The significance of SIMD, SSE and AVX

Stephen Blair-Chappell
Intel Compiler Labs

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."

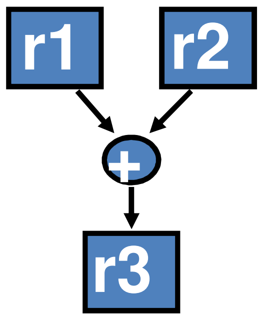
A customer when moving from version 9.1 to version 10 of the Intel compiler

Auto-Vectorisation

Vector Processing

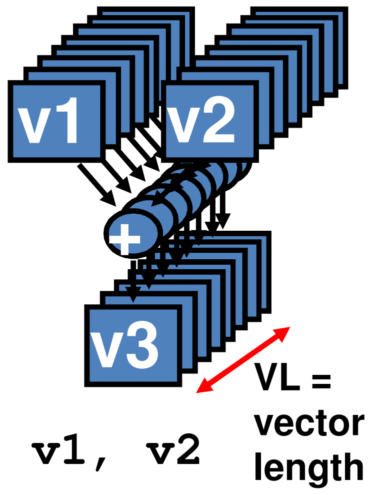
- A specific case of data level parallelism (DLP)
- Same operation simultaneously executed on N >1 elements of a vector.

Scalar Processing



```
add.d r3, r1, r2
```

Vector Processing

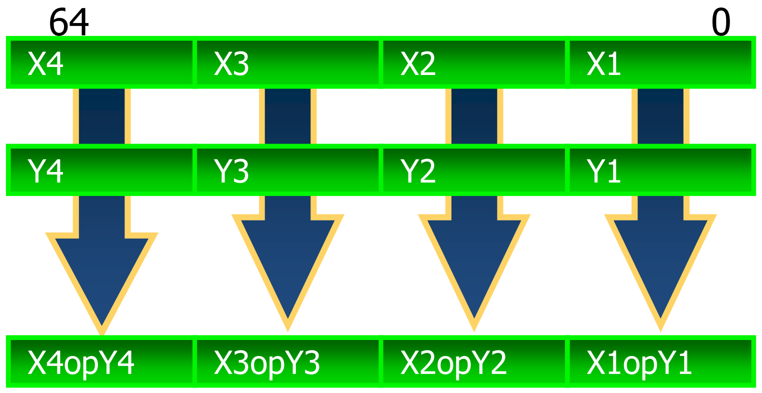


```
addvec.d v3, v1, v2
```

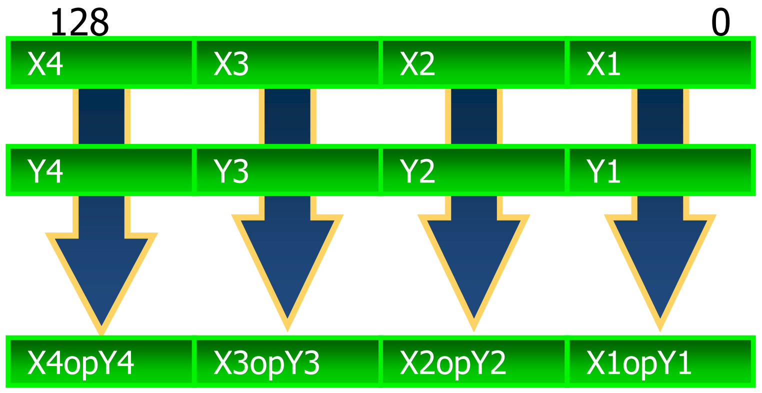
SIMD: Continuous Evolution

1999	2000	2004	2006	2007	2008	2009	2010\11
SSE	SSE2	SSE3	SSSE3	SSE4.1	SSE4.2	AES-NI	AVX
70 instr Single-Precision Vectors Streaming operations	144 instr Double-precision Vectors 8/16/32 64/128-bit vector integer	13 instr Complex Data	32 instr Decode	47 instr Video Graphics building blocks Advanced vector instr	8 instr String/XML processing POP-Count CRC	7 instr Encryption and Decryption Key Generation	~100 new instr. ~300 legacy sse instr updated 256-bit vector 3 and 4-operand instructions

SIMD Types in Processors from Intel [1]

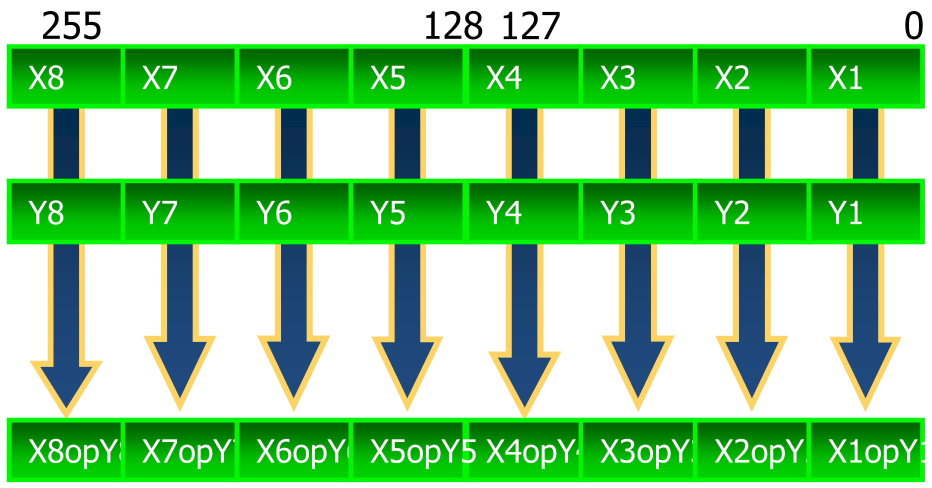


MMX™
Vector size: 64bit
Data types: 8, 16 and 32 bit integers
VL: 2,4,8
For sample on the left: Xi, Yi 16 bit integers



