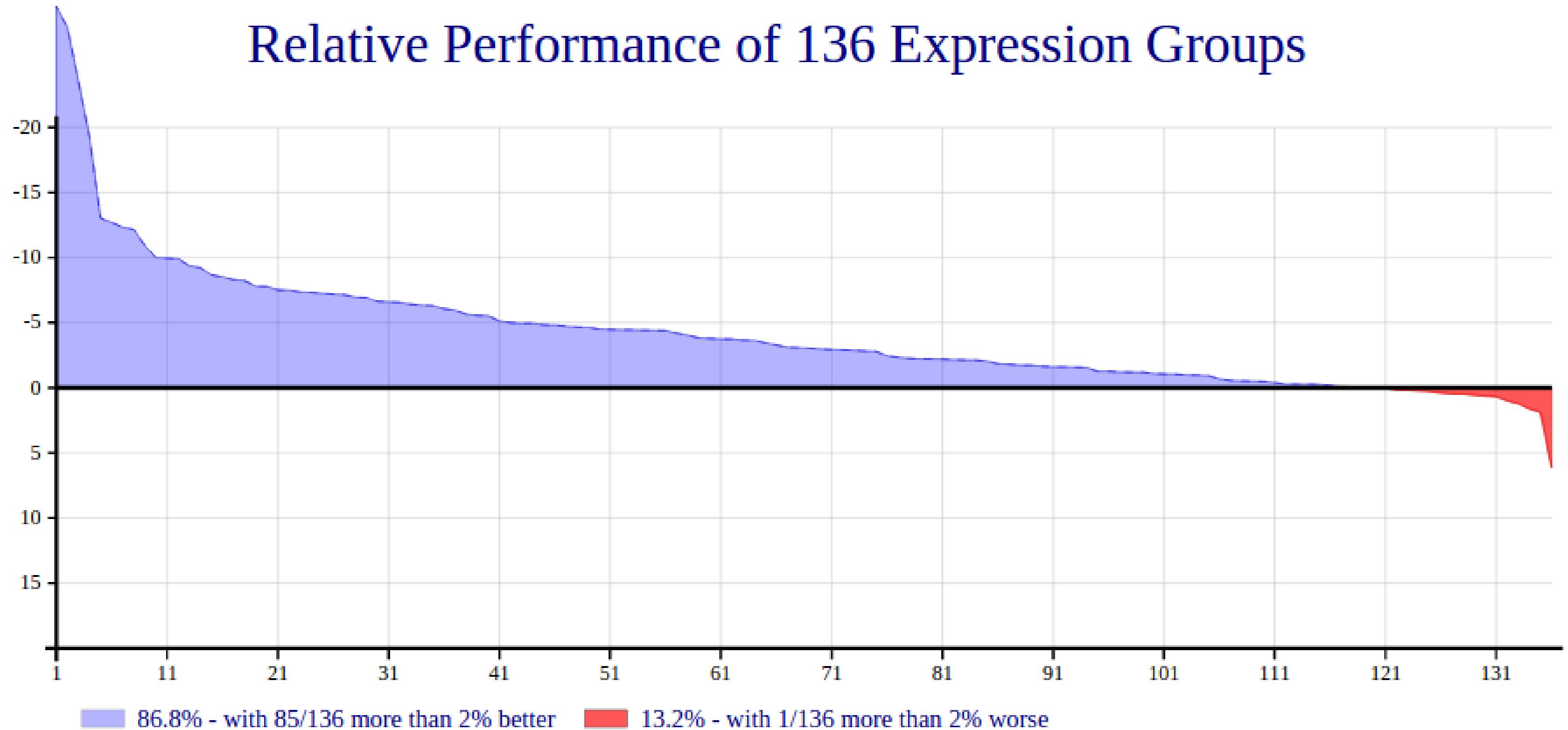
Core Performance

Jay Foad, Roger Hui

DYALOG

Sicily 2015

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



30 Best Groups

Group	#	mean	min	25%	median	75%	max	stddev
1. xsp0⊙ysp0	68	-29.3%	-36.7%	-34.5%	-30.8%	-23.6%	-10.7%	0.0621
2. ∪ya0	28	-27.7%	-85.9%	-80.2%	-0.3%	+0.4%	+6.4%	0.3928
3. bv1 penc0 av1	72	-23.6%	-55.1%	-44.2%	-20.0%	-2.4%	+1.0%	0.1928
4. ┬av0	128	-19.3%	-33.2%	-19.5%	-19.2%	-19.1%	-0.2%	0.0638
5. bt0{σω}bt0	16	-13.0%	-13.2%	-13.2%	-13.1%	-13.0%	-12.1%	0.0031

Upgrading C compilers

- Why do it?
- Why is it difficult?

