

# The significance of SIMD, SSE and AVX

**Stephen Blair-Chappell**  
Intel Compiler Labs

Software and Services Group

Optimization Notice 



# Agenda

- 1. Auto-Vectorisation
- 2. CPU Dispatch
- 3. Manual Processor Dispatch
- 4. A Case Study

**“I must have the Intel compiler, it has sped up our application by two.”**

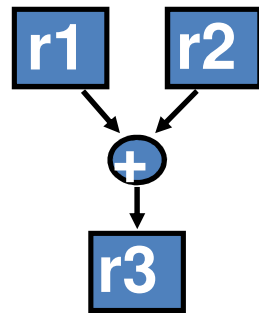
*A customer when moving from version 9.1 to version 10 of the Intel compiler*

# Auto-Vectorisation

# Vector Processing

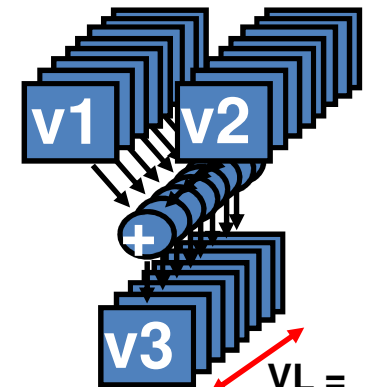
- A specific case of **data level parallelism** (DLP)
- Same operation simultaneously executed on  $N > 1$  elements of a vector.

Scalar Processing



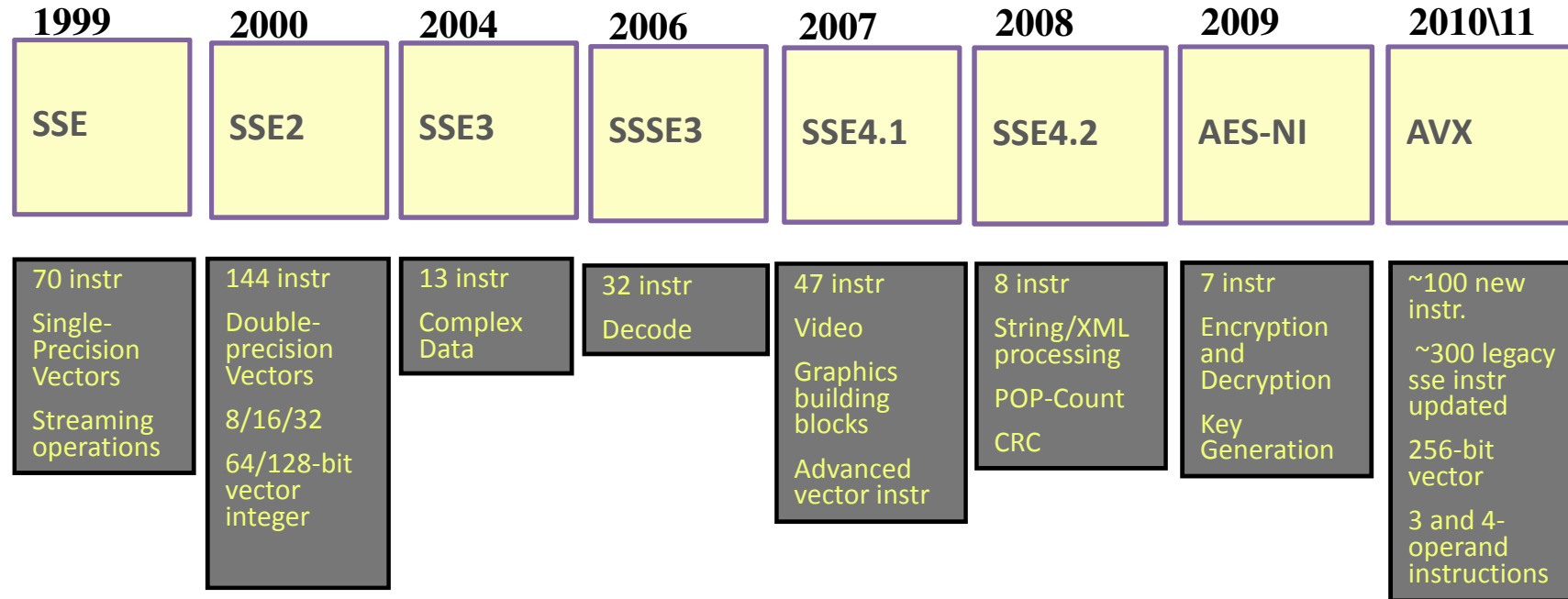
`add.d r3, r1, r2`

Vector Processing

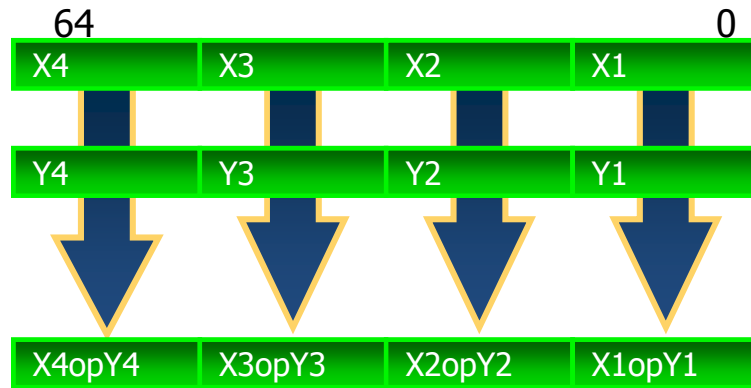


`addvec.d v3, v1, v2` VL = vector length

# SIMD: Continuous Evolution



# SIMD Types in Processors from Intel [1]



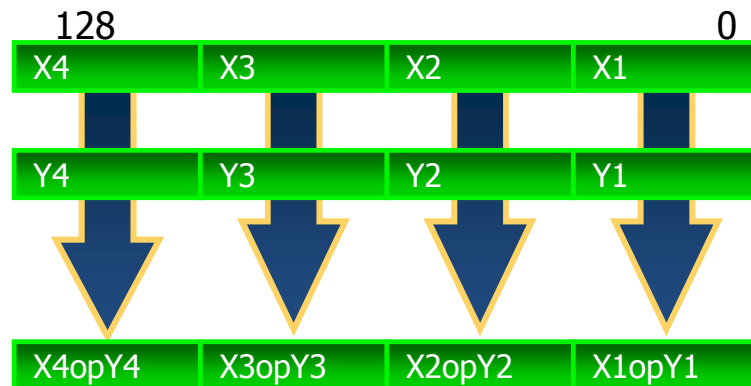
## MMX™

Vector size: 64bit

Data types: 8, 16 and 32 bit integers

VL: 2,4,8

For sample on the left: Xi, Yi 16 bit integers



## Intel® SSE

Vector size: 128bit

Data types:

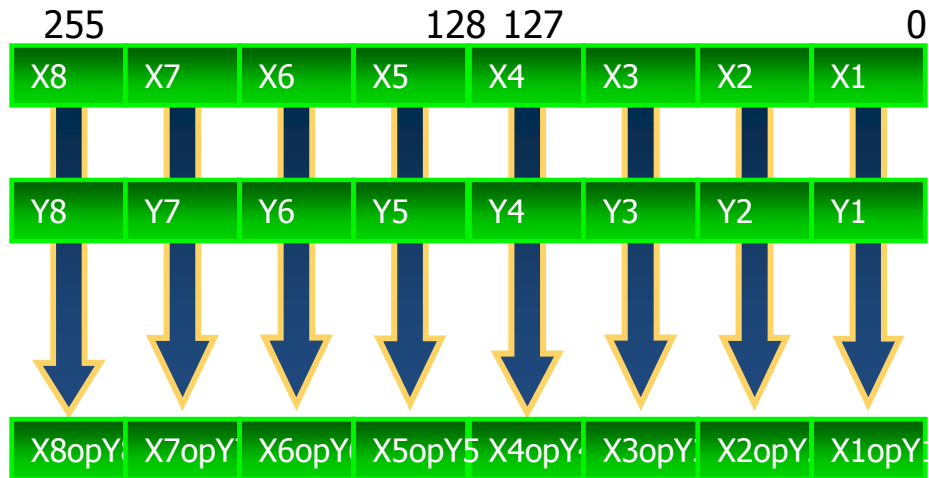
8,16,32,64 bit integers

32 and 64bit floats

VL: 2,4,8,16

Sample: Xi, Yi bit 32 int / float

# SIMD Types in Processors from Intel [2]



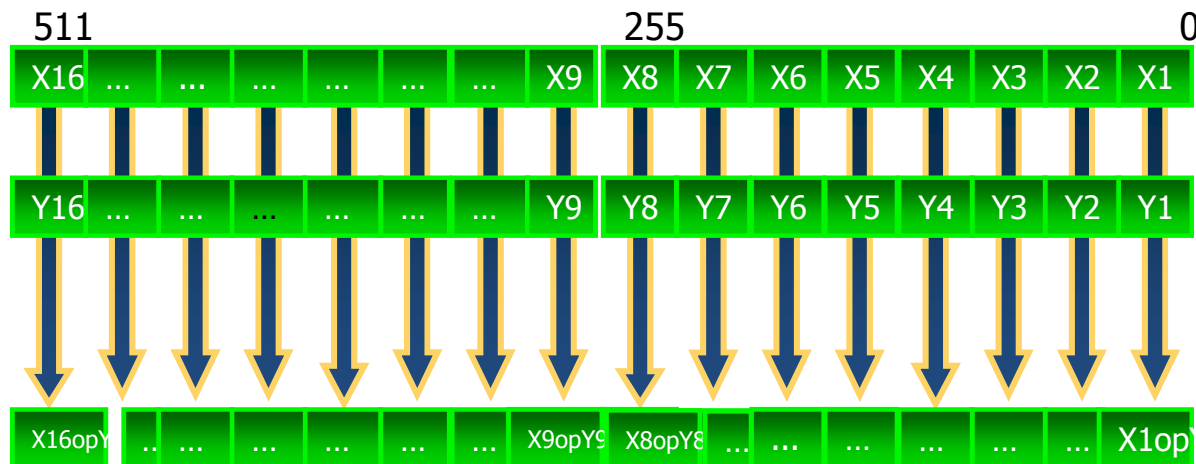
## Intel® AVX

Vector size: 256bit

Data types: 32 and 64 bit floats

VL: 4, 8, 16

Sample:  $X_i, Y_i$  32 bit int or float



## Intel® MIC

Vector size: 512bit

Data types:

32 and 64 bit integers

32 and 64bit floats

(some support for  
16 bits floats)

VL: 8,16

Sample: 32 bit float



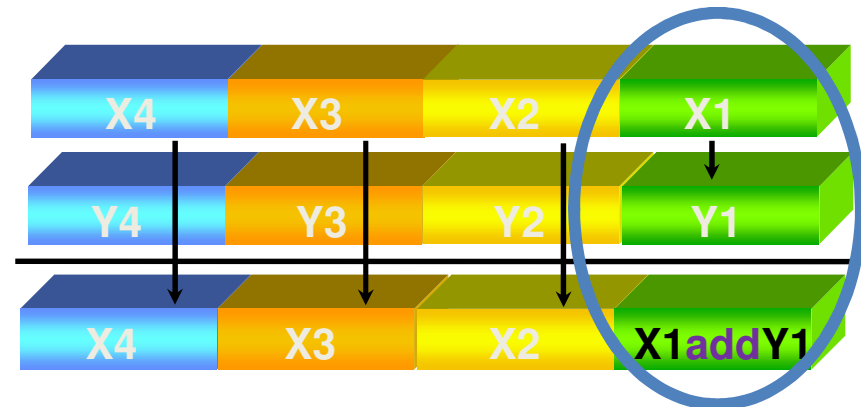
# Scalar and Packed SSE Instructions

The “vector” form of SSE instructions operating on multiple data elements simultaneously are called packed - thus vectorized SSE code means use of packed instructions

- Most of these instructions have a scalar version too operating only one element only