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]



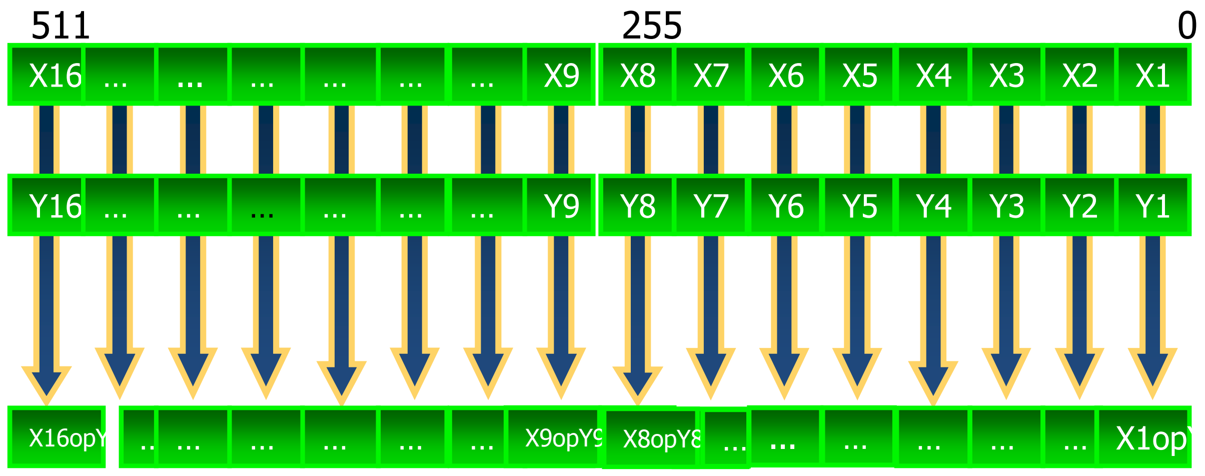
Intel® AVX

Vector size: 256bit

Data types: 32 and 64 bit floats

VL: 4, 8, 16

Sample: Xi, Yi 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

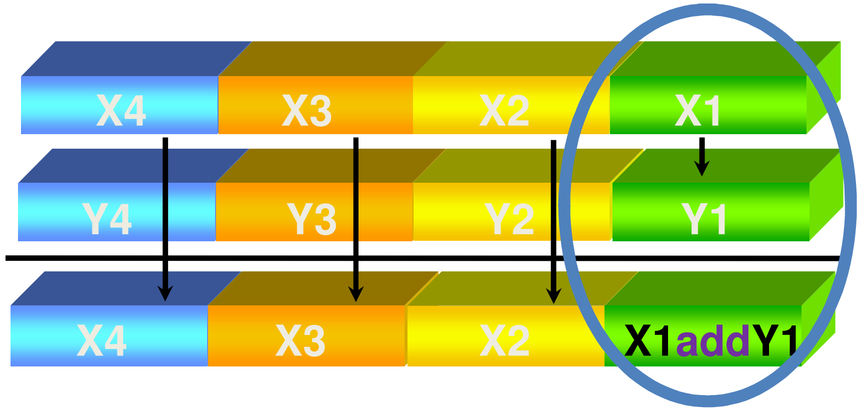
Scalar and Packed SSE Instructions

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

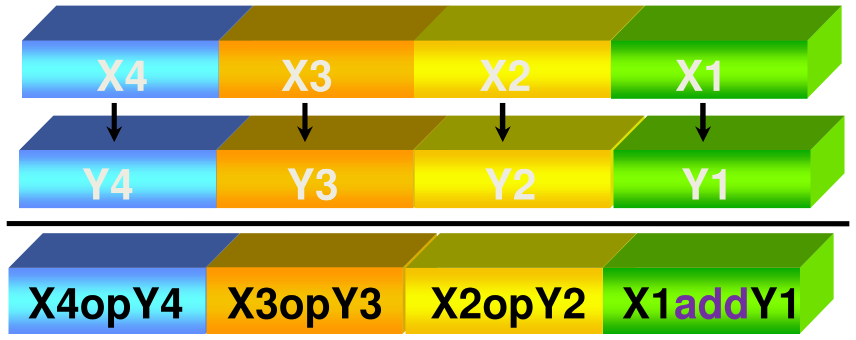
```
addss   Scalar Single-FP Add

          single precision FP data
         scalar execution mode
```

