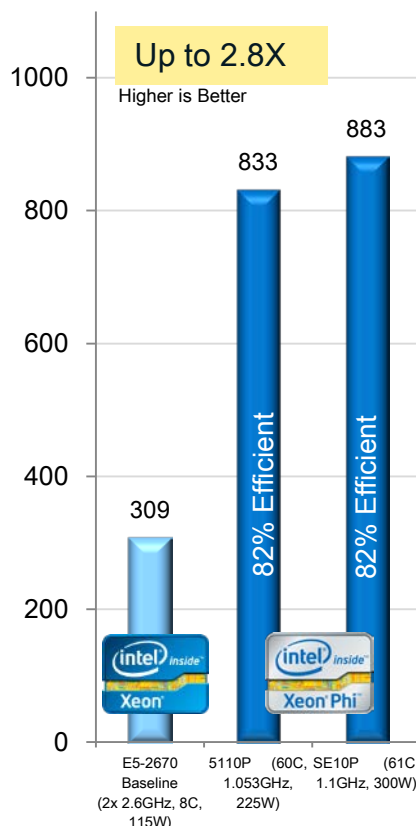


Synthetic Benchmark Summary (Intel® MKL)

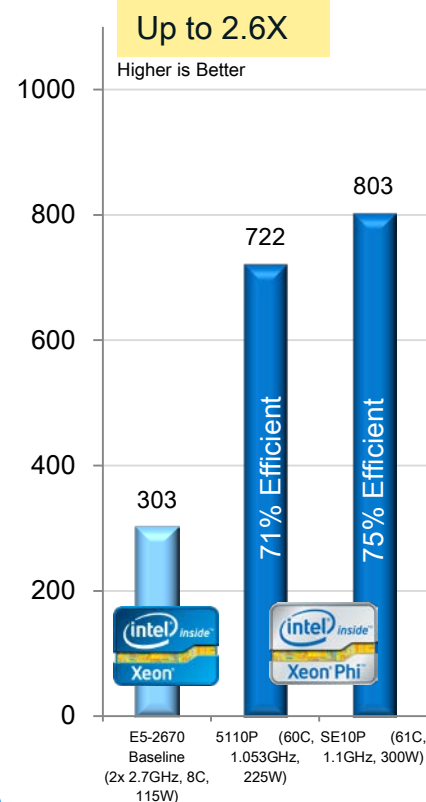
SGEMM (GF/s)



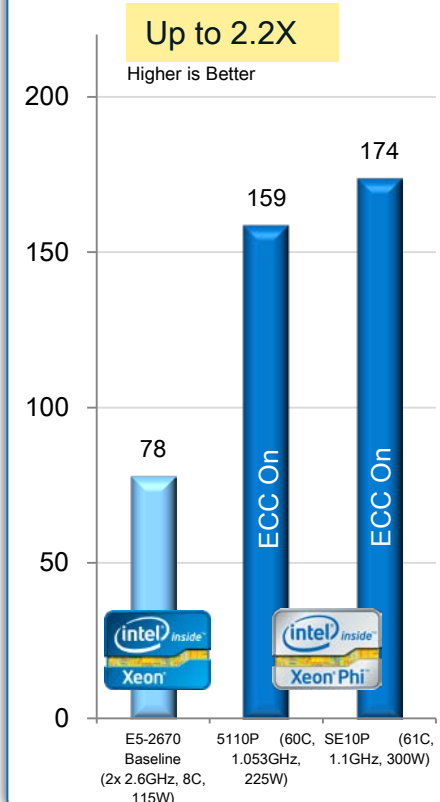
DGEMM (GF/s)



SMP Linpack (GF/s)



STREAM Triad (GB/s)



Coprocessor results: Benchmark run 100% on coprocessor, no help from Intel® Xeon® processor host (aka native)

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.

Source: Intel Measured results as of October 26, 2012. Configuration Details: Please reference slide speaker notes.

