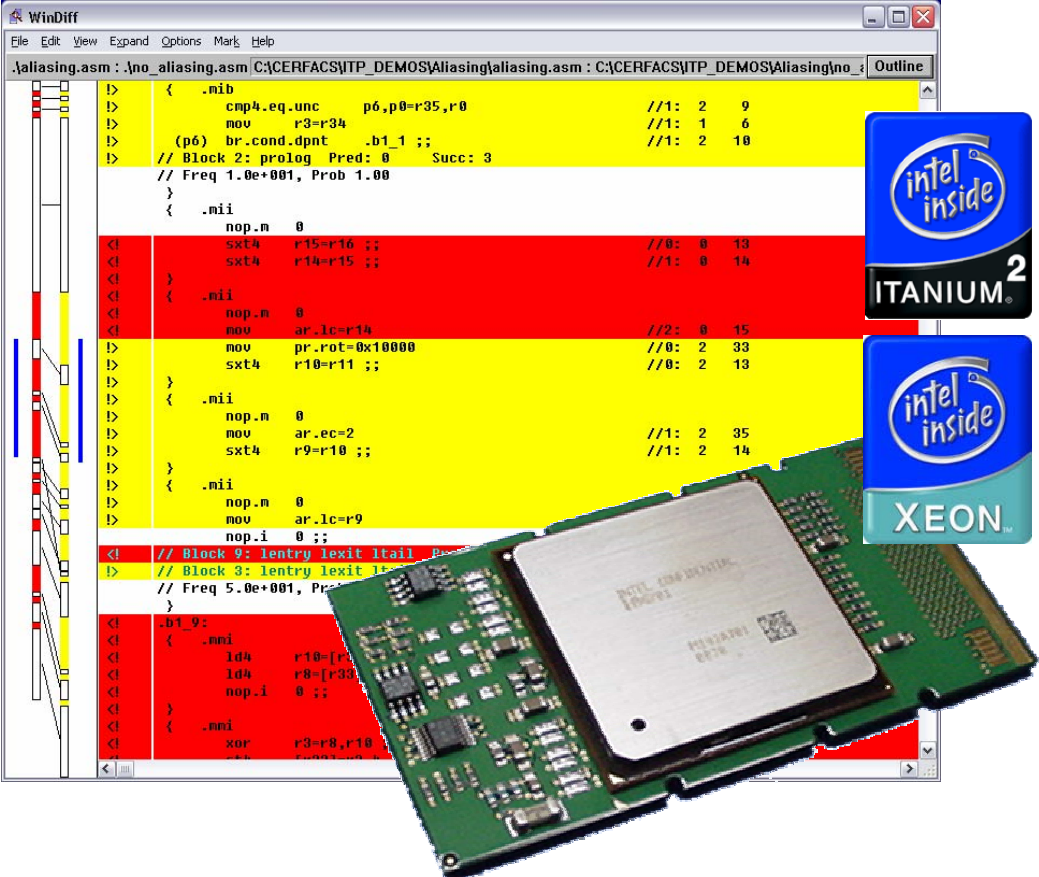


# VTune™ & Intel® Compilers Overview Thru Small Examples

**11-Feb-2003**  
**Toulouse, France**

Jamel Tayeb  
Software Enabling Group  
Intel EMEA, Paris  
jamel.tayeb@intel.com



The image shows a screenshot of the WinDiff application window. The window title is "WinDiff" and it displays a comparison of assembly code between two files: ".\aliasing.asm" and "C:\CERFACS\I7P\_DEMOS\Aliasing\aliasing.asm". The code is color-coded: yellow for the original code and red for the modified code. The assembly code includes instructions like "cmp4.eq.unc", "mov", "br.cond.dpnt", "nop.n", "sxt4", "ar.lc=r14", "pr.rot=0x10000", "ar.ec=2", "ar.lc=r9", "nop.i", "ld4", and "xor". Comments indicate block boundaries and frequencies, such as "// Freq 1.0e+001, Prob 1.00" and "// Freq 5.0e+001, Prob 1.00".

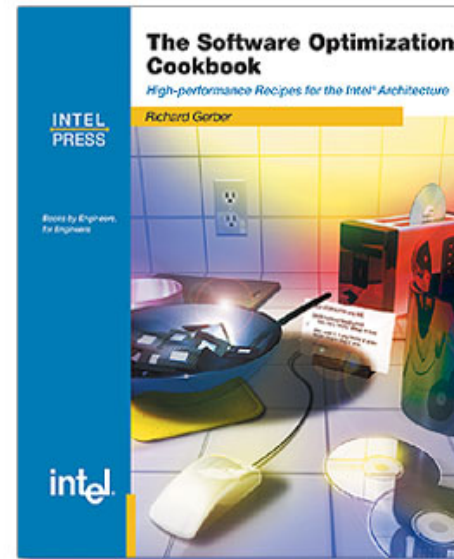
Overlaid on the right side of the screenshot are two Intel logos: "intel inside ITANIUM 2" and "intel inside XEON". Below the logos is a photograph of an Intel Xeon processor mounted on a green printed circuit board (PCB).



# References:



close window



close window

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

# Intel Software Development Tools

## SW Products

### Compilers



### Performance Libraries



### VTune™ Performance Analyzer



### Intel® Threading Tools



## Developer Services

### Intel's Developer Site

For Hardware, Software and Internet Developers

#### Introducing the Intel® Pentium® 4 Processor at 2.20 GHz

The Intel® Pentium® 4 processor, now available at 2.20 GHz, is the next evolutionary step for desktop processor technology.