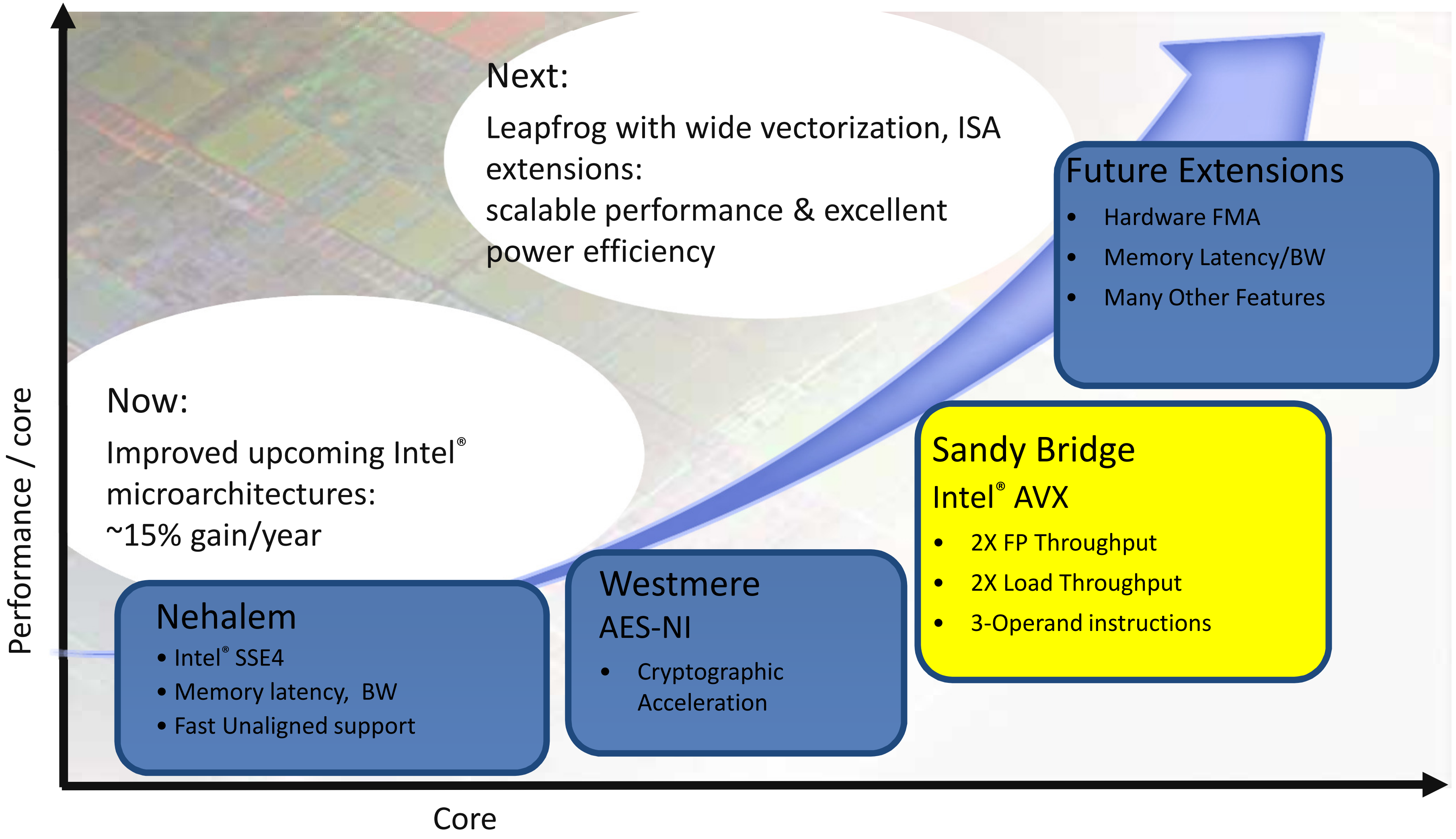


```
addps  Packed Single-FP Add

          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	BENEFITS
• Wider Vectors – Increased from 128 to 256 bit – Two 128-bit load ports	• Up to 2x peak FLOPs (floating point operations per second) output with good power efficiency
• Enhanced Data Rearrangement – Use the new 256 bit primitives to broadcast, mask loads and permute data	• Organize, access and pull only necessary data more quickly and efficiently
• Three and four Operands: Non Destructive Syntax for both AVX 128 and AVX 256	• Fewer register copies, better register use for both vector and scalar code
• Flexible unaligned memory access support	• More opportunities to fuse load and compute operations