For more information go to <http://www.intel.com/performance>

Intel® Xeon Phi™ Coprocessor:

Increases Application Performance up to 10x

Updated

Segment	Customer	Application	Performance Increase ¹ vs. 2S Xeon*
Energy	Acceleware	8 th order isotropic variable velocity	Up to 2.23x
	Sinopec	Seismic Imaging	Up to 2.53x ²
	CNPC (China Oil & Gas)	GeoEast Pre-Stack Time Migration (Seismic)	Up to 3.54x ²
Financial Services	Financial Services	BlackScholes SP Monte Carlo SP	Up to 7.5x Up to 10.75x
Physics	Jefferson Labs	Lattice QCD	Up to 2.79x
Finite Element	Sandia Labs	miniFE (Finite Element Solver)	Up to 2x ³ Up to 1.3x ⁵
Solid State Physics	ZIB (Zuse-Institut Berlin)	Ising 3D (Solid State Physics)	Up to 3.46x
Digital Content Creation/Video	Intel Labs	Ray Tracing (incoherent rays)	Up to 1.88x ⁴
	NEC	Video Transcoding	Up to 3.0x ²
Astronomy	CSIRO/ASKAP (Australia Astronomy)	tHogbom Clean (Astronomy image smear removal)	Up to 2.27x
	TUM (Technische Universität München)	SG++ (Astronomy Adaptive Sparse Grids/Data Mining)	Up to 1.7x
Fluid Dynamics	AWE (Atomic Weapons Establishment - UK)	Cloverleaf (2D Structured Hydrodynamics)	1.77x

Notes:

1. 2S Xeon* vs. 1 Xeon Phi* (preproduction HW/SW & Application running 100% on coprocessor unless otherwise noted)
2. 2S Xeon* vs. 2S Xeon* + 2 Xeon Phi* (offload)
3. 8 node cluster, each node with 2S Xeon* (comparison is cluster performance with and without 1 Xeon Phi* per node) (Hetero)
4. Intel Measured Oct. 2012
5. 8 node cluster, each node with 2S Xeon* (comparison is cluster performance with Xeon only vs. Xeon Phi* only (1 Xeon Phi* per node) (Native)