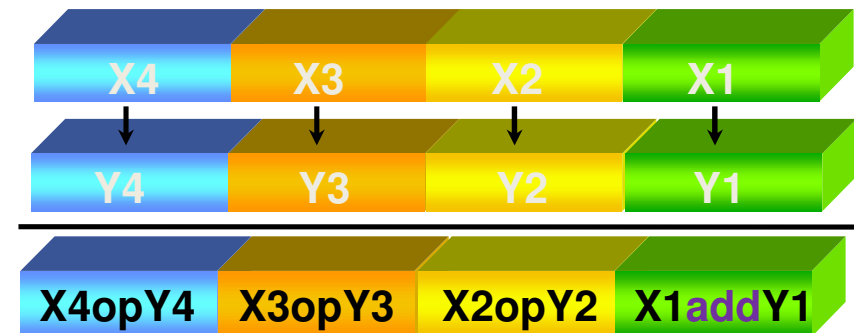
`addss` Scalar Single-FP Add

▲ `↑↑` single precision FP data  
scalar execution mode

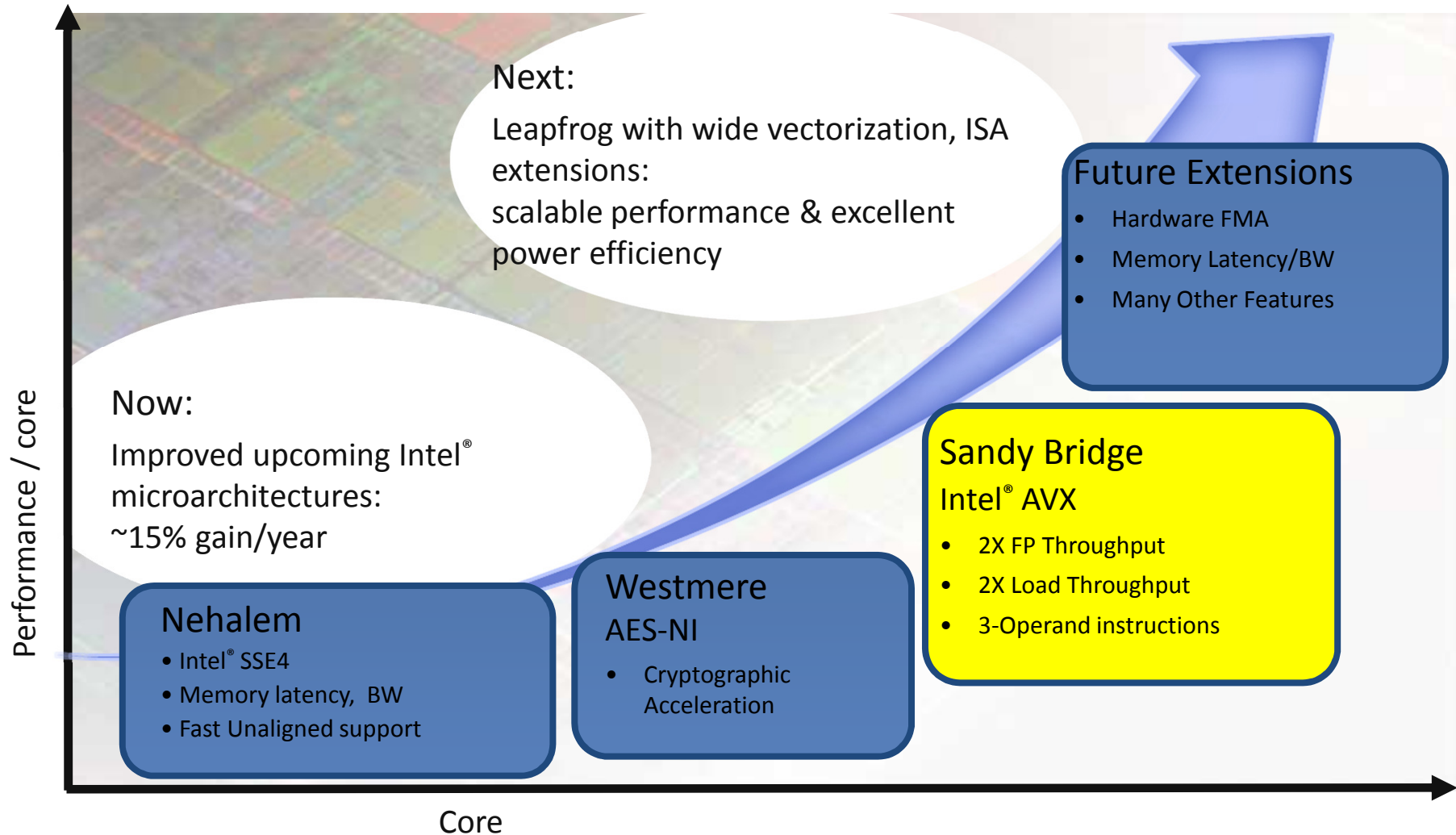


`addps` Packed Single-FP Add

`↑↑` single precision FP data  
packed execution mode



# Intel® AVX - Setting the Pace for Intel® Instruction Set



# Key Intel® Advanced Vector Extensions (Intel® AVX) Features

## KEY FEATURES

- Wider Vectors
  - Increased from 128 to 256 bit
  - Two 128-bit load ports
- Enhanced Data Rearrangement
  - Use the new 256 bit primitives to broadcast, mask loads and permute data
- Three and four Operands: Non Destructive Syntax for both AVX 128 and AVX 256
- Flexible unaligned memory access support
- Extensible new opcode (VEX)

## BENEFITS

- Up to 2x peak FLOPs (floating point operations per second) output with good power efficiency
- Organize, access and pull only necessary data more quickly and efficiently
- Fewer register copies, better register use for both vector and scalar code
- More opportunities to fuse load and compute operations
- Code size reduction

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



# A New 3- and 4- Operand Instruction Format

- Intel® Advanced Vector Extensions (Intel® AVX) has a distinct destination argument that results in fewer register copies, better register use, more load/op macro-fusion opportunities, and smaller code size

*xmm10 = xmm9 + xmm1*

```
movaps xmm10, xmm9  
addpd  xmm10, xmm1
```

▶ `vaddpd xmm10, xmm9, xmm1`

1 less copy,  
3 bytes smaller code size

*xmm10 = xmm9 + m128*

```
movups xmm10, m128  
addpd  xmm10, xmm9
```

▶ `vaddpd xmm10, xmm9, m128`

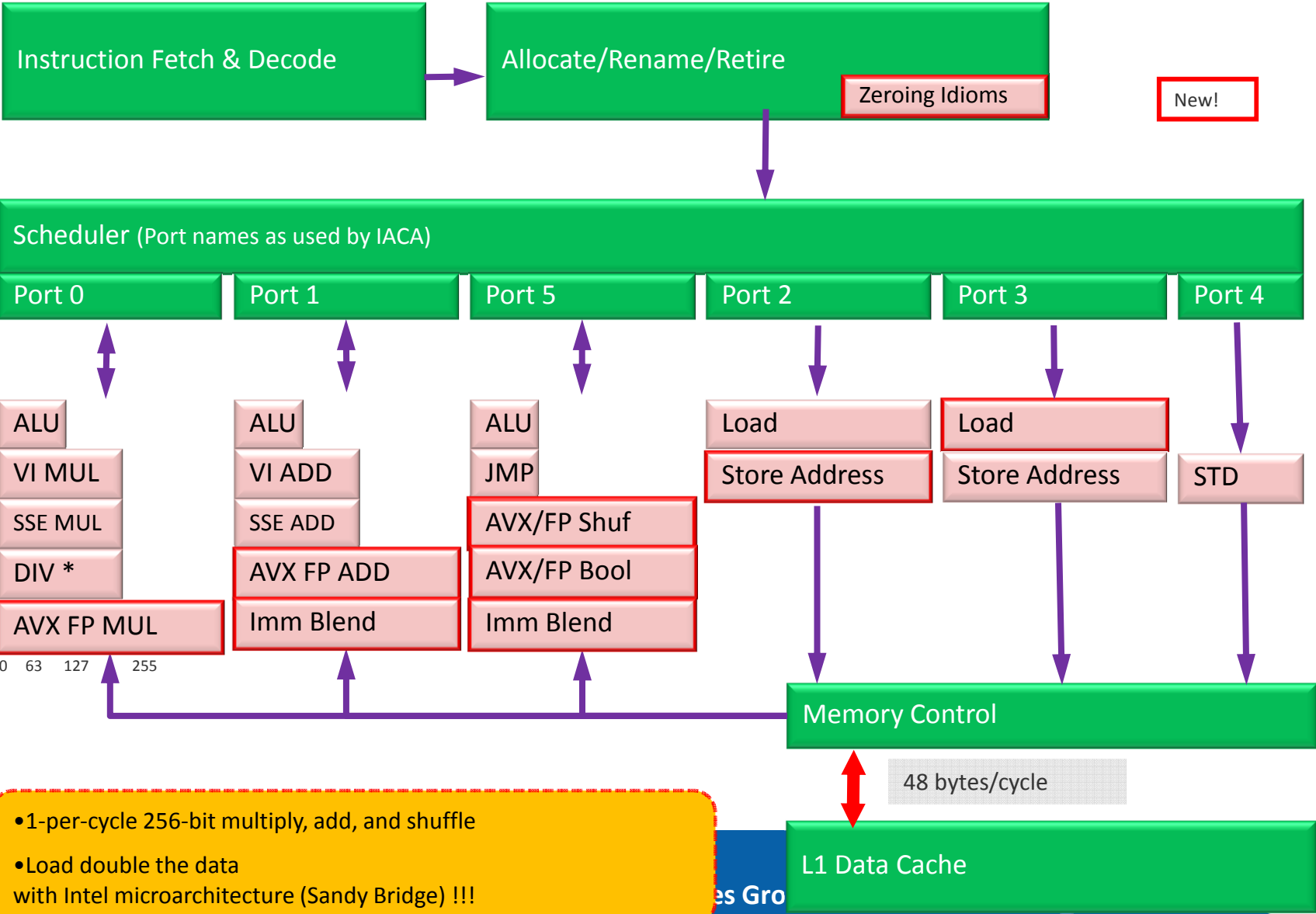
1 more load/op  
fusion opportunity,  
4+ bytes smaller  
code size

