

`_vectorcall` and `__regcall` Demystified

The motivation to have the new `vectorcall` calling convention is to make use of as many registers as possible to pass function arguments rather than pushing the function arguments to stack and reading them from stack inside the function. Targeting this calling convention on performance critical functions can increase the performance since now most of the interaction is happening with registers (as demonstrated below with a code sample). For more information on `vectorcall` calling convention, please refer to the following URLs:

1. <http://msdn.microsoft.com/en-us/library/dn375768.aspx>
2. <http://blogs.msdn.com/b/vcblog/archive/2013/07/12/introducing-vector-calling-convention.aspx>

Intel® C++ Compiler 14.0 Update 1 doesn't support `_vectorcall` calling convention. Instead users can use `__regcall` calling convention which is a workaround for `_vectorcall` (demonstrated in this article).

Code Snippet:

```
#include<intrin.h>
#include<iostream>
using namespace std;
struct Point{
    __m128 x, y;
};
__declspec(noinline) Point add(Point p1, Point p2){
    Point p3;
    p3.x = _mm_add_ps(p1.x, p2.x);
    p3.y = _mm_add_ps(p1.y, p2.y);
    return p3;
}
int main(int argc, char *argv[]){
    Point a[16], b[16], c[16], sum;
    memset(&sum, 0, 32);
    for (int i = 0; i < 16; i++)
        c[i] = add(a[i], b[i]);
    for (int i = 0; i < 16; i++)
    {
        sum.x = _mm_add_ps(sum.x, c[i].x);
        sum.y = _mm_add_ps(sum.y, c[i].y);
    }
    return 0;
}
```

When this code is compiled with `__cdecl` calling convention for Win32 using VS2013 Compiler, it errors out as shown below:

```
C:\Users\amadhuso\Documents\Visual Studio 2013\Projects\Elemvector\Elemvector>cl.exe
Elemvector.cpp /arch:AVX
```

Microsoft (R) C/C++ Optimizing Compiler Version 18.00.21005.1 for x86
Copyright (C) Microsoft Corporation. All rights reserved.

Elemvector.cpp

You are using an Intel supplied intrinsic header file with a third-party compiler.

C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\INCLUDE\xlocale(337) : warning C4530: C++ exception handler used, but unwind semantics are not enabled. Specify /EHsc

Elemvector.cpp(7) : error C2719: 'p1': formal parameter with __declspec(align('16')) won't be aligned

Elemvector.cpp(7) : error C2719: 'p2': formal parameter with __declspec(align('16')) won't be aligned

Intel® C++ Compiler doesn't encounter this problem. We handle this case well.

This article is intended to analyze the code generated by VS2013 compiler and ICL 14.0 Update 1 (x64 build). Analyzing the ASM will give you good insight as to how each calling convention passes the function arguments / return value.

ASM generated by VS2013 compiler (_cdecl calling convention):

>cl.exe /O2 /arch:AVX /FA Elemvector.cpp /EHsc

Microsoft (R) C/C++ Optimizing Compiler Version 18.00.21005.1 for x64
Copyright (C) Microsoft Corporation. All rights reserved.

Elemvector.cpp

You are using an Intel supplied intrinsic header file with a third-party compiler.

Microsoft (R) Incremental Linker Version 12.00.21005.1

Copyright (C) Microsoft Corporation. All rights reserved.

/out:Elemvector.exe

Elemvector.obj

\$LL6@main:

; 17 : c[i] = add(a[i], b[i]);

vmovups ymm0, YMMWORD PTR b[rsp+r9]

lea r8, QWORD PTR \$T2[rsp]

lea rdx, QWORD PTR \$T1[rsp]

lea rcx, QWORD PTR \$T3[rsp]

vmovups YMMWORD PTR \$T2[rsp], ymm0

vmovups YMMWORD PTR \$T1[rsp], ymm0

vzeroupper

call ?add@@YA?AUPoint@@U1@0@Z ; add

add r9, 32 ; 00000020H

vmovups ymm0, YMMWORD PTR [rax]

vmovups YMMWORD PTR c[rsp+r9-32], ymm0

```

        cmp     r9, 512                                ; 00000200H
        jl      SHORT $LL6@main

;      COMDAT ?add@@YA?AUPoint@@U1@0@Z
_TEXT SEGMENT
$T1 = 8
p1$ = 16
p2$ = 24
?add@@YA?AUPoint@@U1@0@Z PROC                ; add, COMDAT
; 8 :   Point p3;
; 9 :   p3.x = _mm_add_ps(p1.x, p2.x);
        vmovups    ymm1, YMMWORD PTR [rdx]
; 10 :  p3.y = _mm_add_ps(p1.y, p2.y);
; 11 :  return p3;
        mov     rax, rcx
        vaddps    ymm1, ymm1, YMMWORD PTR [r8]
        vmovups    YMMWORD PTR [rcx], ymm1
; 12 : }
        ret     0
?add@@YA?AUPoint@@U1@0@Z ENDP                ; add

```

From the generated ASM, it is clear that function arguments/return value is passed in stack and not in registers though the actual computation happens in registers. The move instructions in blue font demonstrate this.