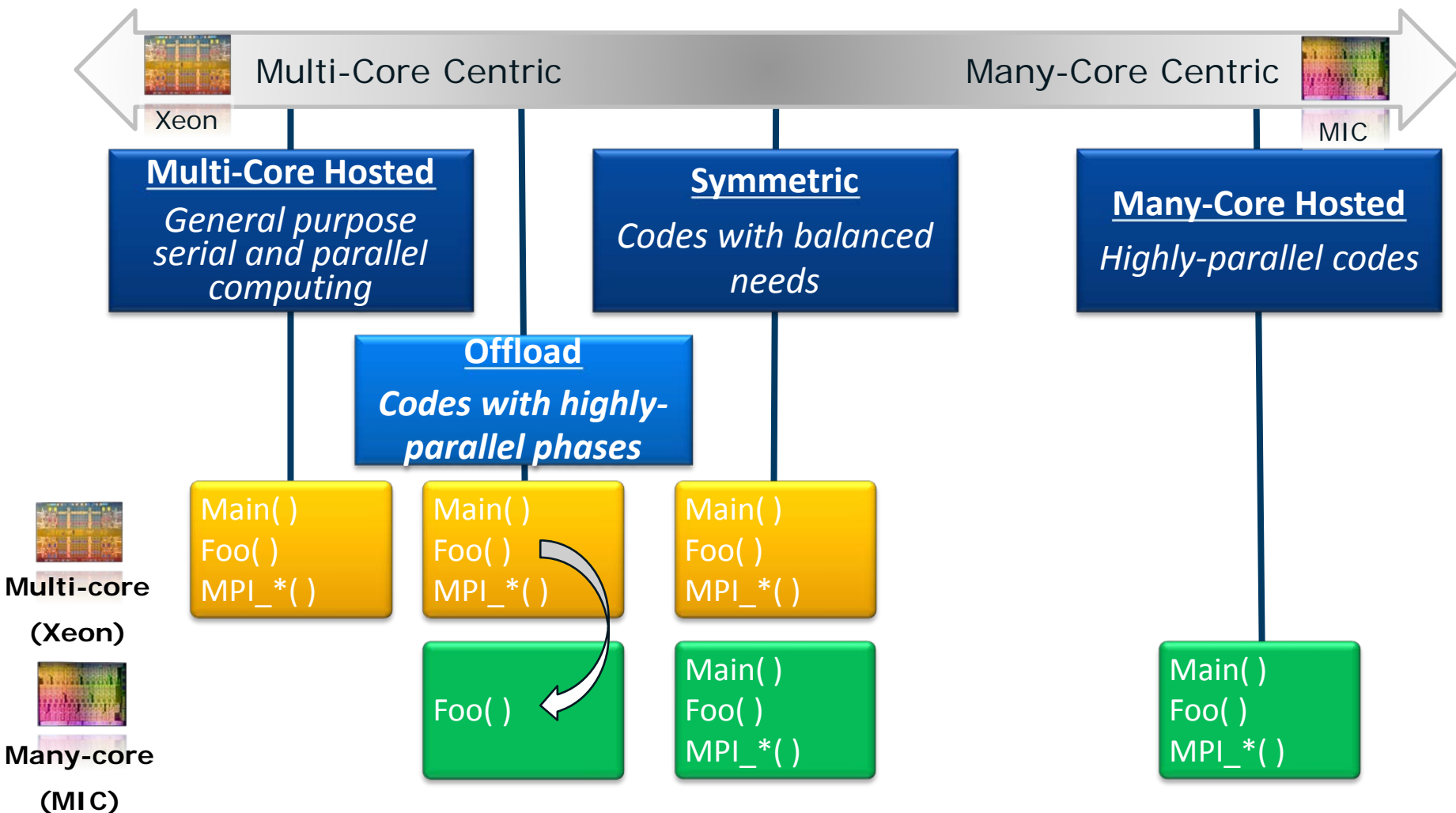
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may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

Source: Customer Measured results as of October 22, 2012. Configuration Details: Please reference slide speaker notes.

For more information go to <http://www.intel.com/performance>



Programming Models and Mindsets



Examples of Offloading



C/C++ Offload Pragma

```
#pragma offload target (mic)
#pragma omp parallel for reduction(+:pi)
for (i=0; i<count; i++) {
    float t = (float)((i+0.5)/count);
    pi += 4.0/(1.0+t*t);
}
pi /= count;
```

MKL Implicit Offload

//MKL implicit offload requires no source code changes,
simply link with the offload MKL Library.

MKL Explicit Offload

```
#pragma offload target (mic) \
    in(transa, transb, N, alpha, beta) \
    in(A:length(matrix_elements)) \
    in(B:length(matrix_elements)) \
    in(C:length(matrix_elements)) \

out(C:length(matrix_elements)alloc_if(0))
sgemm(&transa, &transb, &N, &N, &N, &alpha,
      A, &N, B, &N, &beta, C, &N);
```

Fortran Offload Directive

```
!dir$ omp offload target(mic)
!$omp parallel do
    do i=1,10
        A(i) = B(i) * C(i)
    enddo
!$omp end parallel
```