- New 4- operand Blends example, implicit xmm0 not longer needed

```
movaps xmm0, xmm4  
movaps xmm1, xmm2  
blendvps xmm1, m128
```

▶ `vblendvps xmm1, xmm2, m128, xmm4`

# Intel® Microarchitecture (Sandy Bridge) Highlights



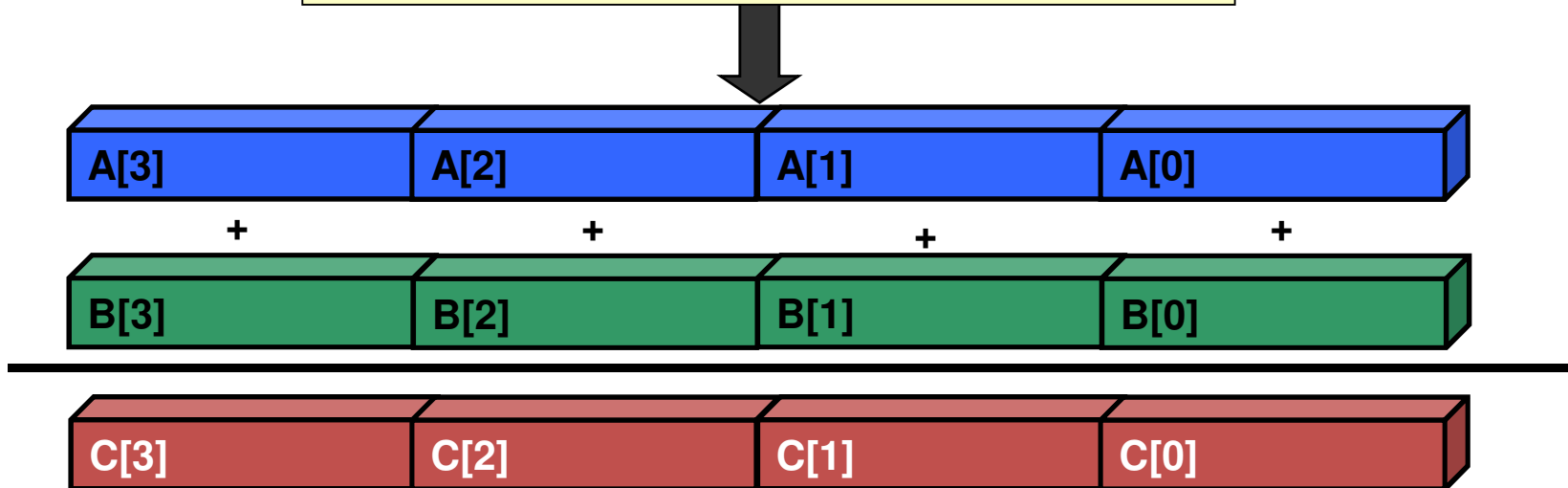
- 1-per-cycle 256-bit multiply, add, and shuffle
- Load double the data with Intel microarchitecture (Sandy Bridge) !!!

\* Not fully pipelined

# Auto-Vectorization

Transforming sequential code to exploit the vector (SIMD, SSE) processing capabilities

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



# Many Ways to introduce SSE Vectorization

Use Performance Libraries  
(e.g. IPP and MKL)

Compiler: Fully automatic vectorization

Cilk Plus Array Notation

Compiler: Auto vectorization hints (`#pragma ivdep, ...`)

User Mandated Vectorization  
( SIMD Directive)

Manual CPU Dispatch (`__declspec(cpu_dispatch ...)`)

SIMD intrinsic class (`F32vec4 add`)

Vector intrinsic (`mm_add_ps()`)

Assembler code (`addps`)

Ease of use



Programmer control

# 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.
```



# 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::  
    movaps    xmm2, A[rdx*8]  
    xorps     xmm0, xmm0  
    cmpltpd   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  
    j1        .B1.2
```

**SSE2**



```
.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  
    j1        .B1.2
```

**AVX**



```
.B1.2::  
    movaps    xmm2, A[rdx*8]  
    xorps     xmm0, xmm0  
    cmpltpd   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  
    j1        .B1.2
```

**SSE4.1**



## Vectorization Report

### “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”

# Elemental Functions

- Use **scalar syntax** to describe an operation on a **single element**
- Apply operation to arrays in parallel
- Utilize both **vector parallelism** and **core parallelism**

```
_declspec(vector)
double option_price_call_black_scholes
    (double S,double K,double r,double sigma,double time)
{
    double time_sqrt = sqrt(time);
    double d1 =
        (log(S/K)+r*time)/(sigma*time_sqrt)+0.5*sigma*time_sqrt;
    double d2 = d1-(sigma*time_sqrt);
    return S*N(d1) - K*exp(-r*time)*N(d2);
}

cilk_for (int i=0; i < NUM_OPTIONS; i++) {
    call_serial[i] = option_price_call_black_scholes(S[i], K[i], r, sigma, time[i]);
}
```

# CPU-Dispatch

## Adding Portability

**“I’ve stopped using the Intel compiler. Each time I ship the product to a customer, they complain that applications crashes”!**”

*A games developer at a recent networking event.*

# Imagine this scenario:

1. Your IT dept have just bought you the **latest and greatest** Intel based workstation.
2. You've heard **auto-vectorisation** can make a real difference to performance
3. You enable auto-vectorisation using **-xhost**
4. You boast to your colleagues, "*my application runs faster than anything you can write...*"
5. You send the application to a colleague - **it refuses to run.**

# What might be the issue?

# How can it be overcome?

# Two Key Decisions to be Made :

1. How do we **introduce** the vector code ?
2. How do we deal with the **multiple** SIMD instruction set **extensions** like SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, AVX ...?