ASM generated by ICC 14.0 Update 1 (`_cdecl` calling convention):

>icl.exe /O2 /QxAVX /FAs Elemvector.cpp /EHsc

Intel(R) C++ Intel(R) 64 Compiler XE for applications running on Intel(R) 64, Version 14.0.1.139 Build 20131008

Copyright (C) 1985-2013 Intel Corporation. All rights reserved.

Elemvector.cpp

Microsoft (R) Incremental Linker Version 12.00.21005.1

Copyright (C) Microsoft Corporation. All rights reserved.

-out:Elemvector.exe

Elemvector.obj

```

.B1.3::                ; Preds .B1.10 .B1.2
;;;                    c[i] = add(a[i], b[i]);
        lea     rcx, QWORD PTR [1664+rsp]            ;17.20
        vmovups xmm0, XMMWORD PTR [48+rsp+rdi]      ;17.20

```

```

mov     rdx, rsi                                ;17.20
vmovups xmm1, XMMWORD PTR [32+rsp+rdi]         ;17.20
mov     r8, rbx                                ;17.20
vmovups XMMWORD PTR [16+rsi], xmm0             ;17.20
vmovups XMMWORD PTR [rsi], xmm1               ;17.20
vmovups xmm2, XMMWORD PTR [560+rsp+rdi]         ;17.20
vmovups xmm3, XMMWORD PTR [544+rsp+rdi]         ;17.20
vmovups XMMWORD PTR [16+rbx], xmm2            ;17.20
vmovups XMMWORD PTR [rbx], xmm3               ;17.20
call    ?add@@YA?AUPoint@@@U1@0@Z            ;17.20
.B1.10::                                       ; Preds .B1.3
vmovups xmm0, XMMWORD PTR [1664+rsp]           ;17.20
inc     r12b                                   ;16.26
vmovups xmm1, XMMWORD PTR [1680+rsp]           ;17.20
vmovups XMMWORD PTR [1056+rsp+rdi], xmm0       ;17.3
vmovups XMMWORD PTR [1072+rsp+rdi], xmm1       ;17.3
add     rdi, 32                               ;16.26
cmp     r12b, 16                             ;16.22
jl      .B1.3                                ; Prob 93%    ;16.22

```

?add@@YA?AUPoint@@@U1@0@Z PROC

; parameter 1: [rdx]

; parameter 2: [r8]

.B2.1:: ; Preds .B2.0

```

mov     rax, rcx                                ;11.10
vmovups xmm0, XMMWORD PTR [rdx]                ;9.21
vmovups xmm1, XMMWORD PTR [16+rdx]             ;10.21
vaddps  xmm3, xmm0, XMMWORD PTR [r8]           ;9.10
vaddps  xmm2, xmm1, XMMWORD PTR [16+r8]        ;10.10
vmovups XMMWORD PTR [16+rcx], xmm2            ;11.10
vmovups XMMWORD PTR [rcx], xmm3               ;11.10
ret                                           ;11.10

```

Even Intel® C++ Compiler does similar transfer of function arguments (obviously because `__cdecl` calling convention defines this). As shown above XMM0 is mapped to `a[i].x`, XMM1 is mapped to `a[i].y`, XMM2 is mapped to `b[i].x` and XMM3 is mapped to `b[i].y`. ASM clearly shows that first the values of the array are fetched from memory to a Intel® XMM register and then before the `add()` function call, those values are copied to stack. Inside the function, the values are taken from stack into Intel® XMM register and after the computation, the result is pushed back to stack from the register.

ASM generated by VS2013 compiler (`_vectorcall` calling convention):

The only code change is as shown below:

From:

```
__declspec(noinline) Point add(Point p1, Point p2){
    Point p3;
    p3.x = _mm_add_ps(p1.x, p2.x);
    p3.y = _mm_add_ps(p1.y, p2.y);
    return p3;
}
```

To

```
__declspec(noinline) Point _vectorcall add(Point p1, Point p2){
    Point p3;
    p3.x = _mm_add_ps(p1.x, p2.x);
    p3.y = _mm_add_ps(p1.y, p2.y);
    return p3;
}
```

>cl.exe /O2 /arch:AVX /FAs Elemvector.cpp /EHsc

Microsoft (R) C/C++ Optimizing Compiler Version 18.00.21005.1 for x64

Copyright (C) Microsoft Corporation. All rights reserved.