C/C++ Language Extensions

```
class _Shared common {
    int data1;
    char *data2;
    class common *next;
    void process();
};

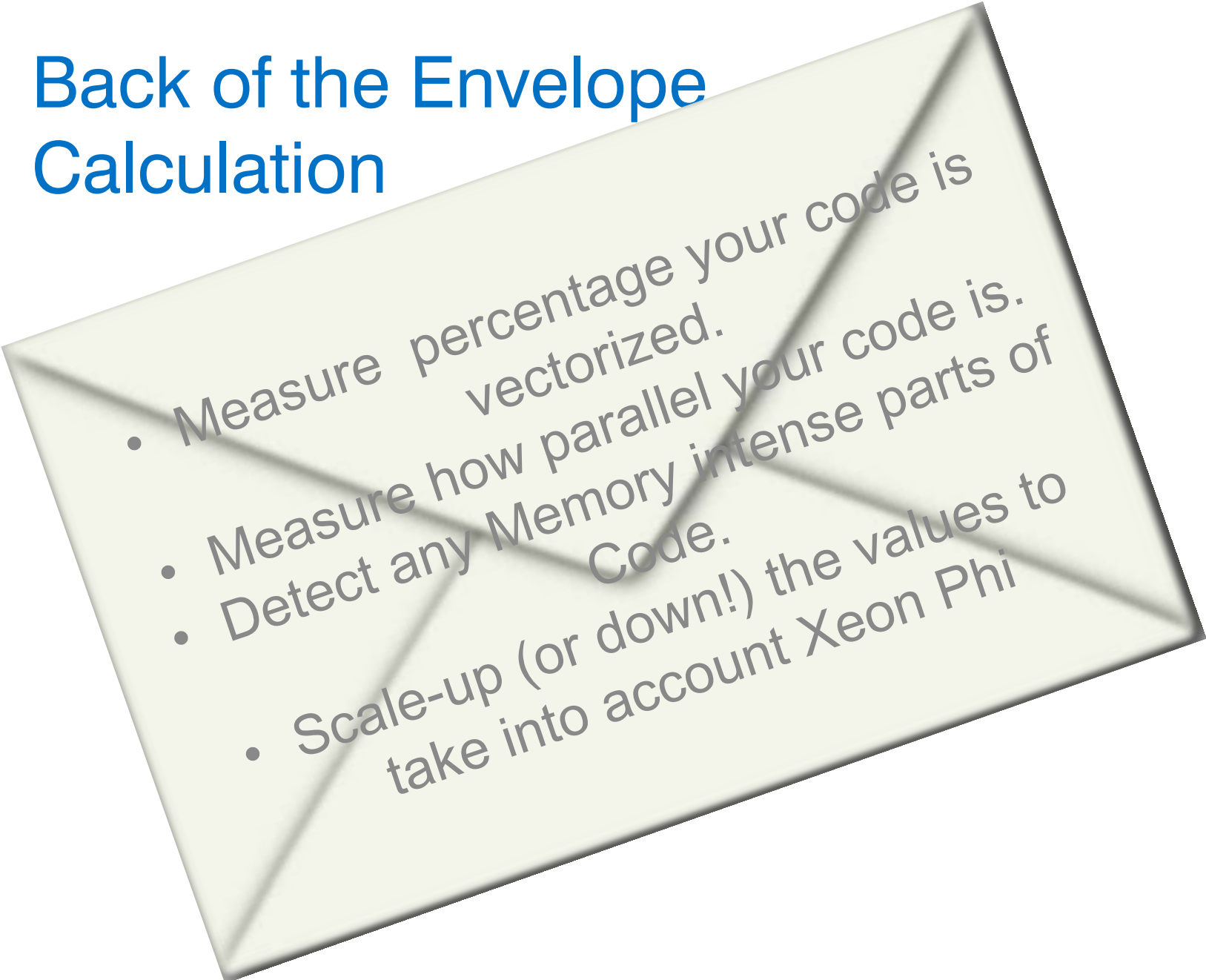
_Shared class common obj1, obj2;

...

_Cilk_spawn _Offload obj1.process();
_Cilk_spawn          obj2.process();

...
```

Back of the Envelope Calculation

- 
- Measure percentage your code is vectorized.
 - Measure how parallel your code is.
 - Detect any Memory intense parts of Code.
 - Scale-up (or down!) the values to take into account Xeon Phi