# Out-of-the-box behaviour – Intel Compiler

- Automatic-vectorisation is **enabled** by default
- (turn it off with `-no-vec`)
  
- The option `-msse2` is used by default (as long as no `x`, `ax` or `-m` option has been used)

`-msse2`: “May generate Intel® SSE2 and SSE instructions. This value is only available on Linux systems”.

# Building for **non-intel** processors (-m)

Option	Description
sse4.1	May generate Intel® SSE4.1, SSSE3, SSE3, SSE2, and SSE instructions.
ssse3	May generate Intel® SSSE3, SSE3, SSE2, and SSE instructions.
sse2	May generate Intel® SSE2 and SSE instructions.
sse	This option has been deprecated; it is now the same as specifying ia32.
ia32	Generates x86/x87 generic code that is compatible with IA-32 architecture.

This option tells the compiler to generate code specialized for the processor that executes your program.

Code generated with these options should execute on any compatible, non-Intel processor with support for the corresponding instruction set.

# Building for Intel processors (-x)

Option	Description
AVX	AVX, SSE4.2, SSE4.1, SSSE3, SSE3, SSE2, and SSE instructions .
SSE4.2	SSE4 Efficient Accelerated String and Text Processing instructions supported by Intel® Core™ i7 processors. SSE4 .1, SSSE3, SSE3, SSE2, and SSE. May optimize for the Intel® Core™ processor family.
SSE4.1	SSE4 Vectorizing Compiler and Media Accelerator, SSSE3, SSE3, SSE2, and SSE . May optimize for Intel® 45nm Hi-k next generation Intel® Core™ microarchitecture.
SSE3_ATOM	MOVBE , (depending on -minstruction ), SSSE3, SSE3, SSE2, and SSE . Optimizes for the Intel® Atom™ processor and Intel® Centrino® Atom™ Processor Technology
SSSE3	SSSE3, SSE3, SSE2, and SSE. Optimizes for the Intel® Core™ microarchitecture.
SSE3	SSE3, SSE2, and SSE. Optimizes for the enhanced Pentium® M processor microarchitecture and Intel NetBurst® microarchitecture.
SSE2	SSE2 and SSE . Optimizes for the Intel NetBurst® microarchitecture.

# Auto-Vectorization –Running on Sandy Bridge

-xAVX

CPU ID

AVX

```
for(i=0;i<NUM;i++)  
{  
    j[i] = h[i] + i + 3  
}
```

*Running on a CPU supporting AVX*

# Auto-Vectorization

**-xAVX**

CPU ID

**AVX**

```
for(i=0;i<NUM;i++)  
{  
    j[i] = h[i] + i + 3  
}
```

*Fatal Error: This program was not built to run in your system.*

*Please verify that both the operating system and the processor support Intel(R) AVX.*

**Running on a CPU not supporting AVX**

## Using `-ax` compiler option ...

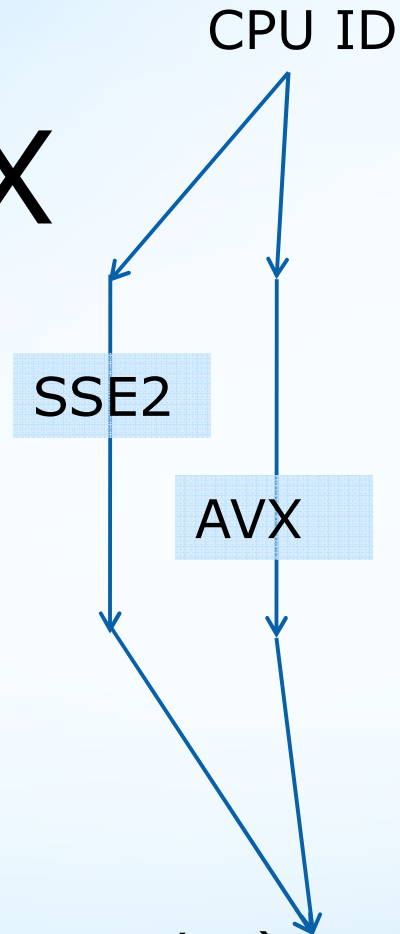
- Generates **multiple paths** if there is a performance benefit
- Generates a **base line** path
- **Other options** (e.g. `-O3`) control the base line path
- At **runtime** path chosen based on what processor code is running on

# The Base line

- Use -m or -x to set base line
- **-m** for non-intel processors
- **-X** for intel processors
- If no -m or -x, compiler defaults to **-mSSE2**
- -m and -x are **mutually exclusive**

# CPU Dispatching

**-axAVX**



Base line  
(set with -m or -x option)  
SSE2

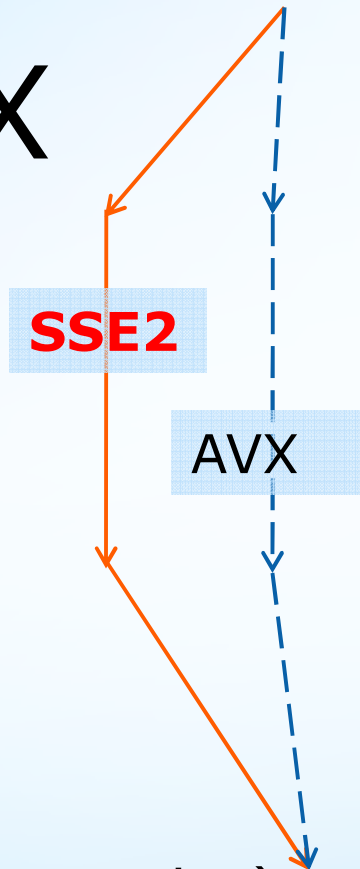
```
for(i=0;i<NUM;i++)  
{  
    j[i] = h[i] + i + 3  
}
```



# Generic low-spec CPU (no support of AVX)

-axAVX

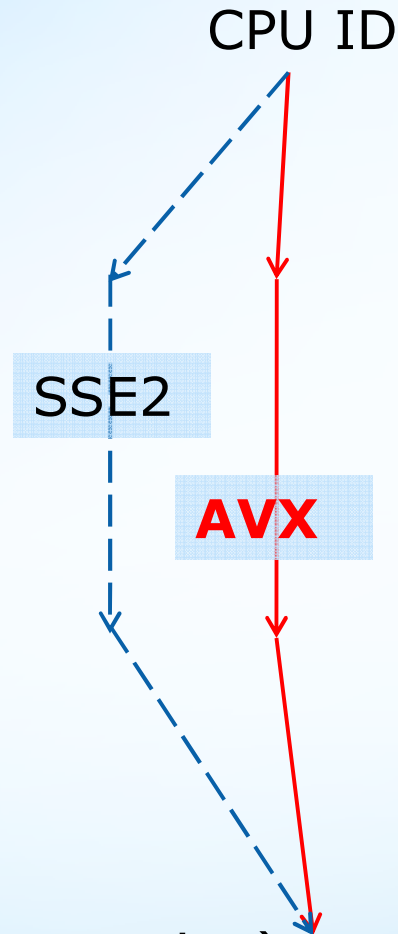
CPU ID



```
for(i=0;i<NUM;i++)  
{  
    j[i] = h[i] + i + 3  
}
```