• Extensible new opcode (VEX)	• 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, x`

1 less copy,
3 bytes smaller code size

xmm10 = xmm9 + m128	

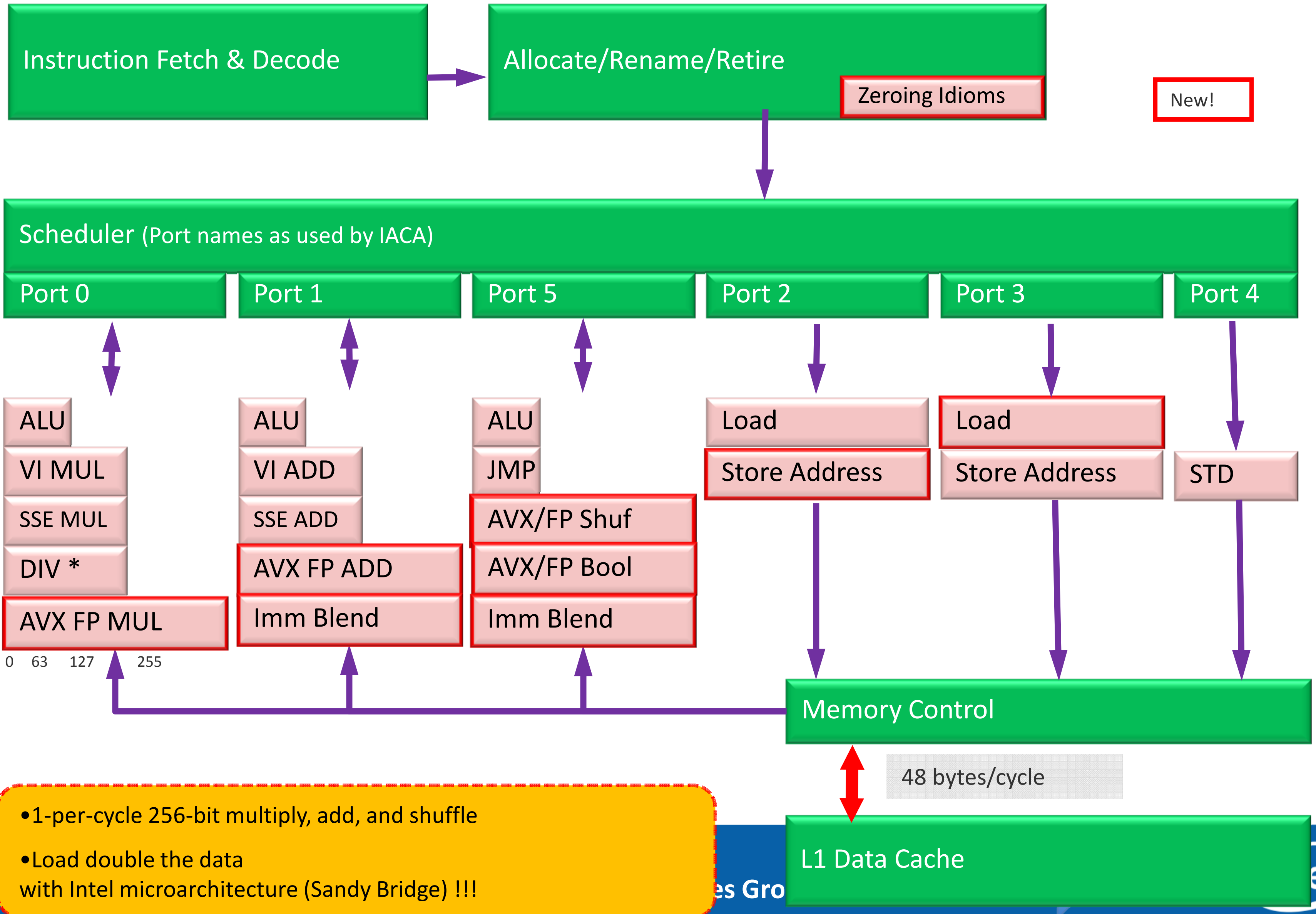
`movups xmm10, m128`
`addpd xmm10, xmm9`
	`vaddpd xmm10, xmm9,`

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

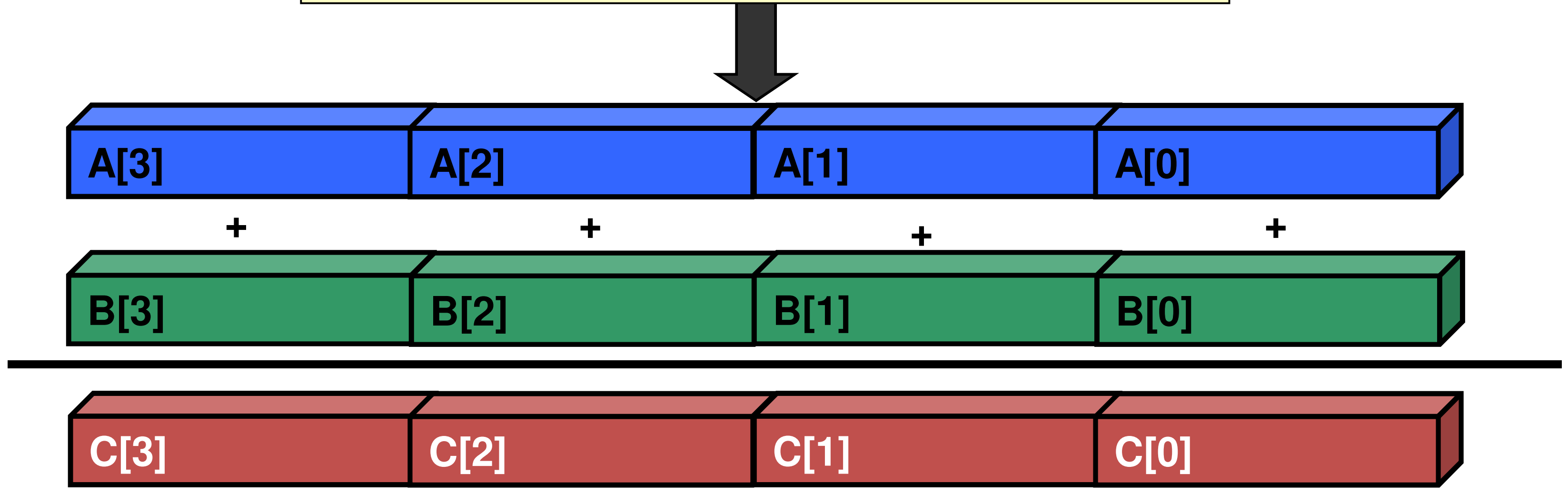
•Load double the data
with Intel microarchitecture (Sandy Bridge) !!!

* 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, …)

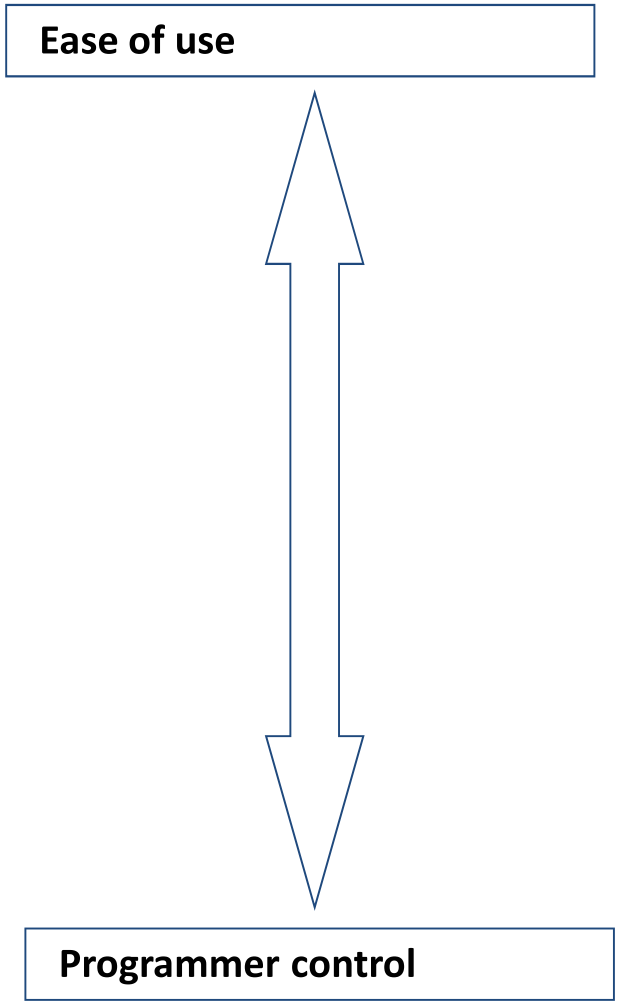
User Mandated Vectorization
(SIMD Directive)

Manual CPU Dispatch (__declspec(cpu_dispatch …))

SIMD intrinsic class (F32vec4 add)

Vector intrinsic (mm_add_ps())

Assembler code (addps)



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
  jl         .B1.2
```