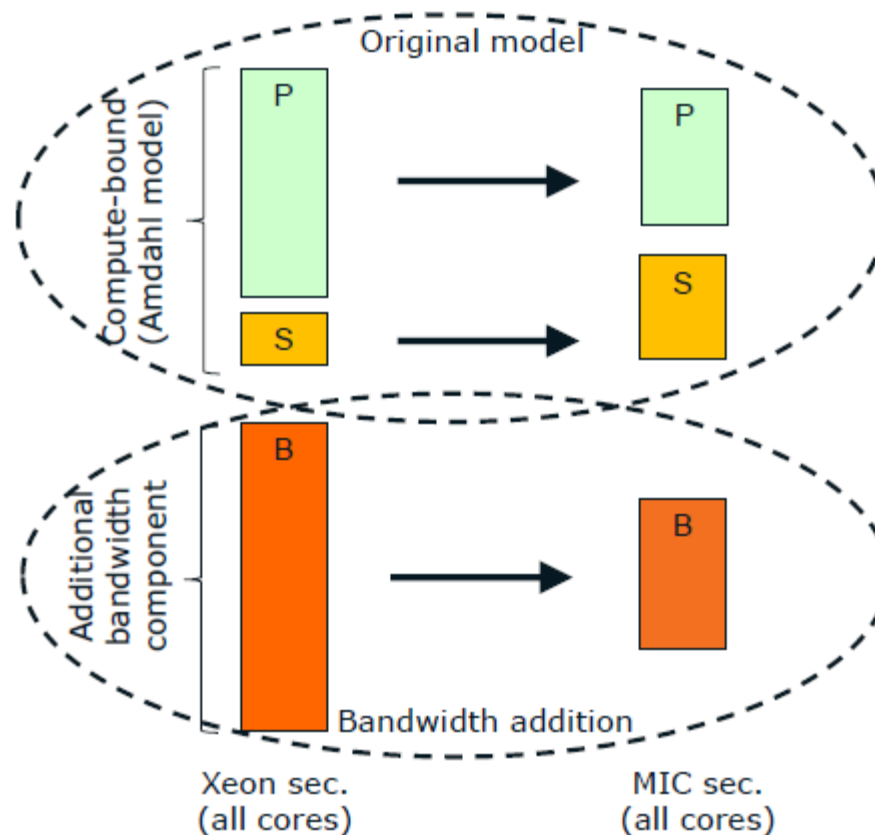
Your code will benefit from running on Xeon Phi if ...