Base line  
(set with -m or -x option)  
SSE2

# Sandy Bridge (supports AVX)



```
for(i=0;i<NUM;i++)  
{  
    j[i] = h[i] + i + 3  
}
```

Base line  
(set with -m or -x option)  
SSE2

# Running on Intel Processors

- If `-ax` and `-x` are used together
- Base line will execute on Intel compatible processors specified by the `-x`

# Running on Intel and non-Intel processors

- If `-ax` and `-m` are used together
- Base line will execute on non-Intel processors compatible with the processor type specified by `-m`

# What option do **AMD** recommend?

**AMD Opteron™ 6100 Series P**  
**AMD Opteron™ 4100 Series P**  
Compiler Options Quick Referen

## ICC

Latest release: 12.0 update3, March 2011

<http://software.intel.com>

Architecture	
Generate instructions specific to Magny-Cours	-msse3 (avoid -ax)
Optimization Levels	
Disable all optimizations	-O0

<http://developer.amd.com/Assets/CompilerOptQuickRef-61004100.pdf>

# Quiz – what option is best?

1. Your application will only ever run on the same CPU as your development machine
2. Your application will run on a farm of AMD Opterons (4100) and Intel i7s
3. Your application will run on Sandy Bridge Machines and Core 2.
4. You have no clue what machine the code will run on.

# Benefit of CPU Dispatch

## Code

- still works on **older** processors
- Works properly on **non-intel** CPUs
  - Non-intel processors will ALWAYS take the base-line
- Code can take advantage of **latest generation** of CPUs

# Manual Processor Dispatch



# Manual processor Dispatch

- Allows you to write processor-specific code
- Provide more than one version of code
- Use `__declspec(cpu_dispatch(cpuid,cpuid...))`

# CPUID Arguments

Argument for cpuid	Processors
<b>future_cpu_16 (subject to change)</b>	2nd generation Intel® Core™ processor family with support for Intel® Advanced Vector Extensions (Intel® AVX).
<b>core_aes_pclmulqdq</b>	Intel® Core™ processors with support for Advanced Encryption Standard (AES) instructions and carry-less multiplication instruction
<b>core_i7_sse4_2</b>	Intel® Core™ processor family with support for Intel® SSE4 Efficient Accelerated String and Text Processing instructions (SSE4.2)
<b>atom</b>	Intel® Atom™ processors
<b>core_2_duo_sse4_1</b>	Intel® 45nm Hi-k next generation Intel® Core™ microarchitecture processors with support for Intel® SSE4 Vectorizing Compiler and Media Accelerators instructions (SSE4.1)
<b>core_2_duo_ssse3</b>	Intel® Core™2 Duo processors and Intel® Xeon® processors with Intel® Supplemental Streaming SIMD Extensions 3 (SSSE3)
<b>pentium_4_sse3</b>	Intel® Pentium 4 processor with Intel® Streaming SIMD Extensions 3 (Intel® SSE3), Intel® Core™ Duo processors, Intel® Core™ Solo processors
<b>pentium_4</b>	Intel® Intel Pentium 4 processors
<b>pentium_m</b>	Intel® Pentium M processors
<b>pentium_iii</b>	Intel® Pentium III processors
<b>generic</b>	Other IA-32 or Intel 64 processors or compatible processors not provided by Intel Corporation

# Manual Dispatch Example

```
#include <stdio.h>
// need to create specific function versions
__declspec(cpu_dispatch(generic, future_cpu_16))
void dispatch_func() {};

__declspec(cpu_specific(generic))
void dispatch_func() {
    printf("Code for non-Intel processors\and generic Intel\n");
}

__declspec(cpu_specific(future_cpu_16))
void dispatch_func() {
    printf("Code for 2nd generation Intel Core processors goes here\n");
}
int main() {
    dispatch_func();
    printf("Return from dispatch_func\n");
    return 0;
}
```