Week of February 10th, 2002

#### Highlights

- Developed a great application in Retail? Get the recognition you deserve!
- New Intel transceivers enable high-density, low-power 10 gigabit communications equipment
- New Intel-Architecture-based 1U and 2U Carrier Grade Servers for Telecom Deployment
- Intel Wins Applied Computing Designs Based on Embedded Intel® Pentium® 4 Processor
- Advancing the Digital Universe: Intel® Networking and Communications at IDF
- Intel Introduces Seven New Mobile Processors
- Intel® chipsets provide a full range of platform choices for the Intel® Pentium® 4 processor
- New for Embedded Applications: the Intel® Pentium® 4 Processor at 2A GHz and the Intel® 845 Chipset
- Intel announces the launch of the Service Availability\* Forum

Roll your cursor over any adjacent link to see an overview.

#### Community

- Developer Update Magazine (IDU)
- Intel® Developer Forum (IDF)
- Intel® Developer Server & Workstation Newsletter
- Intel® PCA Developer Network
- Subscribe to Intel® Developer Networking and Communications News
- Intel® Labs
- Intel Press Books
- Intel Technology Journal (ITJ)
- The Press Room

#### Services

- Intel® Developer Services
- Intel® Microelectronics Services
- Intel® Solution Services

#### Tools & Resources

[www.intel.com/ids](http://www.intel.com/ids)



# Agenda

- VTune™ in action on NetBurst™ Architecture
- C/C++ compiler to target Itanium™ & Itanium® 2 processors

# Compile a release and optimized binary with debug information

**Selection Tool**

IA-32 Compiler Selection

Intel C++ Compiler 7.0 Version

Uses environment variables in Tools->Options->Directories

OK Cancel Help

Intel Itanium(TM) Compiler / Environment Selection

REQUIRES CHANGES TO PROJECT SETTINGS! SEE "Help"  
You must add WIN64 and /machine:ia64 to compiler, linker and librarian settings to invoke any Intel Itanium(TM) compiler. Without these options, you will get an IA-32 compilation.

Intel C++ Compiler 7.0 Version

Use Environment Variables Listed Below  
Overrides environment variables in Tools->Options->Directories

Path C:\Program Files\Intel\Select\bin\C:\Program Files\Common File

Include C:\Program Files\Microsoft SDK\include\PreRelease\C:\Program

Lib C:\Program Files\Microsoft SDK\lib\IA64\C:\Program Files\Micro

**Project Settings**

Settings For: Win32 Release

Category: General

Warning Level: Level 3

Optimizations: Default

Warnings as errors  Generate browse info

Debug info: Program Database

Preprocessor definitions: WIN32NDEBUG, \_WINDOWS

Project Options: "NDEBUG" /D "\_WINDOWS" /Fp"Release/RotateBlend.pch" /X /Fo"Release/" /Fd"Release/" /FD /O3 /Qxk /c

OK Cancel

```

// foreground image
#define FGBMP_FILENAME "foreground.bmp"
DWORD *pFgBmp = NULL;
DWORD FgBmpWidth, FgBmpHeight, FgEmpPitch;
POINT FgPos = { 0, 0};
POINT FgDir = {+2, 0};
float Angle = 0.0f, AngleDir = -0.04f;

void UpdatePositionAndRotation (void)
{
    Angle += AngleDir;
    FgPos.x += FgDir.x;

    if (FgPos.x < 0)
    {
        FgPos.x = 0;
        FgDir.x = -FgDir.x;
        AngleDir *= -1;
    }
    else if (FgPos.x >= FgBmpWidth)
    {
        FgPos.x = FgBmpWidth - 1;
        FgDir.x = -FgDir.x;
        AngleDir *= -1;
    }
}

```

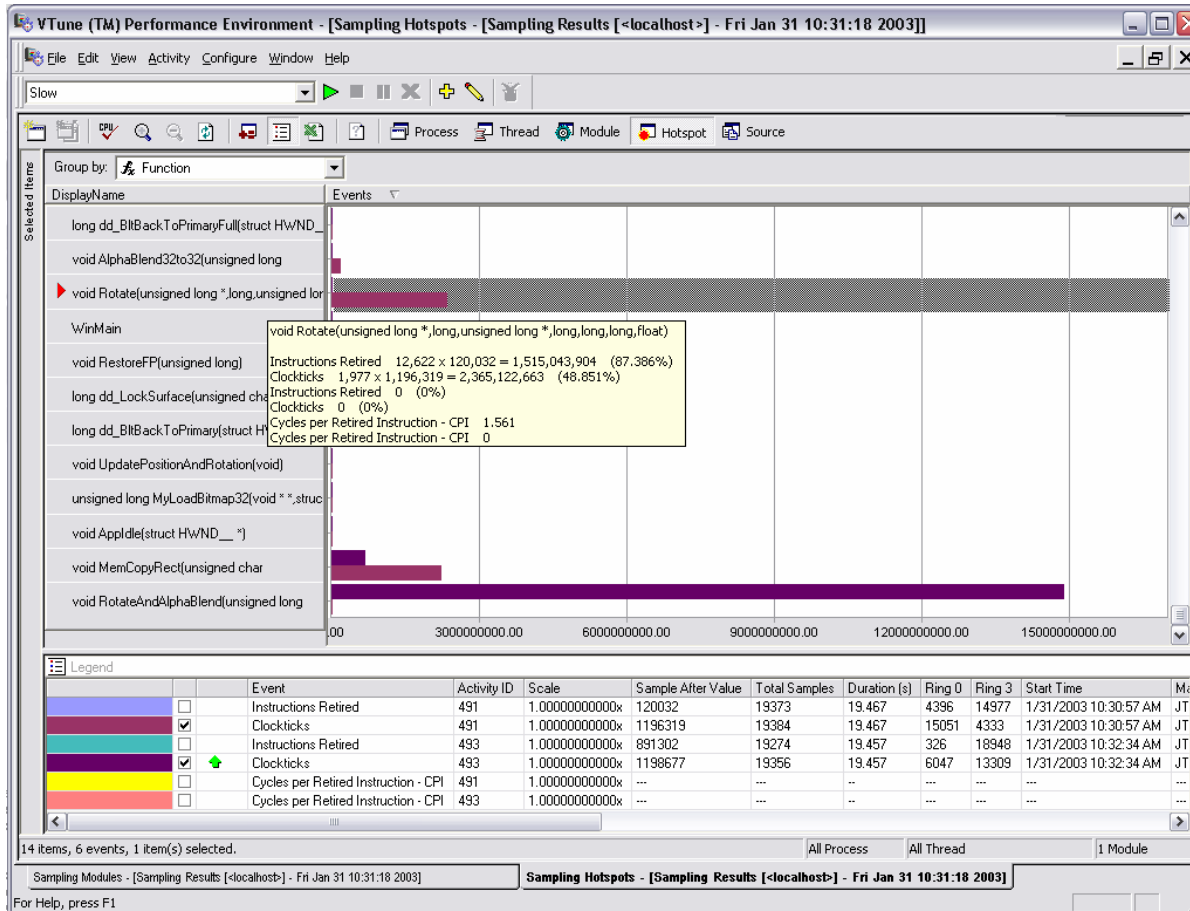
Deleting intermediate files and output files for project: RotateBlend - Win32 Release