- It is highly parallel
- Is effectively vectorised

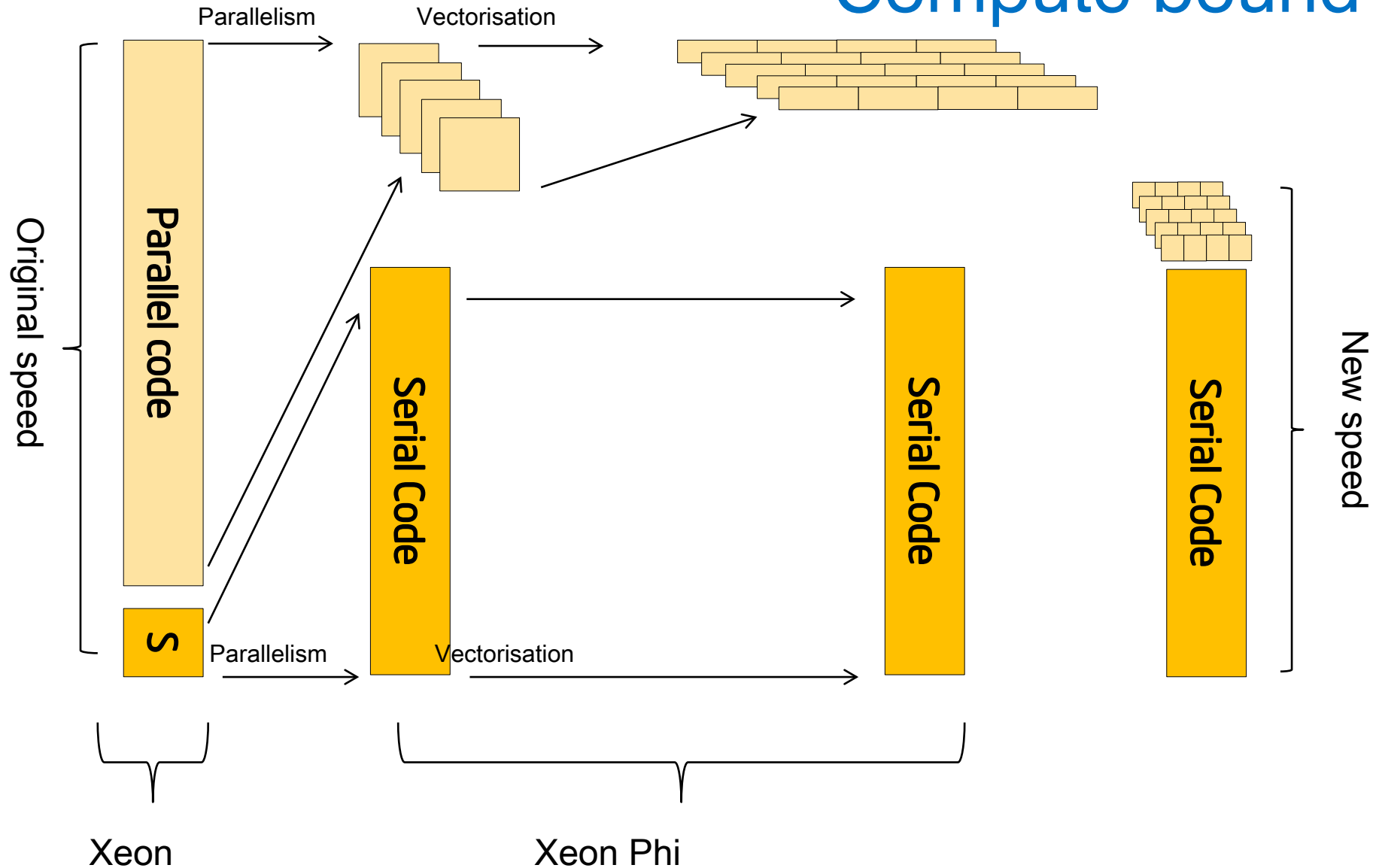
or **bandwidth**
constrained

Three things to consider

Three components to consider
P – the parallel part of the program
S – the serial part of the program
B – the bandwidth constrained part of the program



Compute bound



The Parallel, Vector and Clock Factors

Parallel Factor =

Num Xeon Cores / Num Phi Cores

$$16 / 61 = 0.26229$$

Vector Factor =

(Xeon Vector Length * Xeon Instruction Level Parallelism) /
(Phi Vector Length * Phi Instruction Level Parallelism)

$$AVX-FMA^{**} \quad 4 * 2 / 8 * 2 = .5$$

$$AVX-non-FMA \quad 4 * 2 / 8 * 1 = 1$$

$$SSE-FMA^{**} \quad 2 * 2 / 8 * 2 = .25$$

$$SSE-non-FMA \quad 2 * 2 / 8 * 1 = .5$$

Clock Factor =

Xeon Frequency / Phi Frequency

$$3.1 / 1.09 = 2.844$$

Combined = Parallel Factor * Vector Factor * Clock factor

$$AVX-FMA^{**} \quad 0.26229 * .5 * 2.844 = 0.373$$

$$AVX-non-FMA \quad 0.26229 * 1 * 2.844 = 0.746$$

$$SSE-FMA^{**} \quad 0.26229 * .25 * 2.844 = 0.187$$

$$SSE-non-FMA \quad 0.26229 * .5 * 2.844 = 0.373$$

*NB we are comparing 2 socket SNB with
single coprocessor (64 bit floating point doubles)*

** FMA:source code is capable of using FMA when built for Xeon Phi

FMA**
x5.38
Faster
(SSE2)

FMA**
x2.68
Faster
(AVX)

Non-FMA
X2.68
Faster
(SSE2)

Non-FMA
x1.34
Faster
(AVX)