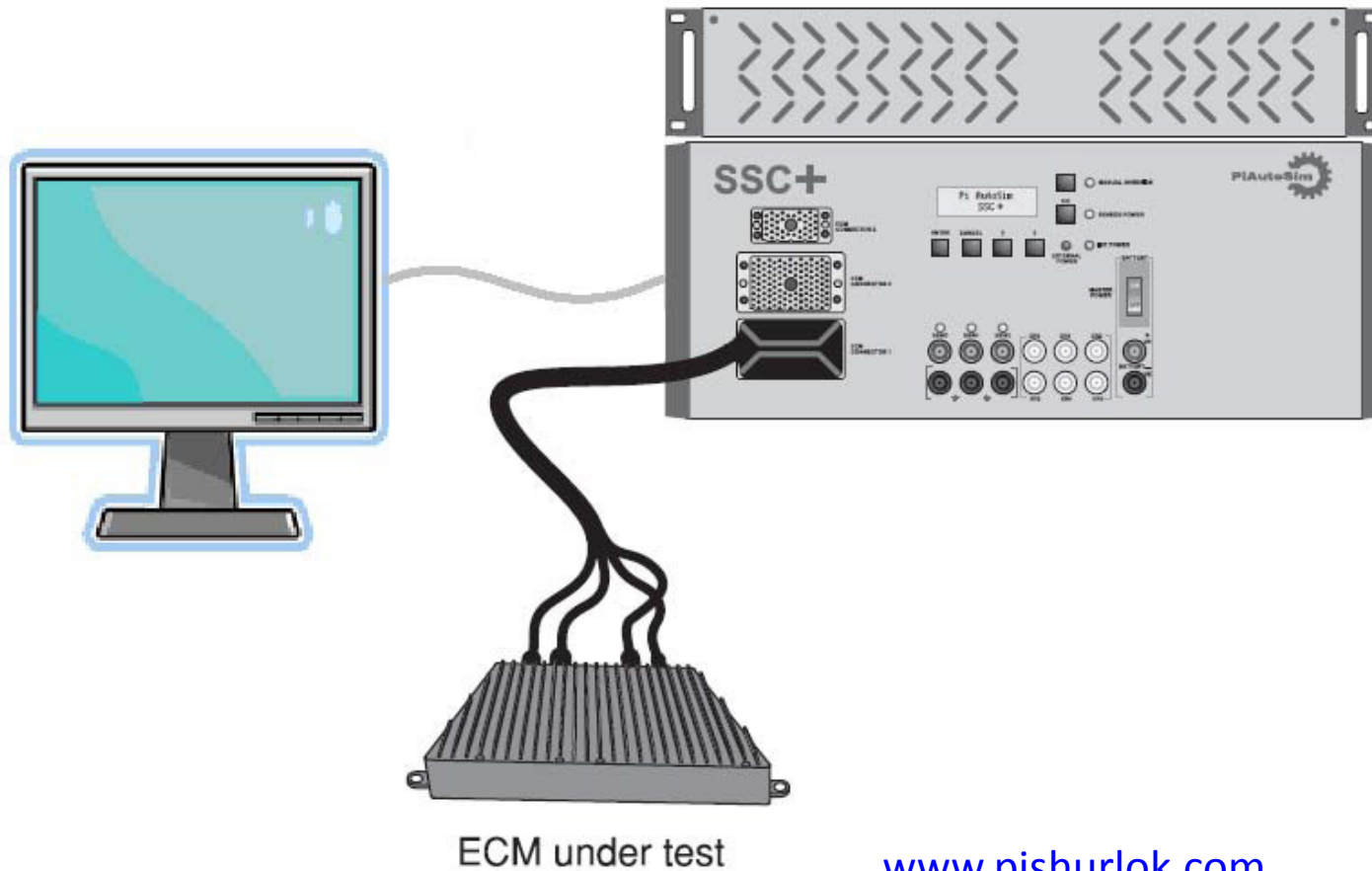
# Questions to Ask

- Is my **application** going to run on a different CPU to my **development** platform?
- Is my application going to run on one **specific generation** of CPU?
- Is my application just going to run on just **Intel** CPUs?
- Will my application be running on **non-intel** processors?