Compiling... app.cpp bitmap.cpp dd.cpp RotateBlend.cpp StdAfx.cpp

Linking... xilink6: executing 'C:\PROGRA~1\MICROS~3\VC98\Bin\link.exe'

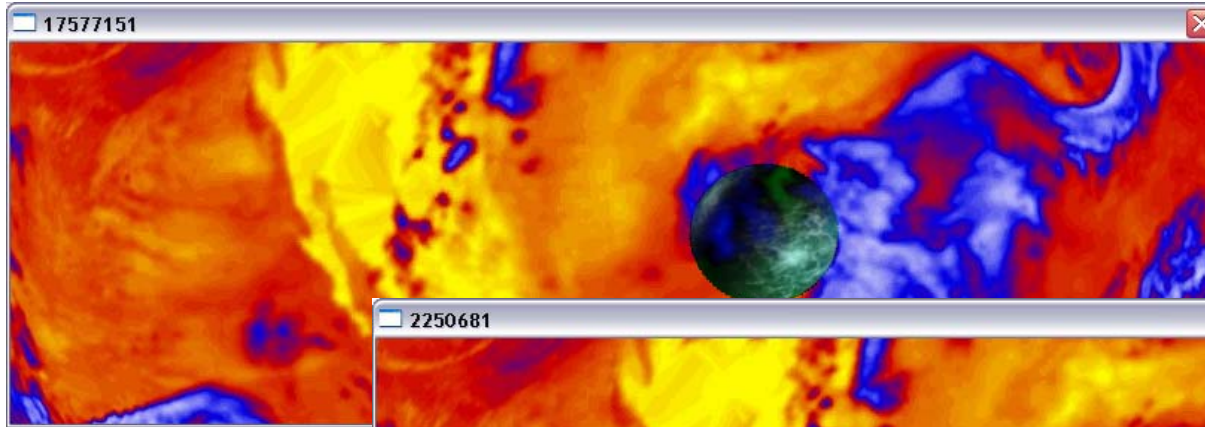
RotateBlend.exe - 0 error(s), 0 warning(s)

# Run VTune™ !

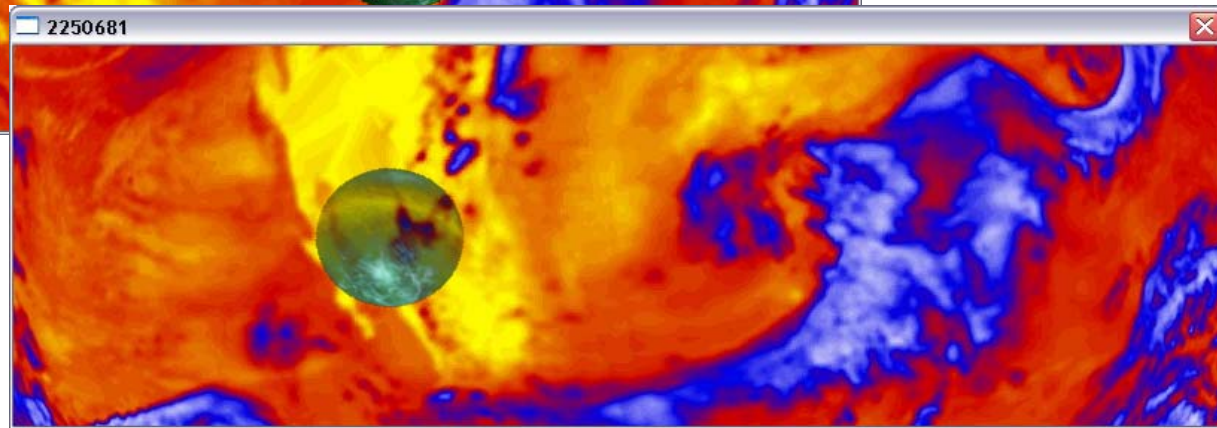


- Remove fp to int conversions
- Limit rectangle copy
- One call to sin & cos per frames
- Remove multiply
- Simplify bilinear interpolation
- Etc.