The Serial Factor

Serial Factor =

Clock Factor * ILP Factor * Issue Factor

Where

Clock Factor = $2.6 / 1.09$

For FMA type calculations

ILP Factor^{***} = $2/2 = 1$

For non-FMA type calculations

ILP Factor = $2/1$

Issue factor =

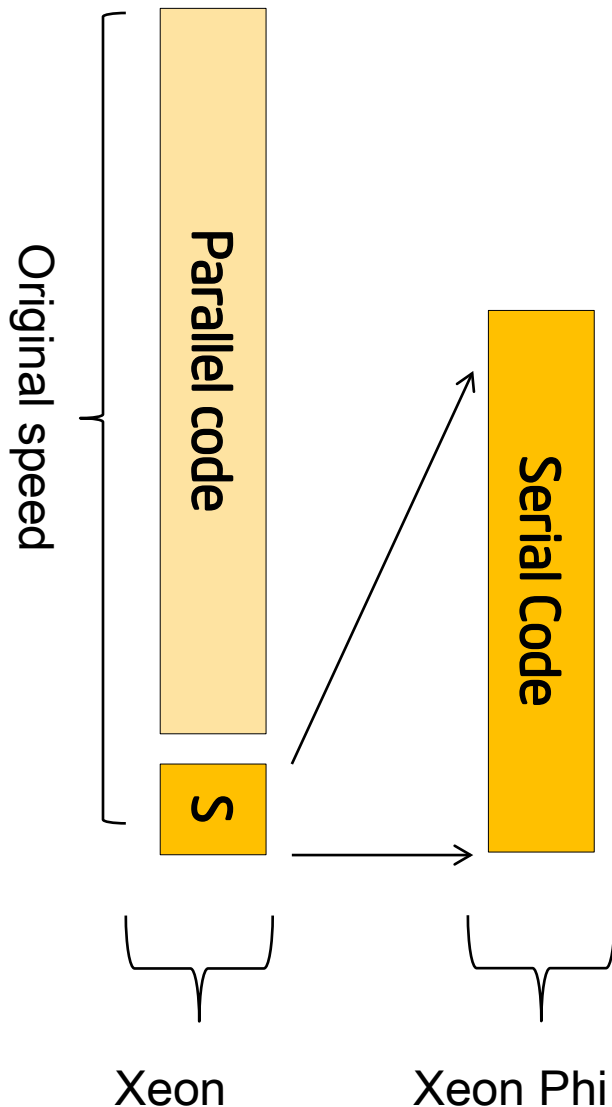
Num cycles to issue instruction on Phi /
Num cycles to issue instruction on Xeon
= $2/1$

Note: in single threaded code Xeon Phi uses
two cycles to issue an instruction
(in threaded mode it takes just one cycle)

^{***} FMA: source code is capable of using Fused Multiple Add
when built for Xeon Phi

FMA
x4.77
slower**

**Non-FMA
x9.54
slower**



Factors (2.6 GHz Clock)

Host	SIMD	Serial	Vector	Parallel	Clock
Single socket 2.6 GHz. FMA**	AVX	4.772	0.5	0.1333	2.386
	SSE2		0.25		
Single socket 2.6 GHz No FMA	AVX	9.544	1		
	SSE2		0.5		
Twin socket 2.6 GHz FMA**	AVX	4.772	0.5	0.2666	
	SSE2		0.25		
Twin socket 2.6 GHz No FMA	AVX	9.544	1		
	SSE2		0.5		