SSE2

```
.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
  jl         .B1.2
```

SSE4.1

```
.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

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.

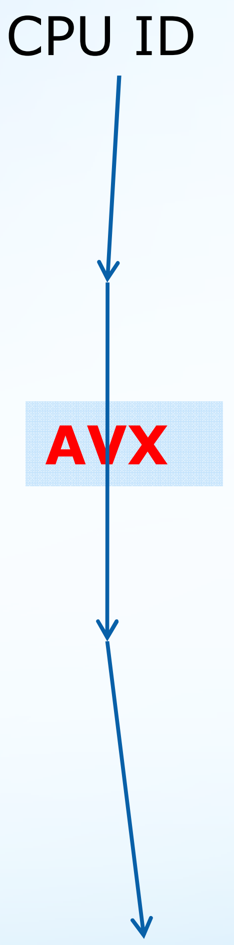
This option tells the compiler to generate code specialized for the processor that executes your program.
Code generated with these options should execute on any compatible, non-Intel processor with support for the corresponding instruction set.

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

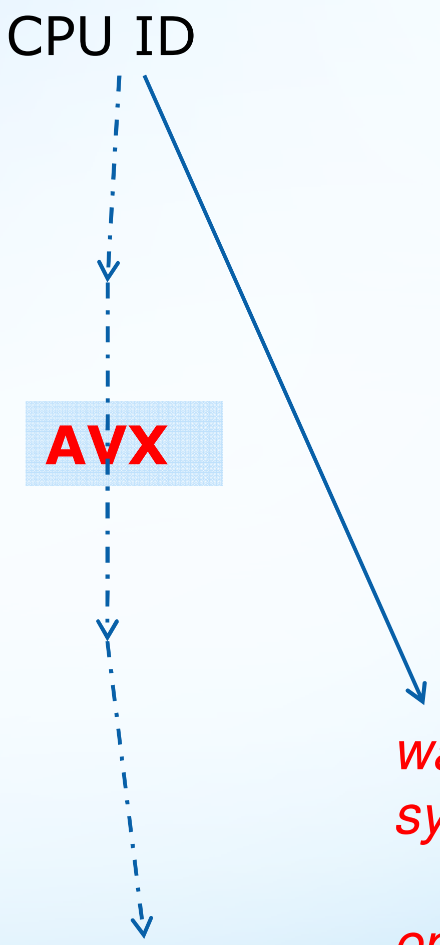


```
for(i=0;i<NUM;i++)
{
    j[i] = h[i] + i + 3
}
```

Running on a CPU supporting AVX

Auto-Vectorization

–xAVX



```
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

Optimization Notice

Using –ax compiler option …

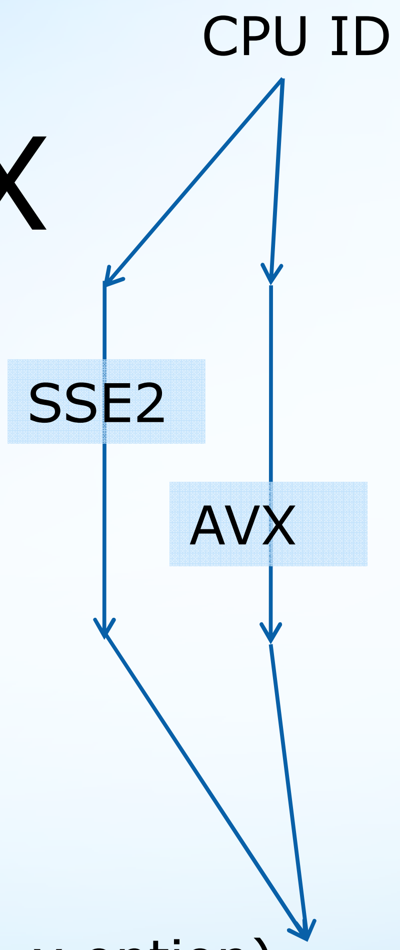
- Generates multiple paths if there is a performance benefit
- Generates a base line path
- Other options (e.g. -O3) control the base line path
- At runtime path chosen based on what processor code is running on

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

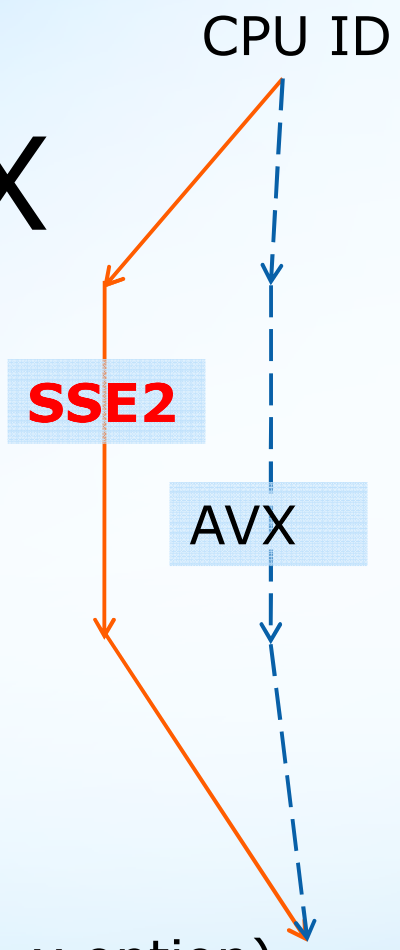


```
for(i=0;i<NUM;i++)
{
    j[i] = h[i] + i + 3
}
```