# Analyze, recode ... and optimize!



x 7.8

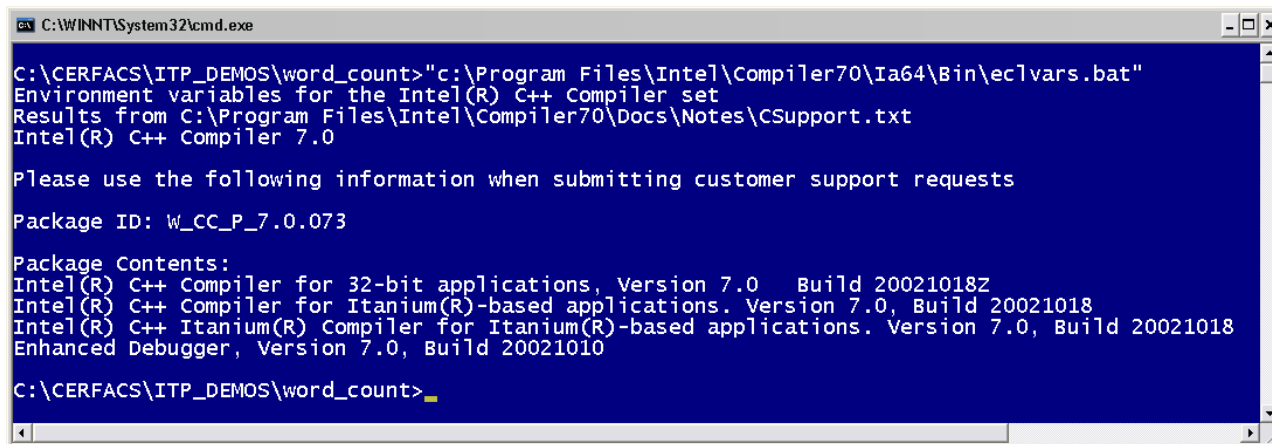


# Agenda

- VTune™ in action on NetBurst™ Architecture
- C/C++ compiler to target Itanium™ & Itanium® 2 processors

# First Example – Array\_xor 1/4

```
void array_xor(int *dest, int *src1, int *src2, int
    count) {
    while(count--) {
        *dest++ = *src1++ ^ *src2++;
    }
}
```



```
C:\CERFACS\ITP_DEMOS\word_count>"c:\Program Files\Intel\Compiler70\Ia64\Bin\ec1vars.bat"
Environment variables for the Intel(R) C++ Compiler set
Results from C:\Program Files\Intel\Compiler70\Docs\Notes\Csupport.txt
Intel(R) C++ Compiler 7.0

Please use the following information when submitting customer support requests

Package ID: W_CC_P_7.0.073

Package Contents:
Intel(R) C++ Compiler for 32-bit applications, Version 7.0   Build 20021018Z
Intel(R) C++ Compiler for Itanium(R)-based applications, Version 7.0, Build 20021018
Intel(R) C++ Itanium(R) Compiler for Itanium(R)-based applications, Version 7.0, Build 20021018
Enhanced Debugger, Version 7.0, Build 20021010

C:\CERFACS\ITP_DEMOS\word_count>_
```

```
> ecl /S /Faaliasing.asm aliasing.c
```

# First Example – Array\_xor 2/4

```
.b1_9:
{   .mmi
    ld4  r10=[r34],4
    ld4  r8=[r33],4
    nop.i    0 ;;    // stop
} {   .mmi
    xor  r3=r8,r10 ;;    // stop
    st4  [r32]=r3,4
    nop.i    0
} {   .mib
    nop.m    0
    nop.i    0
    br.cloop.sptk .b1_9 ;;    // stop
}
```

```
{   .mii
    nop.m    0
    mov     ar.lc=r14
    nop.i    0 ;;
}
```

# First Example – Array\_xor 3/4

```
void array_xor(int * restrict dest, int * restrict
    src1, int * restrict src2, int count) {
    while(count--) {
        *dest++ = *src1++ ^ *src2++;
    }
}
```

```
> ecl /S /Ow /Qrestrict /Fano_aliasing.asm no_aliasing.c
```

# First Example – Array\_xor 4/4

```
.b1_3:
{   .mmi
(p16) ld4      r32=[r8],4
(p16) ld4      r34=[r3],4
(p17) xor      r36=r33,r35 ;; // stop
} {   .mib
(p17) st4      [r2]=r36,4
      nop.i    0
      br.ctop.sptk .b1_3 ;; // stop
}
```

```
.b1_3:
{   .mii
(p16) ld4      r32=[r8],4
(p17) xor      r36=r33,r35
      nop.i    0
} {   .mmb
(p16) ld4      r34=[r3],4
(p18) st4      [r2]=r37,4
      br.ctop.sptk .b1_3 ;;
}
```

```
{   .mii
      nop.m    0
      mov      pr.rot=0x10000
      sxt4     r10=r11 ;;
} {   .mii
      nop.m    0
      mov      ar.ec=2
      sxt4     r9=r10 ;;
} {   .mii
      nop.m    0
      mov      ar.lc=r9
      nop.i    0 ;;
}
```

# Second Example – Word\_Count 1/16

```
#include <xmmintrin.h>
#define FALSE 0

int word_count_1(__m64 *buf64, int size) {
    int count, n;
    unsigned char c, *buf;
    int prev_was_letter, this_is_letter;

    buf                = (unsigned char *)buf64;
    prev_was_letter    = FALSE;
    count              = 0;

    for(n=0;n<size;n++) {
        c = buf[n];
        this_is_letter = ((c>='A')&&(c<='Z'))||((c>='a')&&(c<='z'));
        if(this_is_letter && !prev_was_letter) count++;
        prev_was_letter = this_is_letter;
    }
    return count;
}
```

```
typedef unsigned long long __m64;
```

**New Data Types Available**

New Data Type	MMX™ Technology	Streaming SIMD Extensions	Streaming SIMD Extensions 2	Itanium® Processor
__m64	X	X	X	X
__m128	N/A	X	X	X
__m128d	N/A	N/A	X	X
__m128i	N/A	N/A	X	X