Xeon: 8 cores per socket

Phi: Using 60 of 61 cores

** FMA: source code is capable of using FMA when built for Xeon Phi

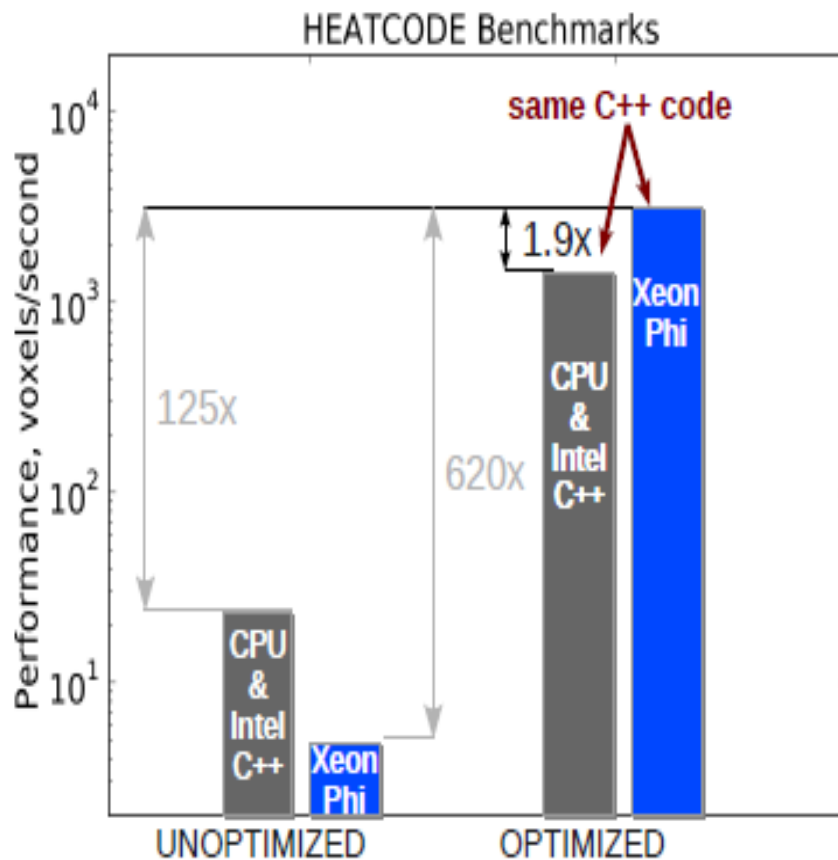
NOTE: Serial Factor already includes the Clock factor

'Finger in the air' speedups (from 2 socket 2.6Ghz SSE2)

- An application that is highly parallel and effectively vectorised - speed up **x2.5**
- An application that is highly parallel but not vectorised - speed up **x1.3**
- An application that is not parallel but is vectorised - slow down by **x1.5**
- A Serial application - slow down by **x12.0**
- A Bandwidth constrained application - speed up by **x2.4**

What you experience in practice may be different from these figures.
These are only 'back of the envelope' figures.

*Xeon Phi
optimisation
work usually is
of benefit to
'regular' Xeon
CPU codes.*



Dual-socket Intel Xeon E5-2670 CPU (16 cores total)

versus

Intel Xeon Phi 5110P coprocessor (60 cores)

See configuration information at end of
slide deck

KNL Public Knowledge

- Knights Landing is the code name for the **2nd generation** product in the Intel® Many Integrated Core Architecture
- Knights Landing targets Intel's **14 nanometer** manufacturing process
- Knights Landing will be productized as a **processor** (running the host OS) and a **coprocessor** (a PCIe end-point device)
- Knights Landing will feature on-package, **high-bandwidth memory**
- **Flexible memory modes** for the on package memory include: flat, cache, and hybrid modes
- Intel® Advanced Vector Extensions **AVX-512**

Typical Hands-on Xeon Phi training agenda

Day 1 – Getting Ready

- 10.00 Welcome
- 10.30 Two Essential Requirements
- 11.00 Parallelism (L)
- 12.30 Lunch
- 1.30 Vectorisation (L)
- 4.00 Advance Profiling (Walkthrough)
- 5.00 End



25th & 26th
June 2014
Manchester

Day 2-Xeon Phi Programming

- 09.00 Start
- 09.15 Native & Offload Programming for Xeon Phi (L)
- 11.30 A Case Study
- 12.00 Lunch
- 1.00 Vectorisation on Xeon Phi (L)
- 1.50 Parallelism on Xeon Phi (L)
- 3.40 Wrap-up
- 4.00 End

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 © , Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, VTune, and Cilk 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



Backup



