Core Performance

Jay Foad, Roger Hui

Sicily 2015
Performance Comparison
Between Windows-64 14.1.24671.0 and Windows-64 14.0.21929.0

Relative Performance of 136 Expression Groups

86.8% - with 85/136 more than 2% better  13.2% - with 1/136 more than 2% worse

30 Best Groups

<table>
<thead>
<tr>
<th>Group</th>
<th>#</th>
<th>mean</th>
<th>min</th>
<th>25%</th>
<th>median</th>
<th>75%</th>
<th>max</th>
<th>stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsp0/sysp0</td>
<td>68</td>
<td>-29.3%</td>
<td>-36.7%</td>
<td>-34.5%</td>
<td>-30.8%</td>
<td>-23.6%</td>
<td>-10.7%</td>
<td>0.0621</td>
</tr>
<tr>
<td>uya</td>
<td>28</td>
<td>-27.7%</td>
<td>-85.9%</td>
<td>-80.2%</td>
<td>-0.3%</td>
<td>+0.4%</td>
<td>+6.4%</td>
<td>0.3928</td>
</tr>
<tr>
<td>bv1 penc0 av1</td>
<td>72</td>
<td>-23.6%</td>
<td>-55.1%</td>
<td>-44.2%</td>
<td>-20.0%</td>
<td>-2.4%</td>
<td>+1.0%</td>
<td>0.1928</td>
</tr>
<tr>
<td>r av0</td>
<td>128</td>
<td>-19.3%</td>
<td>-33.2%</td>
<td>-19.5%</td>
<td>-19.2%</td>
<td>-19.1%</td>
<td>-0.2%</td>
<td>0.0638</td>
</tr>
<tr>
<td>br01[qy1]r0</td>
<td>16</td>
<td>-13.0%</td>
<td>-13.2%</td>
<td>-13.2%</td>
<td>-13.1%</td>
<td>-13.0%</td>
<td>-12.1%</td>
<td>0.0031</td>
</tr>
</tbody>
</table>
Upgrading C compilers

• Why do it?
• Why is it difficult?
AIX: XL C/C++

- Dyalog up to 14.0 used XL C/C++ 12.1 (May 2012)
- Dyalog 14.1 onwards uses XL C/C++ 13.1 (June 2014)
- Newer compiler supports POWER8 hardware
Linux: GCC

• Dyalog up to 14.0 uses GCC 4.3.3 (January 2009)
• Dyalog 14.1 onwards uses GCC 4.9.2 (October 2014)
• Newer compiler has much better support for SIMD instruction sets (both auto and manual vectorisation)
Performance Comparison
Between Linux-64 15.0.24741.0 S Development gcc4.9.2 and gcc4.3.3

Relative Performance of 136 Expression Groups

72.8% - with 60/136 more than 2% better
27.2% - with 9/136 more than 2% worse

30 Best Groups

<table>
<thead>
<tr>
<th>Group</th>
<th>#</th>
<th>mean</th>
<th>min</th>
<th>25%</th>
<th>median</th>
<th>75%</th>
<th>max</th>
<th>stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>xbql+.xybql</td>
<td>42</td>
<td>-24.4%</td>
<td>-84.5%</td>
<td>-47.5%</td>
<td>-12.8%</td>
<td>+0.0%</td>
<td>+19.2%</td>
<td>0.3039</td>
</tr>
<tr>
<td>at1{(i@)(i</td>
<td>o</td>
<td>)}at1</td>
<td>9</td>
<td>-21.8%</td>
<td>-39.8%</td>
<td>-25.3%</td>
<td>-20.6%</td>
<td>-14.9%</td>
</tr>
<tr>
<td>&gt;sn0</td>
<td>20</td>
<td>-19.1%</td>
<td>-65.5%</td>
<td>-45.8%</td>
<td>-8.4%</td>
<td>+0.0%</td>
<td>+0.2%</td>
<td>0.2528</td>
</tr>
<tr>
<td>E1e4</td>
<td>3</td>
<td>14.8%</td>
<td>24.3%</td>
<td>24.3%</td>
<td>16.0%</td>
<td>3.4%</td>
<td>3.4%</td>
<td>0.1053</td>
</tr>
</tbody>
</table>
Best Expressions

Best: ylt2+.xlw2 (-84.5%), yit2+.xiw2 (-83.2%), xiq2+.yiq2 (-75.5%), xiw4+.yit4 (-71.9%), xiw2+.yit2 (-69.9%), xlw4+.xylt4 (-61.7%), xlw2+.xylt2 (-59.3%), xiw1+.xylt1 (-57.2%), yit1+.xwiw1 (-55.2%), xzw4+.yxzt4 (-48.4%), xzw2+.yxzt2 (-47.5%), yzt2+.xxzw2 (-46.8%)