# Second Example – Word\_Count 2/16

```

.b1_18:
{ .mii
    ld1        r3=[r32],1
    cmp.eq.unc p0,p12=r0,r0
    cmp.eq.unc p0,p10=r0,r0 ;;           // stop
} { .mmi
    cmp4.gt.unc p9,p8=65,r3 ;;         // stop
    (p8)      cmp4.ge    p12,p9=90,r3
    nop.i     0 ;;                     // stop
} { .mii
    (p9)      cmp4.gt.unc p0,p7=97,r3
    (p9)      cmp4.gt    p10,p0=97,r3
    nop.i     0 ;;                     // stop
} { .mmi
    (p7)      cmp4.lt    p10,p12=122,r3 ;; // stop
    (p12)     cmp4.ne.unc p0,p6=r2,r0
    (p12)     add        r2=1,r0 ;;     // stop
} { .mib
    (p10)     add        r2=0,r0
    (p6)      add        r8=1,r8
    br.cloop.sptk .b1_18 ;;           // stop
}

```

c <sub>type</sub>	pseudo-op of	PR[qp]=0		PR[qp]=1					
				result=0, No Source NaTs		result=1, No Source NaTs		One or More Source NaTs	
		PR[p <sub>1</sub> ]	PR[p <sub>2</sub> ]	PR[p <sub>1</sub> ]	PR[p <sub>2</sub> ]	PR[p <sub>1</sub> ]	PR[p <sub>2</sub> ]	PR[p <sub>1</sub> ]	PR[p <sub>2</sub> ]
none				0	1	1	0	0	0
unc		0	0	0	1	1	0	0	0
or				0	0	1	1	0	0
or.andcm						1	0		
orcm	or			1	1			0	0
andcm	and					0	0	0	0
and.orcm	or.andcm			0	1				

L1	L2	L3	Mem
4.01	4.18	4.42	7.53

# Second Example – Word\_Count 3/16

```

WinDiff
File Edit View Expand Options Mark Help
.\word_count_1.asm : .\word_count_1_03.asm C:\CERFACS\I\TP_DEMOS\word_count\word_count_1.asm : C:\CERFACS\I\TP_DEMOS\word_cou
Outline
<! // mark_description "Intel(R) C++ Compiler";
// mark_description "Version 7.0, Build 20021018 ";
<| // mark_description "-Qlocation,link,C:\Program Files\Microsoft SDK\bin\Win64 -S -Faward_count_1.asm";
!> // mark_description "-Qlocation,link,C:\Program Files\Microsoft SDK\bin\Win64 -S -03 -Faward_count_1_03.asm";
.ident "Intel(R) C++ Compiler"
<| .ident "-Qlocation,link,C:\Program Files\Microsoft SDK\bin\Win64 -S -Faward_count_1.asm"
!> .ident "-Qlocation,link,C:\Program Files\Microsoft SDK\bin\Win64 -S -03 -Faward_count_1_03.asm"
//.radix C
.file "word_count_1.c"
.section .text, "xa", "progbits"
.section .pdata, "a", "progbits"
.align 4
.section .xdata, "a", "progbits"
.align 8
.section .data, "wa", "progbits"
.align 16
.section .bss, "wa", "nobits"
.align 16
.section .rdata, "a", "progbits"
.align 16
.section .tls$, "was", "progbits"
.align 16
.section .data1, "wa", "progbits"
.align 16
.section .sdata, "was", "progbits"
.align 16

```

# Second Example – Word\_Count 4/16

```

#include <xmmintrin.h>
#define FALSE 0

int word_count_2(__m64 *buf64, int size) {
    int count, n;
    unsigned char c, *buf;
    int prev_was_letter, this_is_letter;

    buf                = (unsigned char *)buf64;
    prev_was_letter    = FALSE;
    count              = 0;

    for(n=0;n<size;n++) {
        c = buf[n];
        c &= 0xdf; // Map c to upper case by zeroing bit 5
        this_is_letter = (c>='A') && (c<='Z'); // Remove lower case test
        if(this_is_letter && !prev_was_letter) count++;
        prev_was_letter = this_is_letter;
    }
    return count;
}

```

	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
:	64	40	100	&#64;	@	96	60	140	&#96;	`
	65	41	101	&#65;	A	97	61	141	&#97;	a
	66	42	102	&#66;	B	98	62	142	&#98;	b
	67	43	103	&#67;	C	99	63	143	&#99;	c
	68	44	104	&#68;	D	100	64	144	&#100;	d
	69	45	105	&#69;	E	101	65	145	&#101;	e
	70	46	106	&#70;	F	102	66	146	&#102;	f
	71	47	107	&#71;	G	103	67	147	&#103;	g
	72	48	110	&#72;	H	104	68	150	&#104;	h

0	1	0	0	0	1	0	0
0	1	1	0	0	1	0	0

# Second Example – Word\_Count 5/16

```

.b1_2:
.pred.rel "mutex", p20, p21
{
  .mii
  (p16)    ld1      r36=[r3],1
  (p17)    cmp4.le.unc p22,p19=65,r33
  (p21)    cmp4.ne.unc p0,p23=r35,r0
} {
  .mmf
  (p20)    add      r34=0,r0
  (p21)    add      r34=1,r0
           nop.f    0 ;;          // stop
} {
  .mii
  (p22)    cmp4.lt   p19,p0=90,r33
  (p16)    and      r32=r36,r2
  (p22)    cmp4.lt.unc p0,p20=90,r33
} {
  .mfb
  (p23)    add      r8=1,r8
           nop.f    0
           br.ctop.sptk .b1_2 ;;  // stop
}

```

## Predicate Relationship Annotation

The predicate relationship annotation `.pred.rel` provides information for the assembler about a logical relationship between the values of predicate registers. It is relevant only for explicit code.

The annotation `.pred.rel` takes the following forms:

"mutex"	mutual exclusion
"imply"	implication
"clear"	clear existing relations

When conflicting instructions follow an entry point, IAS ignores all existing predicate relationships defined before the entry point.