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

Generic low-spec CPU (no support of AVX)

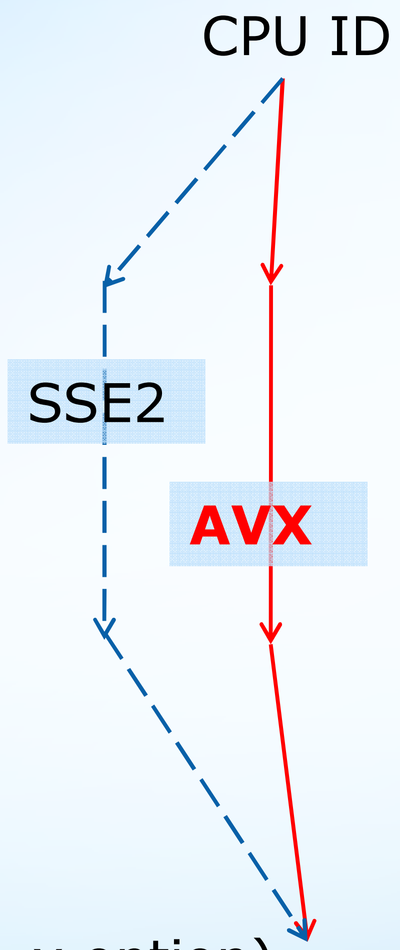
–axAVX



```
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

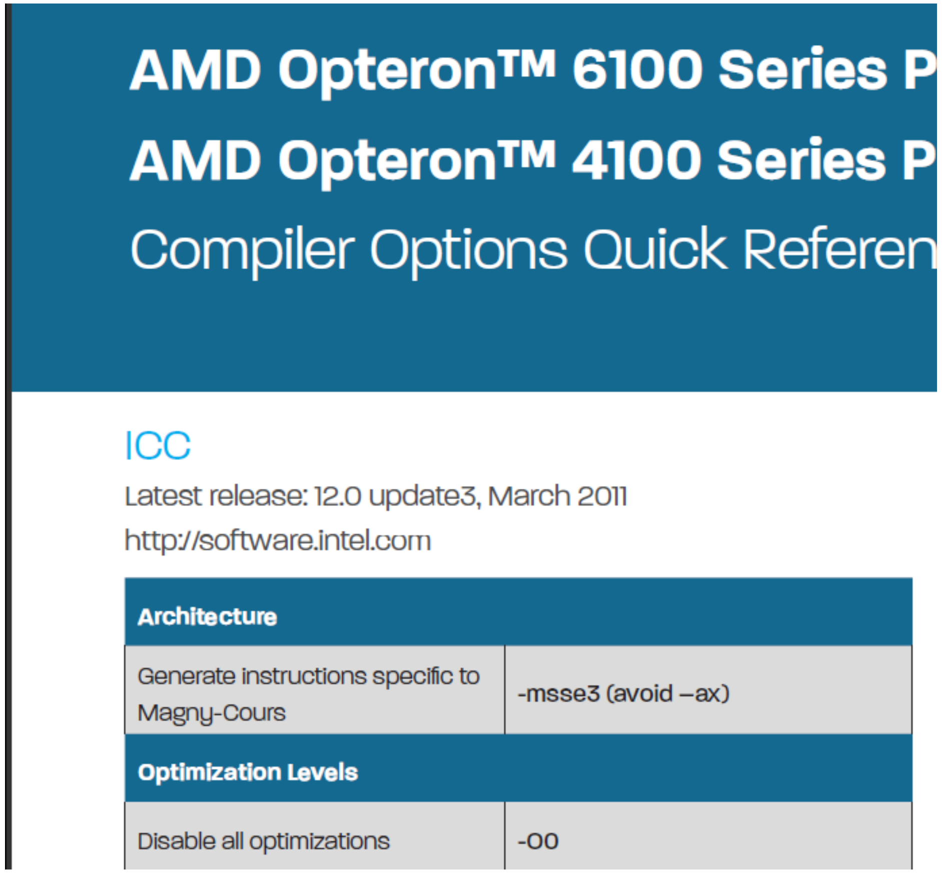
Running on Intel Processors

- 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. You application will only ever run on the same CPU as you 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. Your 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 __declespec(cpu_dispatch(cpuid,cpuid…)

CPUID Arguments

Argument for cpuid	Processors
future_cpu_16 (subject to change)	2nd generation Intel® Core™ processor family with support for Intel® Advanced Vector Extensions (Intel® AVX).
core_aes_pclmulqdq	Intel® Core™ processors with support for Advanced Encryption Standard (AES) instructions and carry-less multiplication instruction
core_i7_sse4_2	Intel® Core™ processor family with support for Intel® SSE4 Efficient Accelerated String and Text Processing instructions (SSE4.2)
atom	Intel® Atom™ processors
core_2_duo_sse4_1	Intel® 45nm Hi-k next generation Intel® Core™ microarchitecture processors with support for Intel® SSE4 Vectorizing Compiler and Media Accelerators instructions (SSE4.1)
core_2_duo_ssse3	Intel® Core™2 Duo processors and Intel® Xeon® processors with Intel® Supplemental Streaming SIMD Extensions 3 (SSSE3)
pentium_4_sse3	Intel® Pentium 4 processor with Intel® Streaming SIMD Extensions 3 (Intel® SSE3), Intel® Core™ Duo processors, Intel® Core™ Solo processors
pentium_4	Intel® Intel Pentium 4 processors
pentium_m	Intel® Pentium M processors
pentium_iii	Intel® Pentium III processors
generic	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;
}
```