Elemvector.cpp

You are using an Intel supplied intrinsic header file with a third-party compiler.

Microsoft (R) Incremental Linker Version 12.00.21005.1

Copyright (C) Microsoft Corporation. All rights reserved.

/out:Elemvector.exe

Elemvector.obj

Below is the corresponding ASM

\$LL6@main:

```
vmovups    xmm2, XMMWORD PTR b$[rsp+rdi]
vmovups    xmm3, XMMWORD PTR b$[rsp+rdi+16]
vmovups    xmm0, xmm2
vmovups    xmm1, xmm3
vzeroupper
call       ?add@@YQ?AUPoint@@@U1@0@Z      ; add
lea        rsi, QWORD PTR [rsi+32]
lea        rdi, QWORD PTR [rdi+32]
vmovups    XMMWORD PTR $T1[rsi+16], xmm1
vmovups    XMMWORD PTR $T1[rsi], xmm0
vmovups    ymm0, YMMWORD PTR $T1[rsi]
vmovups    YMMWORD PTR [rsi-32], ymm0
```

```

dec     rbp
jne     SHORT $LL6@main

```

```

?add@@@YQ?AUPoint@@@U1@0@Z PROC                ; add, COMDAT

```

```

sub     rsp, 24
vmovaps     XMMWORD PTR [rsp], xmm6
vmovaps     xmm6, XMMWORD PTR [rsp]
vaddps     xmm0, xmm0, xmm2
vaddps     xmm1, xmm1, xmm3
add     rsp, 24
ret     0

```

Unlike the first case, the values of the array are passed in Intel® XMM registers this time rather through the stack.

Intel® C++ Compiler 14.0 Update 1 doesn't support `_vectorcall` calling convention. Instead you can request the compiler to use as many as registers as possible during function call by specifying `__regcall` calling convention. Below is the demonstration of the same. The only code change is as follows:

From:

```

__declspec(noinline) Point _vectorcall add(Point p1, Point p2){
    Point p3;
    p3.x = _mm_add_ps(p1.x, p2.x);
    p3.y = _mm_add_ps(p1.y, p2.y);
    return p3;
}

```

To:

```

__declspec(noinline) Point __regcall add(Point p1, Point p2){
    Point p3;
    p3.x = _mm_add_ps(p1.x, p2.x);
    p3.y = _mm_add_ps(p1.y, p2.y);
    return p3;
}

```

ASM generated by ICL 14.0 Update 1 Compiler (`__regcall` calling convention):

```
>icl.exe /O2 /QxAVX /FAs Elemvector.cpp /EHsc /Qregcall
```

Intel(R) C++ Intel(R) 64 Compiler XE for applications running on Intel(R) 64, Version 14.0.1.139 Build 20131008

Copyright (C) 1985-2013 Intel Corporation. All rights reserved.

Elemvector.cpp

Microsoft (R) Incremental Linker Version 12.00.21005.1

Copyright (C) Microsoft Corporation. All rights reserved.

-out:Elemvector.exe

Elemvector.obj

```
.B1.3::                ; Preds .B1.10 .B1.2
    vmovdqu  xmm0, XMMWORD PTR [64+rsp+rbp]      ;17.20
    vmovdqu  xmm1, XMMWORD PTR [80+rsp+rbp]      ;17.20
    vmovdqu  xmm2, XMMWORD PTR [576+rsp+rbp]     ;17.20
    vmovdqu  xmm3, XMMWORD PTR [592+rsp+rbp]     ;17.20
    call     ?__regcall2__add@@@YE?AUPoint@@@U1@0@Z ;17.20
.B1.10::               ; Preds .B1.3
    inc     r15b                ;16.26
    vmovups  XMMWORD PTR [1088+rsp+rbp], xmm0     ;17.3
    vmovups  XMMWORD PTR [1104+rsp+rbp], xmm1     ;17.3
    add     rbp, 32              ;16.26
    cmp     r15b, 16             ;16.22
    jl      .B1.3               ; Prob 93%         ;16.22

?__regcall2__add@@@YE?AUPoint@@@U1@0@Z PROC
; parameter 1: xmm0 xmm1
; parameter 2: xmm2 xmm3
.B2.1::                ; Preds .B2.0
    vaddps  xmm0, xmm2, xmm0     ;9.10
    vaddps  xmm1, xmm3, xmm1     ;10.10
    ret
```

By using __regcall calling convention, the function arguments and return value are passed using Intel® XMM registers rather than through stack. _vectorcall calling convention will introduced in a future version of the Intel® C++ Compiler.