L1	L2	L3	Mem
4.01	4.13	4.63	7.61

# Second Example – Word\_Count 6/16

```
#include <xmmintrin.h>
#define FALSE 0

int word_count_3(__m64 *buf64, int size) {
    int count, n;
    unsigned char c, *buf;
    int prev_was_letter, this_is_letter;

    buf                = (unsigned char *)buf64;
    prev_was_letter    = FALSE;
    count              = 0;

    for(n=0;n<size;n++) {
        c = buf[n];
        // Map c to upper case by zeroing bit 5
        // And replace a comparison by a subtraction
        c &= 0xdf;
        c -= 'A';
        this_is_letter = (c<26); // Modified case test
        if(this_is_letter && !prev_was_letter) count++;
        prev_was_letter = this_is_letter;
    }
    return count;
}
```

# Second Example – Word\_Count 7/16

```

.b1_2:
{ .mii
  (p16)    ld1        r35=[r3],1
  (p17)    add        r36=-65,r38
  (p18)    cmp4.gt.unc p20,p21=26,r39
} { .mmf
  (p23)    add        r8=1,r8
           nop.m      0
           nop.f      0 ;;          // stop
.pred.rel "mutex", p20, p21
} { .mii
  (p20)    cmp4.eq.unc p22,p0=r34,r0
  (p16)    and        r37=r35,r2
  (p17)    zxt1       r38=r36
} { .mmb
  (p20)    add        r33=1,r0
  (p21)    add        r33=0,r0
           br.ctop.sptk .b1_2 ;;    // stop
}

```

## Zero Extend

**Format:** (qp) zxt $_{xsz}$   $r_1 = r_3$

129

**Description:** The value in GR  $r_3$  is zero extended above the bit position specified by  $xsz$  and the result is placed in GR  $r_1$ . The mnemonic values for  $xsz$  are given in [Table 2-49 on page 2-218](#).

**Operation:** if (PR[qp]) {  
     check\_target\_register( $r_1$ );  
     GR[ $r_1$ ] = zero\_ext(GR[ $r_3$ ],  $xsz * 8$ );  
     GR[ $r_1$ ].nat = GR[ $r_3$ ].nat;  
 }

**Interruptions:** Illegal Operation fault

$xsz$ Mnemonic	Bit Position
1	7
2	15
4	31

L1	L2	L3	Mem
3.01	3.30	3.68	6.60

# Second Example – Word\_Count 8/16

```
#include <xmmintrin.h>
#define FALSE 0

int word_count_4(__m64 *buf64, int size) {
    int count, n;
    unsigned char c, *buf;
    int prev_was_letter, this_is_letter;

    buf                = (unsigned char *)buf64;
    prev_was_letter    = FALSE;
    count              = 0;

    for(n=0;n<size;n++) {
        c = buf[n];
        c &= 0xdf;
        c -= 'A';
        this_is_letter = (c<26);
        // Bit logic versus logic
        if((this_is_letter ^ prev_was_letter) & this_is_letter) count++;
        prev_was_letter = this_is_letter;
    }
    return count;
}
```

# Second Example – Word\_Count 9/16

```

.b1_2:
{ .mml
  (p16)    ld1        r39=[r9],1
  (p19)    xor        r40=r34,r35
  (p17)    add        r32=-65,r36
} { .mml
  (p20)    cmp4.ne.unc p22,p0=r38,r0
  (p23)    add        r8=1,r8
  (p18)    cmp4.gt.unc p24,p25=26,r37 ;;           // stop
.pred.rel "mutex", p24, p25
} { .mml
  (p24)    add        r33=1,r0
  (p16)    and        r35=r39,r2
  (p17)    zxt1       r36=r32
} { .mmb
  (p25)    add        r33=0,r0
  (p19)    and        r37=r40,r34
                br.ctop.sptk .b1_2 ;;           // stop
}

```

L1	L2	L3	Mem
4.01	4.12	4.44	7.67

# Second Example – Word\_Count 10/16

```
#include <xmmintrin.h>
#define FALSE 0

int word_count_5(__m64 *buf64, int size) {
    int count, n;
    unsigned char c, *buf;
    int prev_was_letter, this_is_letter;

    buf                = (unsigned char *)buf64;
    prev_was_letter    = FALSE;
    count              = 0;

    for(n=0;n<size;n++) {
        c = buf[n];
        c &= 0xdf;
        c -= 'A';
        this_is_letter = (c<26);
        // Bit logic versus logic
        // and remove the if
        count += ((this_is_letter ^ prev_was_letter) & this_is_letter);
        prev_was_letter = this_is_letter;
    }
    return count;
}
```

# Second Example – Word\_Count 11/16

```

.b1_2:
{
  .mii
  (p16)    ld1        r42=[r9],1
  (p17)    add        r41=-65,r36
  (p18)    cmp4.gt.unc p21,p22=26,r37
} {
  .mmf
  (p19)    xor        r40=r34,r35
  (p20)    add        r39=r32,r38
              nop.f    0 ;;                // stop
.pred.rel "mutex", p21, p22
} {
  .mii
  (p21)    add        r33=1,r0
  (p16)    and        r35=r42,r2
  (p17)    zxt1       r36=r41
} {
  .mmb
  (p22)    add        r33=0,r0
  (p19)    and        r37=r40,r34
              br.ctop.sptk .b1_2 ;;        // stop
}

```

L1	L2	L3	Mem
3.01	3.30	3.69	6.71

# Second Example – Word\_Count 12/16

```

#include <xmmintrin.h>
#define FALSE 0

int word_count_6(__m64 *buf64, int size) {
    int count, n;
    unsigned char c, *buf;
    int prev_was_letter, this_is_letter;

    buf                = (unsigned char *)buf64;
    prev_was_letter    = FALSE;
    count              = 0;

    for(n=0;n<size;n++) {
        c = buf[n];
        c &= 0xdf;
        c -= 'A';
        this_is_letter = (c<26);
        // Remove the & makes count all transitions, and makes count doubled at the end
        count += (this_is_letter ^ prev_was_letter);
        prev_was_letter = this_is_letter;
    }
    // Compute real count
    count = (count + 1) / 2;
    return count;
}