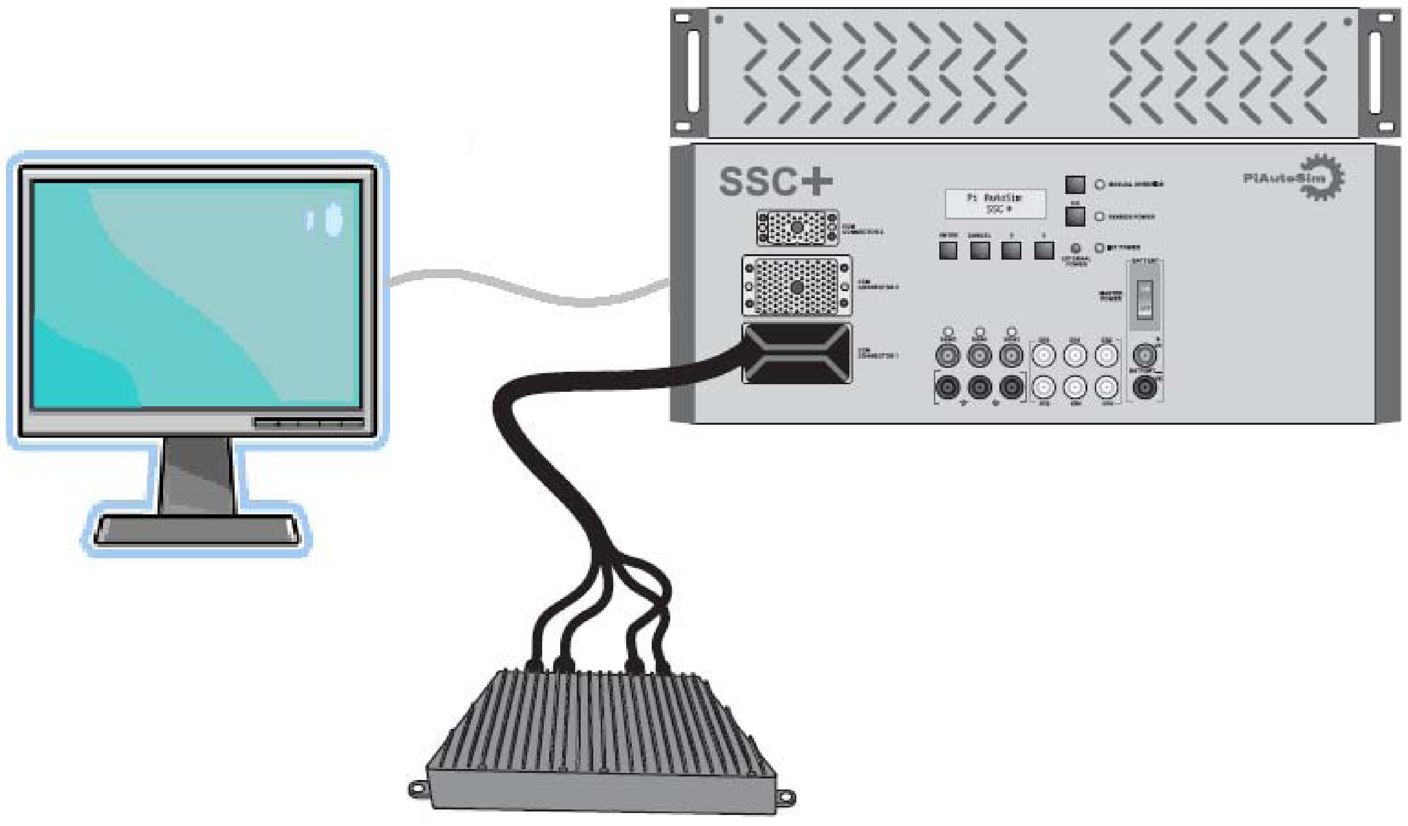
Questions to Ask

- 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 gong to run on just Intel CPUs?
- Will my application be running on non-intel processors?