# A Case Study

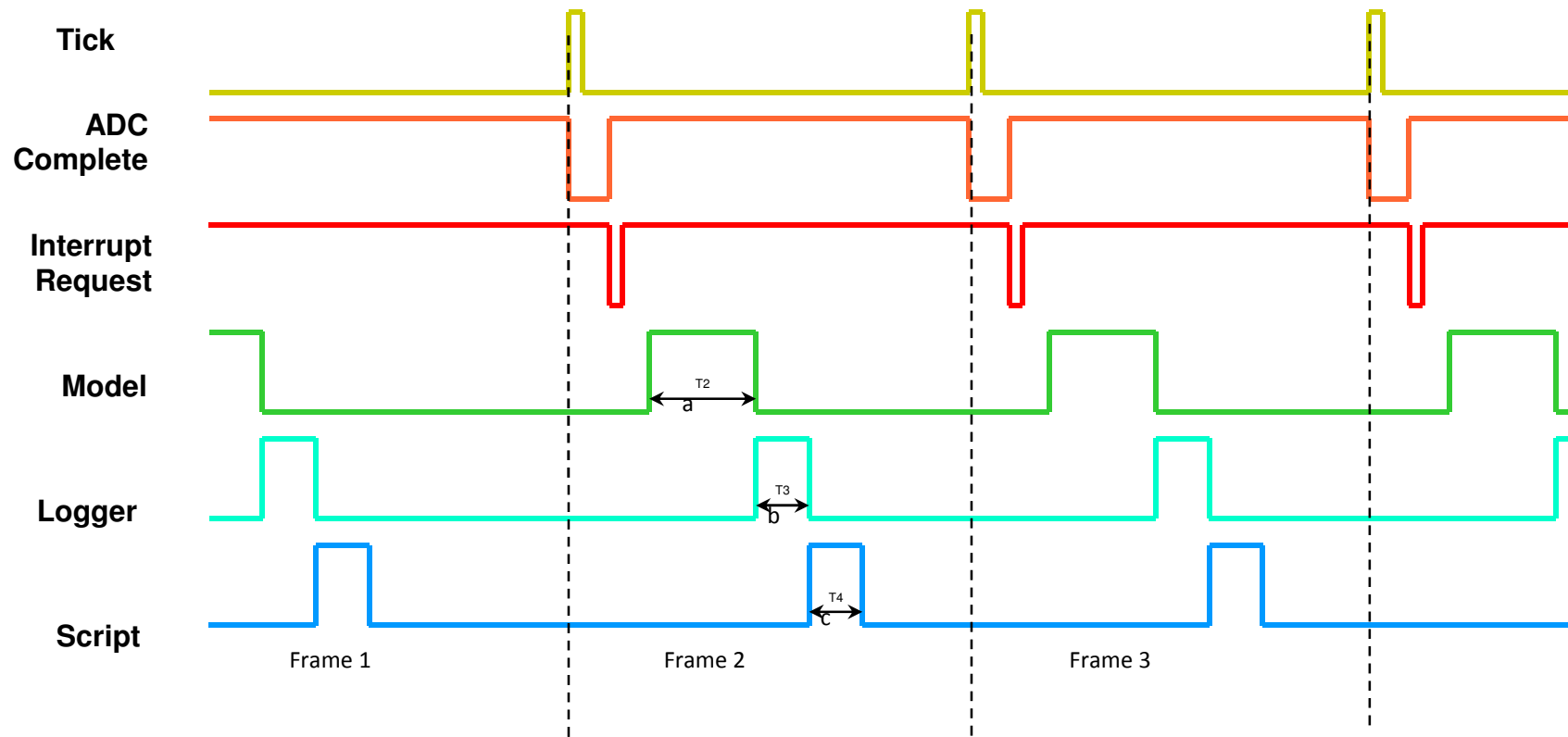
## An Engine Simulator

# The Simulation Environment

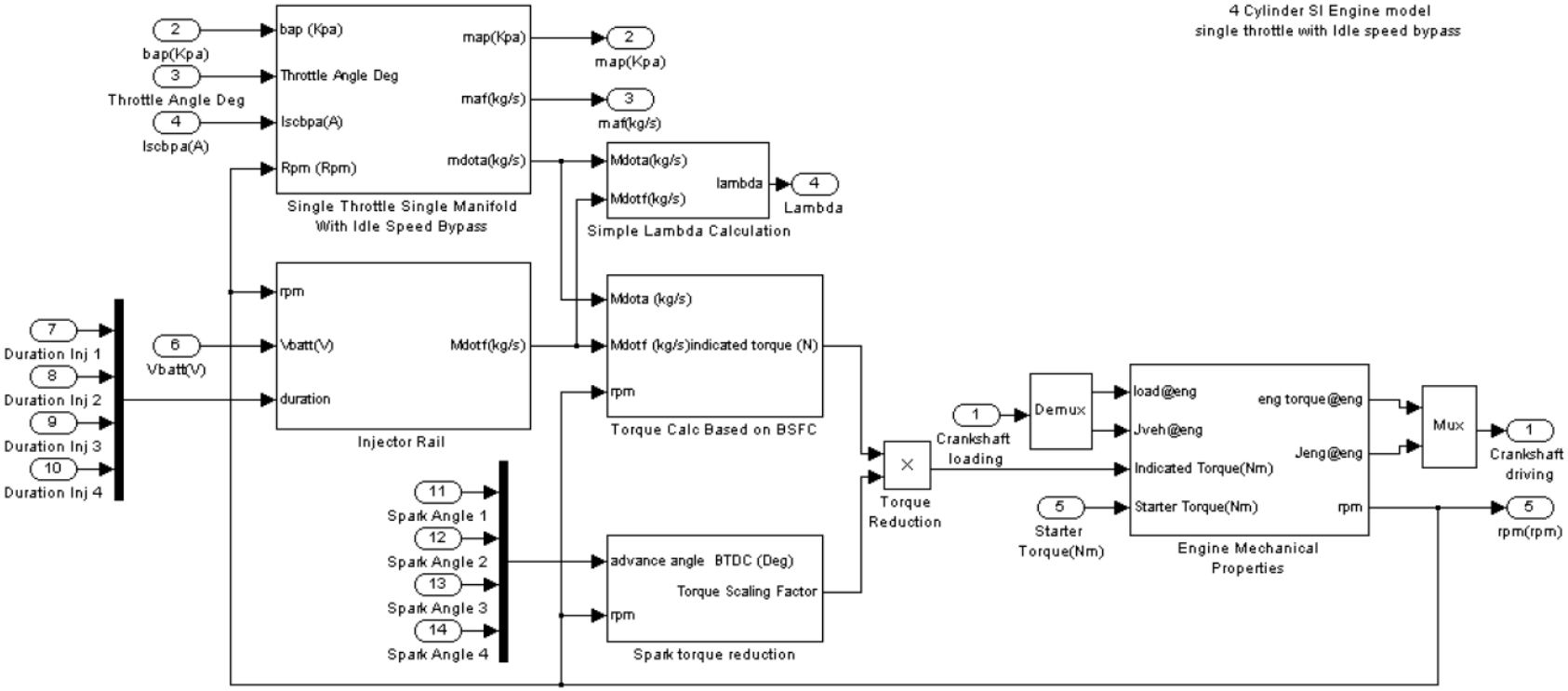


[www.pishurlok.com](http://www.pishurlok.com)

# The Simulation Frames




# Matlab design of the Engine Simulator





# Results on 100k loop simulation

CPU	No Auto-Vectorisation	With Auto-Vectorisation	Speedup
P4	39.344	21.9	<b>1.80</b>
Core 2	5.546	0.515	<b>10.77</b>
Speedup	<b>7.09</b>	<b>45.52</b>	<b>76</b>



# Vtune confirms reason for Speedup

CPU EVENT	Without Vect	With Vect
CPU_CLK_UNHALTED.CORE	16,641,000,448	1,548,000,000
INST_RETIRED.ANY	3,308,999,936	1,395,000,064
X87_OPS_RETIRED.ANY	250,000,000	0
SIMD_INST_RETIRED	0	763,000,000

Full paper available here: <http://edc.intel.com/Link.aspx?id=1045>

# Summary of Simulation Performance Improvements

- Performance gains through **migrating** to newer **silicon**
- Performance gains by using **Intel compiler.**

# Closing Remarks

- Try **Auto-vectorisation** – it can make a difference!
- **Out-of-the-box** use does not deliver the best optimisation
- If you are running on more than one generation of CPU use `-ax` (**CPU dispatching**)
- Use `-m` option on non-intel CPUs

# Any Questions



# Optimization Notice

## Optimization Notice

Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the “Intel® Compiler User and Reference Guides” under “Compiler Options.” Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel® compilers, associated libraries and associated development tools 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 Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® 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.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20101101

Software and Services Group



# Legal Disclaimer

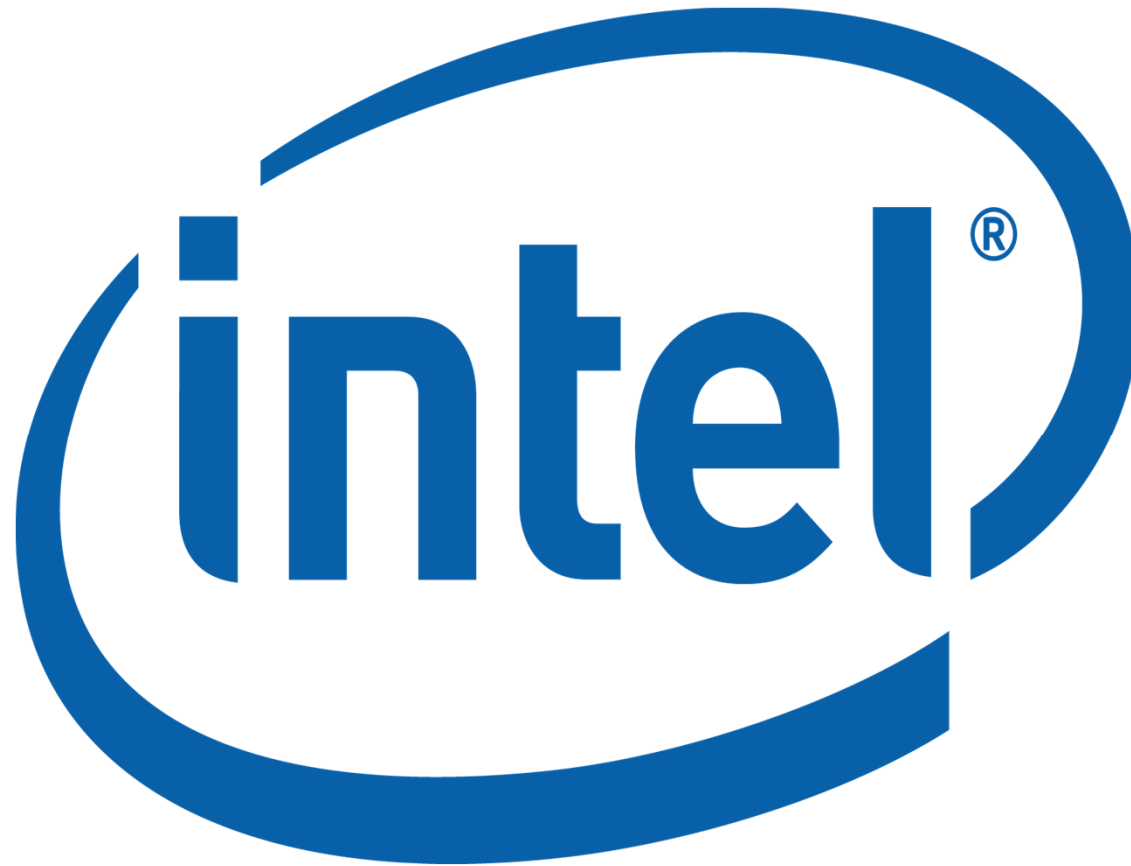
INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING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

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference [www.intel.com/software/products](http://www.intel.com/software/products).

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom, Centrino Atom Inside, Centrino Inside, Centrino logo, Cilk, Core Inside, FlashFile, i960, InstantIP, Intel, the Intel logo, Intel386, Intel486, IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skool, Sound Mark, The Journey Inside, Viiv Inside, vPro Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.

\*Other names and brands may be claimed as the property of others.

Copyright © 2010. Intel Corporation.



Software and Services Group





# Backup