```

# Second Example – Word\_Count 13/16

```

.b1_2:
{
  .mii
  (p16)    ld1        r37=[r9],1
  (p17)    add        r38=-65,r8
  (p18)    cmp4.gt.unc p20,p21=26,r3
} {
  .mmf
  (p19)    xor        r39=r34,r35
            nop.m      0
            nop.f      0 ;;
            // stop
.pred.rel "mutex", p20, p21
} {
  .mii
  (p20)    add        r33=1,r0
  (p16)    and        r8=r37,r2
  (p17)    zxt1       r3=r38
} {
  .mmb
  (p21)    add        r33=0,r0
  (p19)    add        r35=r36,r39
            br.ctop.sptk .b1_2 ;;
            // stop
}

```

```

.b1_2:
{
  .mii
  (p16)    ld1        r37=[r9],1
  (p17)    add        r38=-65,r8
  (p18)    cmp4.gt.unc p20,p21=26,r3
} {
  .mmf
  (p19)    xor        r39=r34,r35
            nop.m      0
            nop.f      0 ;;
.pred.rel "mutex", p20, p21
} {
  .mii
  (p20)    add        r33=1,r0
  (p16)    and        r8=r37,r2
  (p17)    zxt1       r3=r38
} {
  .mmb
  (p21)    add        r33=0,r0
  (p19)    add        r35=r36,r39
            br.ctop.sptk .b1_2 ;;
}

```

L1	L2	L3	Mem
3.01	3.30	3.66	6.55

# Second Example – Word\_Count 14/16

```
#include <xmmintrin.h>
#include <ia64intrin.h>

int word_count_7(__m64 *buf64, int size) {
    int count = 0;
    __m64 c8, temp, p8 = 0;
    size /= 8;

    while(size--> 0) {
        c8 = *buf64++;
        c8 = _mm_and_si64(c8, 0xdfdfdfdfdfdfdfdf);
        c8 = _mm_add_pi8(c8, 0xc1c1c1c1c1c1c1c1);
        c8 = _mm_cmplt_pi8(c8, 0x1a1a1a1a1a1a1a1a);
        p8 = _m_from_int64(_m64_shrp(_m_to_int64(c8), _m_to_int64(p8), 56));
        temp = _mm_xor_si64(c8, p8);
        temp = _m_from_int64(_m64_popcnt(_m_to_int64(temp)));
        count += _m_to_int64(temp);
        p8 = c8;
    }
    count = (count + 8) / 16;
    return count;
}
```

## Conversion Intrinsics

The prototypes for these intrinsics are in the `ia64intrin.h` header file.

Intrinsic	Description
<code>__int64 _m_to_int64(__m64 a)</code>	Convert a of type <code>__m64</code> to type <code>__int64</code> . Translates to <code>nop</code> since both types reside in the same register on Itanium-based systems.
<code>__m64 _m_from_int64(__int64 a)</code>	Convert a of type <code>__int64</code> to type <code>__m64</code> . Translates to <code>nop</code> since both types reside in the same register on Itanium-based systems.
<code>__int64 __round_double_to_int64(double d)</code>	Convert its double precision argument to a signed integer.
<code>unsigned __int64 __getf_exp(double d)</code>	Map the <code>getf.exp</code> instruction and return the 16-bit exponent and the sign of its operand.

# Second Example – Word\_Count 15/16

```
.b1_3:
{ .mmi
(p20) mov r54=r35
(p16) ld8 r53=[r9],8
(p23) popcnt r40=r38
} { .mmi
(p19) mov r44=r35
(p21) xor r55=r36,r52
(p25) add r50=r51,r42 ;; // stop
} { .mii
(p16) and r38=r53,r8
(p20) shrp r51=r54,r45,56
(p21) mov r36=r55
} { .mmb
(p17) paddl r42=r39,r3
(p18) pcmp1.lt r33=r43,r2
br.ctop.sptk .b1_3 ;; // stop
}
```

### Population Count

**Format:** (qp) popcnt  $r_1=r_3$  19

**Description:** The number of bits in GR  $r_3$  having the value 1 is counted, and the resulting sum is placed in GR  $r_1$ .

**Operation:**

```
if (PR[qp]) {
  check_target_register(r1);

  res = 0;
  // Count up all the one bits
  for (i = 0; i < 64; i++) {
    res += GR[r3][i];
  }

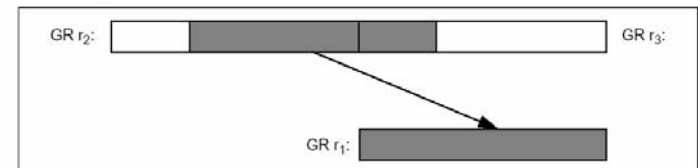
  GR[r1] = res;
  GR[r2].nat = GR[r3].nat;
}
```

### Shift Right Pair

**Format:** (qp) shrp  $r_1=r_2, r_3, count_6$  110

**Description:** The two source operands, GR  $r_2$  and GR  $r_3$ , are concatenated to form a 128-bit value and shifted to the right  $count_6$  bits. The least-significant 64 bits of the result are placed in GR  $r_1$ . The immediate value  $count_6$  can be any number in the range 0 to 63.