Worst: xsw4+.yxst4 (+19.2%), xsw2+.yxst2 (+18.0%), yst1+.xsw1 (+11.5%), xsw1+.yxst1 (+10.9%), xsq2+.yxsq2 (+10.7%), xsq1+.yxsq1 (+6.3%), xdq1+.yxdq1 (+5.7%), xbw4+.ybt4 (+3.0%), xbw2+.ybt2 (+1.9%), xbj2+.ybj2 (+0.8%)
+13.2% : (xb0 ∨ yb0) [#1 of 30 Worst Groups]

Best: xl4 ∨ yl4 (-32.0%), xl4 ∧ yl4 (-30.1%), xi4 ∨ yi4 (-29.2%), xi4 ∧ yi4 (-24.6%), xs4 ∨ ys4 (-22.1%), xs4 ∧ ys4 (-16.4%), xl4 ∧ yi4 (-13.4%), xi4 ∧ yi4 (-13.3%), xl4 ∨ yi4 (-11.3%), xi4 ∨ yi4 (-10.1%), xi4 ∧ ys4 (-5.0%), xs4 ∧ yi4 (-4.3%)

Worst: xl2 ∨ yl2 (+77.3%), xl2 ∧ yl2 (+76.5%), xs2 ∨ yl2 (+74.4%), xl2 ∨ ys2 (+73.2%), xi2 ∨ yi2 (+70.0%), xl2 ∧ yl2 (+68.8%), xi2 ∧ ys2 (+65.4%), xs2 ∧ yi2 (+61.9%), xl2 ∧ yi2 (+47.5%), xi2 ∧ yi2 (+46.4%), xi2 ∧ yl2 (+45.8%), xs2 ∧ yi2 (+45.2%)
Windows: Visual Studio

- Dyalog up to 14.1 uses VS 2005 (October 2005)
- Dyalog 15.0 onwards will use VS 2015 (July 2015)
First steps with VS 2015
Relative Performance of 136 Expression Groups

30 Best Groups

<table>
<thead>
<tr>
<th>Group</th>
<th>#</th>
<th>mean</th>
<th>min</th>
<th>25%</th>
<th>median</th>
<th>75%</th>
<th>max</th>
<th>stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edq1</td>
<td>3</td>
<td>-35.8%</td>
<td>-61.8%</td>
<td>-61.8%</td>
<td>-44.4%</td>
<td>-1.2%</td>
<td>-1.2%</td>
<td>0.3120</td>
</tr>
<tr>
<td>xsp0@ysp0</td>
<td>68</td>
<td>-31.1%</td>
<td>-48.4%</td>
<td>-44.0%</td>
<td>-33.3%</td>
<td>-10.8%</td>
<td>-2.9%</td>
<td>0.1510</td>
</tr>
<tr>
<td>dv2@dht2</td>
<td>2</td>
<td>-27.2%</td>
<td>-29.4%</td>
<td>-29.4%</td>
<td>-27.2%</td>
<td>-25.0%</td>
<td>-25.0%</td>
<td>0.0308</td>
</tr>
<tr>
<td>c1t1</td>
<td>34</td>
<td>-7.0%</td>
<td>-21.6%</td>
<td>-14.3%</td>
<td>-8.0%</td>
<td>-4.7%</td>
<td>-6.0%</td>
<td>0.0838</td>
</tr>
</tbody>
</table>
First steps with VS 2015

• The interpreter relies heavily on `setjmp()` and `longjmp()`
• VS 2015 made (undocumented) changes to work better with C++
• Solution: re-implement them in assembler
Performance Comparison Between Windows-64 15.0.25135.0 W Development vs2015.pqa and vs2005.pqa

Relative Performance of 136 Expression Groups

30 Best Groups

<table>
<thead>
<tr>
<th>Group</th>
<th>#</th>
<th>mean</th>
<th>min</th>
<th>25%</th>
<th>median</th>
<th>75%</th>
<th>max</th>
<th>stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>xbr1+.xybr1</td>
<td>42</td>
<td>-38.8%</td>
<td>-75.8%</td>
<td>-62.8%</td>
<td>-54.3%</td>
<td>-6.6%</td>
<td>+4.9%</td>
<td>0.2933</td>
</tr>
<tr>
<td>ebdq1</td>
<td>3</td>
<td>-38.5%</td>
<td>-61.8%</td>
<td>-61.8%</td>
<td>-47.4%</td>
<td>-6.1%</td>
<td>-6.1%</td>
<td>0.2891</td>
</tr>
<tr>
<td>xsp0+ysp0</td>
<td>68</td>
<td>-37.4%</td>
<td>-49.4%</td>
<td>-46.8%</td>
<td>-39.1%</td>
<td>-25.4%</td>
<td>-12.0%</td>
<td>0.1055</td>
</tr>
<tr>
<td>dbr1+dbr1</td>
<td>2</td>
<td>30.0%</td>
<td>23.3%</td>
<td>33.3%</td>
<td>33.3%</td>
<td>30.0%</td>
<td>37.8%</td>
<td>0.0314</td>
</tr>
</tbody>
</table>
**+26.4% : (*b0) [#1 of 30 Worst Groups]**