A Case Study

An Engine Simulator

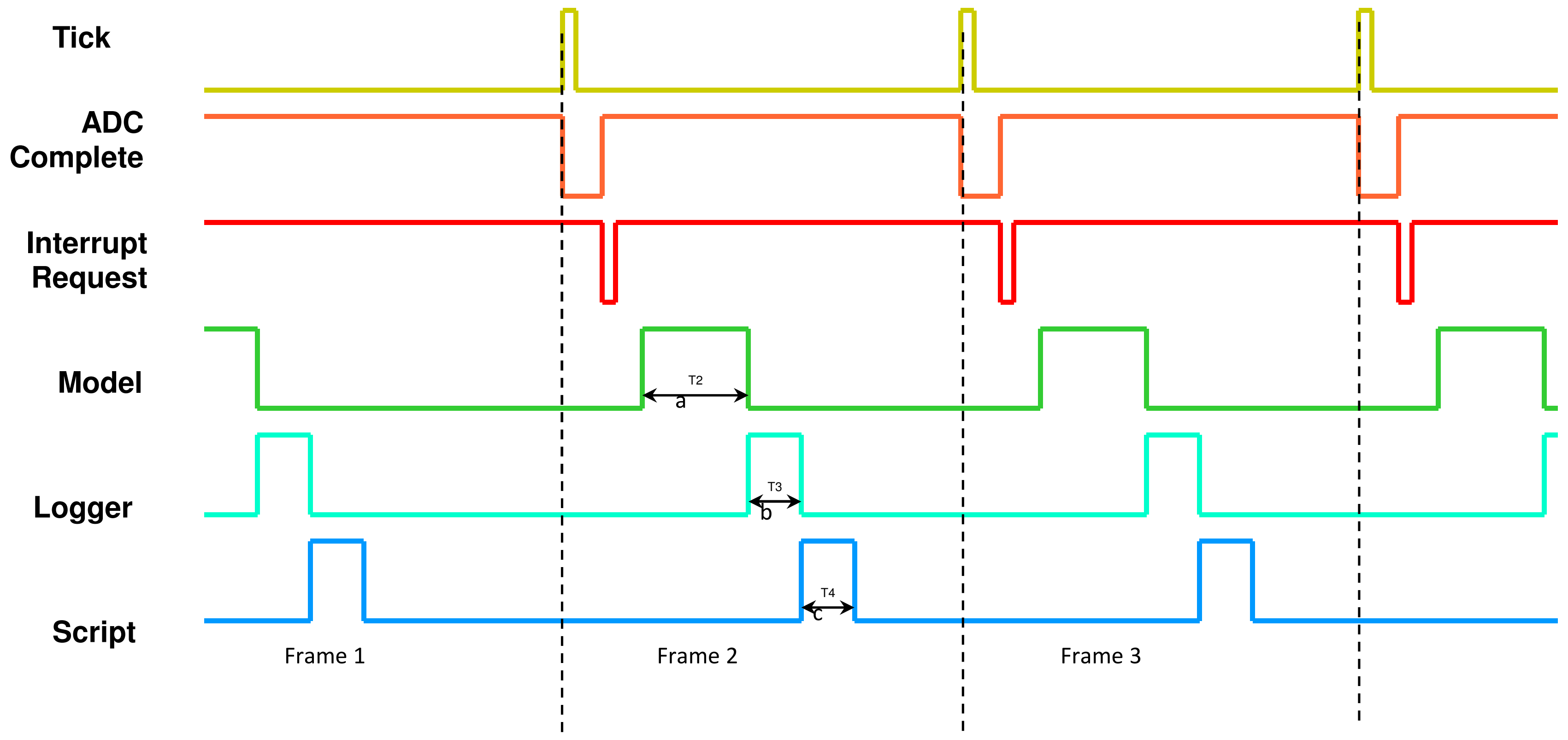
The Simulation Environment



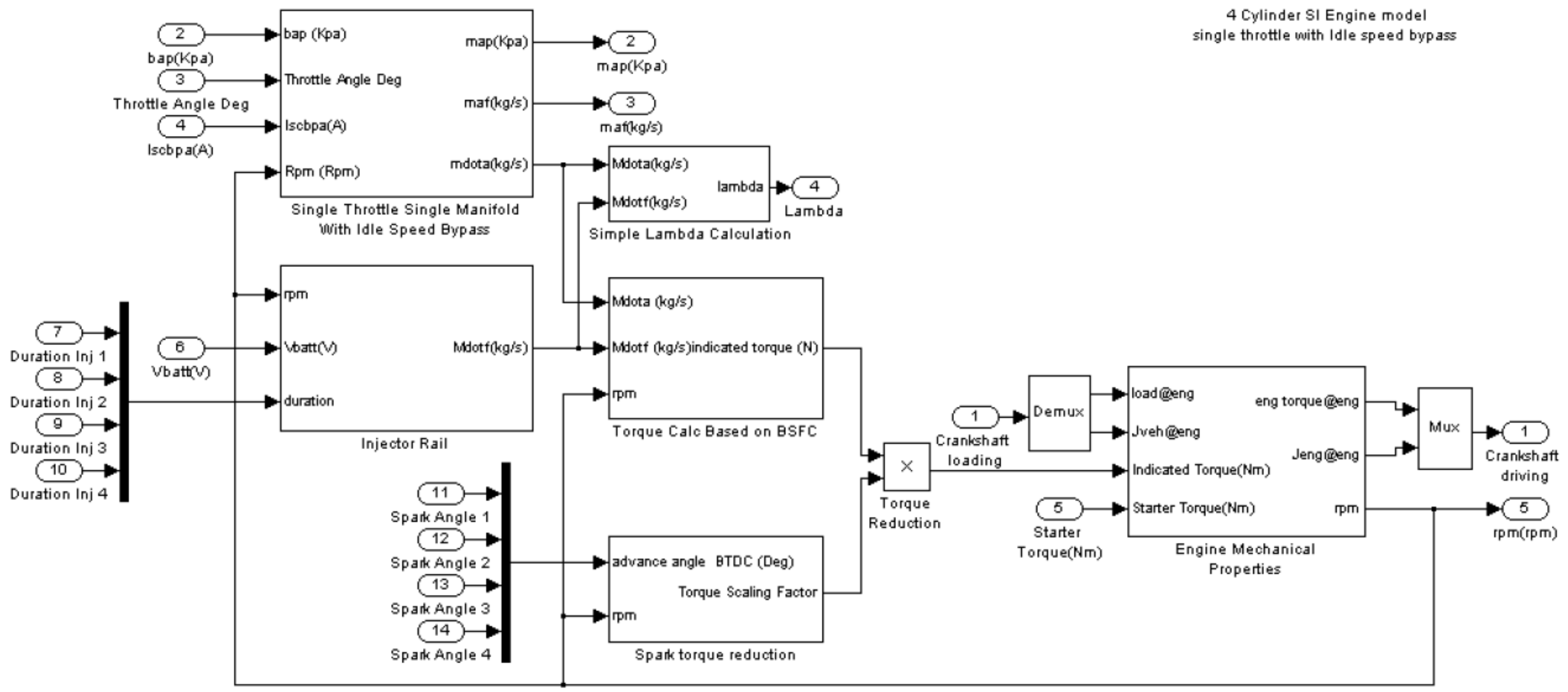
ECM under test

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	**1.80**
Core 2	5.546	0.515	**10.77**
Speedup	**7.09**	**45.52**	**76**

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

Legal Disclaimer

Backup