Figure 2-43. Shift Right Pair

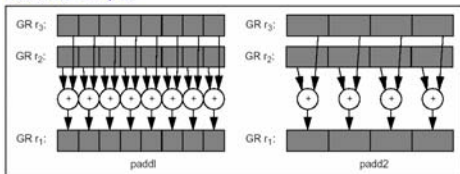


**Operation:**

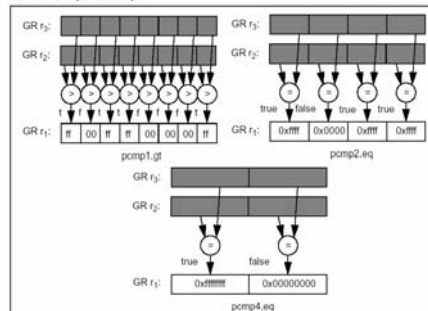
```
if (PR[qp]) {
  check_target_register(r1);

  temp1 = shift_right_unsigned(GR[r3], count6);
  temp2 = GR[r2] << (64 - count6);
  GR[r1] = zero_ext(temp1, 64 - count6) | temp2;
  GR[r2].nat = GR[r2].nat || GR[r3].nat;
}
```

Parallel Add Examples



Parallel Compare Example



L1	L2	L3	Mem
0.39	0.70	0.89	3.65



## Second Example – Word\_Count 16/16

function	L1	L2	L3	Memory
<i>word_count_1</i>	4.01	4.18	4.42	7.53
<i>word_count_2</i>	4.01	4.13	4.63	7.61
<i>word_count_3</i>	3.01	3.30	3.68	6.60
<i>word_count_4</i>	4.01	4.12	4.44	7.67
<i>word_count_5</i>	3.01	3.30	3.69	6.71
<i>word_count_6</i>	3.01	3.30	3.66	6.55
<i>word_count_7</i>	0.39	0.70	0.89	3.65

cycles per character

# Summary – Help the compiler!

- **Optimization starts with your algorithm!**
- **Use common sense**
  - Lookup tables, approximation, etc.
- **Understand and avoid aliasing**
  - But also pointer chasing, recursion, etc.
  - Use restrict keyword (/Qrestrict)
- **Use (and abuse of) IPO and PGO**
  - IPO: Interprocedural Optimization
    - /Qipo, /Qwp\_ipo
  - PGO: Profile-Guided Optimization
    - /Oprof\_gen, /Qprof\_use (\*.dyn, \*.dpi)
- **Use compiler switches**
  - /O2, /O3, etc. (make experiments, then VTune, ASM)
- **Use Intrinsic**
  - i.e. `_mm_cmpeq_pi8()` -> make use of `pcmp1.eq`, `__m64`, etc.

# Summary – Use the Compiler

- **Push the optimizations to their highest level on modules giving largest contribution to CPI**
  - O3, HLO will invoke data prefetching, most aggressive loop unrolling
  - Profile guided feedback + Inter procedural Inlining (pgo+ipo) work well **together**
- **Use compiler reports to gain insights into automated optimizations**
  - `/Qopt_report_phasephase, /Qopt_report_filefilename`
  - *Phase* = ipo, hlo, ecg (SW Pipelining), ilo, omp, or all
  - Frequently doing some of the automated optimization manually and then recompiling results in even better code

# Thank You.

**developer.intel.com**

**<http://www.intel.com/intelpress/>**

## Technology Leadership



**intel.**

## Packed Arithmetic Intrinsics

The prototypes for MMX™ technology intrinsics are in the `mmintrin.h` header file.

Intrinsic Name	Alternate Name	Corresponding Instruction	Operation	Signed	Argument Values/Bits	Result Values/Bits
<code>_m_paddb</code>	<code>_mm_add_pi8</code>	PADDB	Addition	--	8/8	8/8

```
__m64 _m_paddb(__m64 m1, __m64 m2)
```

Add the eight 8-bit values in `m1` to the eight 8-bit values in `m2`.

## Logical Intrinsics

The prototypes for MMX™ technology intrinsics are in the `mmintrin.h` header file.

Intrinsic Name	Alternate Name	Operation	Corresponding Instruction
<code>_m_pxor</code>	<code>_mm_xor_si64</code>	Bitwise Exclusive OR	PXOR

```
__m64 _m_pxor(__m64 m1, __m64 m2)
```

Perform a bitwise XOR of the 64-bit value in `m1` with the 64-bit value in `m2`.

## MMX™ Technology Logical Intrinsics

The prototypes for MMX™ technology intrinsics are in the `mmintrin.h` header file.

Intrinsic Name	Alternate Name	Operation	Corresponding Instruction
<code>_m_pand</code>	<code>_mm_and_si64</code>	Bitwise AND	PAND
<code>_m_pandn</code>	<code>_mm_andnot_si64</code>	Logical NOT	PANDN
<code>_m_por</code>	<code>_mm_or_si64</code>	Bitwise OR	POR
<code>_m_pxor</code>	<code>_mm_xor_si64</code>	Bitwise Exclusive OR	PXOR

```
__m64 _m_pand(__m64 m1, __m64 m2)
```

Perform a bitwise AND of the 64-bit value in `m1` with the 64-bit value in `m2`.

```
__m64 _m_pandn(__m64 m1, __m64 m2)
```

Perform a logical NOT on the 64-bit value in `m1` and use the result in a bitwise AND with the 64-bit value in `m2`.

```
__m64 _m_por(__m64 m1, __m64 m2)
```

Perform a bitwise OR of the 64-bit value in `m1` with the 64-bit value in `m2`.

```
__m64 _m_pxor(__m64 m1, __m64 m2)
```

Perform a bitwise XOR of the 64-bit value in `m1` with the 64-bit value in `m2`.

## Native Intrinsics for Itanium® Instructions

The prototypes for these intrinsics are in the `ia64intrin.h` header file.

### Integer Operations

Intrinsic	Corresponding Instruction
<code>__int64 _m64_popcnt(__int64 a)</code>	<code>popcnt</code> (Population count)

```
__int64 _m64_popcnt(__int64 a)
```

The number of bits in the 64-bit integer `a` that have the value 1 are counted, and the resulting sum is returned.