**Best:** *z2 (-36.9%), *z4 (-34.3%), *z1 (-34.2%), *z0 (-15.0%)

**Worst:** *b2 (+71.4%), *b4 (+66.0%), *s4 (+55.5%), *s2 (+53.1%), *d2 (+53.1%), *d4 (+52.4%), *s1 (+49.1%), *d1 (+41.9%)

**Best Expressions**

- **193.0 / 121.7 = -36.9%**
- **1898.7 / 1247.9 = -34.3%**
- **142.5 / 93.8 = -34.2%**

**Worst Expressions**
VS 2015

- Supports .NET Framework version 4 by default
- Makes it hard to support Windows XP (unsupported since April 2014)
- So Dyalog 15.0 will use .NET 4 and won't support XP
Vector instructions (Intel)

- MMX (1997)
- SSE (1999)
- SSE2 (2001)
- SSE3 (2004)
- SSSE3 (2006)
- SSE4.1
- SSE4.2
- AVX (2008)
- BMI (2012)
- BMI2 (2013)
- AVX2
- AVX-512 (2015)
- AVX-1024
Auto-vectorisation

- C function pdtFFF (+ . x)
- Inner loop in C:

```c
for(k=0; k<p; ++k)
    *zp++ += x * *rargp++;
```
Auto-vectorisation

- C function pdtFFF (+ . ×)
- Inner loop in assembler:

```assembly
0000000013FD0B860 movupd xmm1, xmmword ptr [rdx]
0000000013FD0B864 movupd xmm0, xmmword ptr [rcx]
0000000013FD0B868 add r8, 2
0000000013FD0B86C add rcx, 10h
0000000013FD0B870 add rdx, 10h
0000000013FD0B874 mulpd xmm1, xmm3
0000000013FD0B878 addpd xmm1, xmm0
0000000013FD0B87C movupd xmmword ptr [rcx-10h], xmm1
0000000013FD0B881 cmp r8, rbx
0000000013FD0B884 jb pdtFFF+0D0h (013FD0B860h)
```
Auto-vectorisation

- C function pdtFFF (\(+ \cdot \times\))
- Inner loop in assembler:

```assembly
000000013FD0B860  movupd   xmm1,xmmword ptr [rdx]
000000013FD0B864  movupd   xmm0,xmmword ptr [rcx]
000000013FD0B868  add       r8,2
000000013FD0B86C  add       rcx,10h
000000013FD0B870  add       rdx,10h
000000013FD0B874  mulpd    xmm1,xmm3
000000013FD0B878  addpd    xmm1,xmm0
000000013FD0B87C  movupd  xmmword ptr [rcx-10h],xmm1
000000013FD0B881  cmp       r8,rbx
000000013FD0B884  jb        pdtFFF+0D0h (013FD0B860h)
```
Manual vectorisation
The Intel Intrinsics Guide is an interactive reference tool for Intel intrinsic instructions, which are C style functions that provide access to many Intel instructions - including Intel® SSE, AVX, AVX-512, and more - without the need to write assembly code.

### _m128d _mm_add_pd (_m128d a, _m128d b)_

**Synopsis**

```c
__m128d _mm_add_pd (__m128d a, __m128d b)
#include "emmintrin.h"
Instruction: addpd xmm, xmm
CPUID Flags: SSE2
```

**Description**

Add packed double-precision (64-bit) floating-point elements in `a` and `b`, and store the results in `dst`.

**Operation**

```c
FOR j := 0 to 1
    i := j*64
ENDFOR
```

**Performance**

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Latency</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Haswell</td>
<td>3</td>
<td>0.8</td>
</tr>
<tr>
<td>Ivy Bridge</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Sandy Bridge</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Westmere</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Nehalem</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

### _m128d _mm_mask_add_pd (_m128d src, _mmask8 k, _m128d a, _m128d b)_

### _m128d _mm_maskz_add_pd (_mmask8 k, _m128d a, _m128d b)_
Manual vectorisation

• Reductions: \( \lceil / \rceil / / + / \)

• Intel & AMD:
  SSE2, SSE4.1
  AVX, AVX2 (to come)

• Factor of 1 to 30 for \( \lceil / \rceil / \)
• Factor of 2 to 5 for \(+ /\)
Manual vectorisation

- Reductions: $\gamma / \| / + /

- POWER architecture: AltiVec, VSX

- Factor of 2 to 20 for $\gamma / \| /$
- Factor of 2 to 4 for $+/+$
New instructions

• Boolean transpose (⟂) on square matrices (BMI2, POWER8)
• Factor of 10
New instructions

- Boolean compress (BMI2)
- Squeezing to boolean, blowing up from boolean (BMI2, POWER8)
- Factor of 2