AIX: XL C/C++

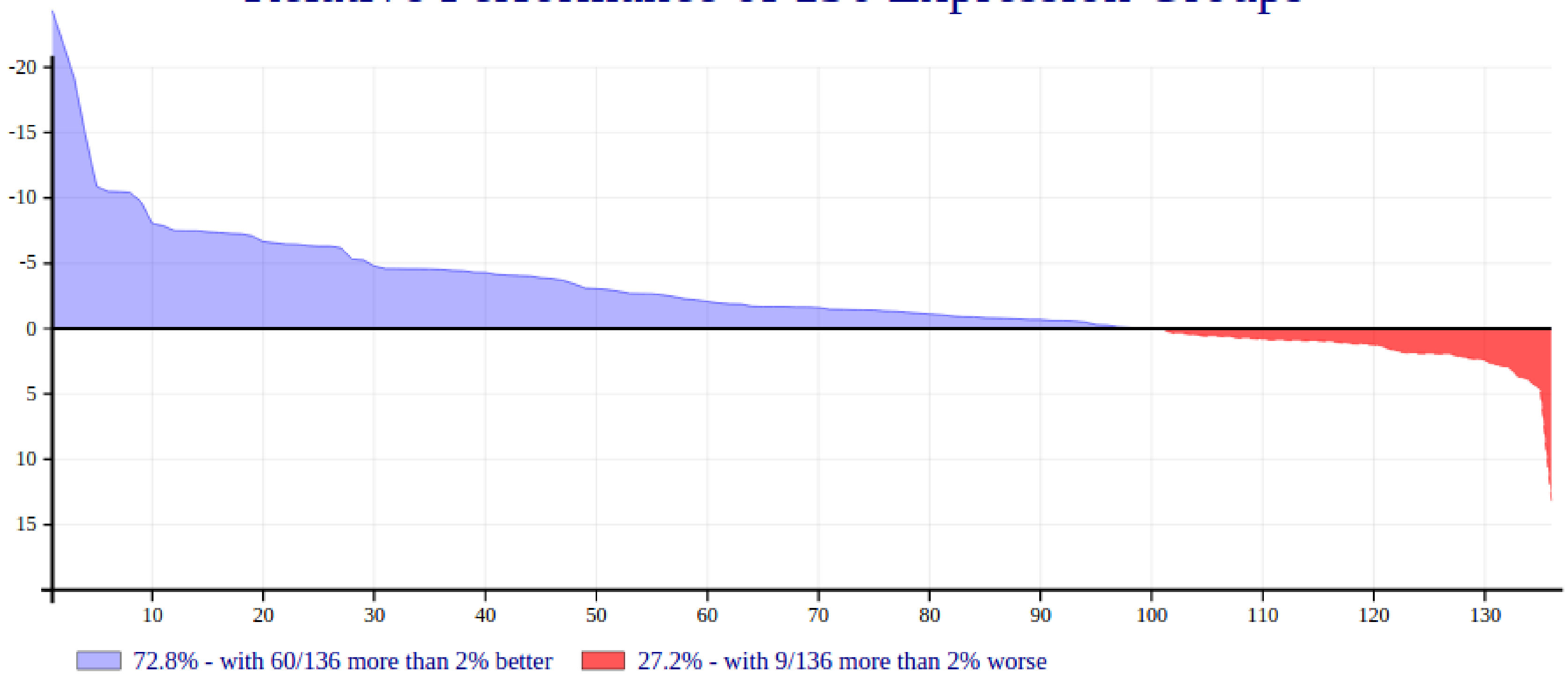
- Dyalog up to 14.0 used XL C/C++ 12.1 (May 2012)
- Dyalog 14.1 onwards uses XL C/C++ 13.1 (June 2014)
- Newer compiler supports POWER8 hardware

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

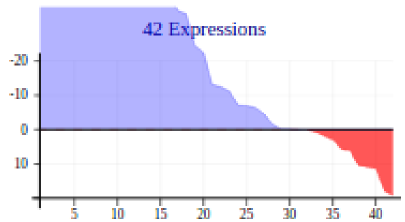


---

30 Best Groups

Group	#	mean	min	25%	median	75%	max	stddev
1. xbq1+.×ybq1	42	-24.4%	-84.5%	-47.5%	-12.8%	+0.0%	+19.2%	0.3039
2. at1{(↓α)ι↓ω}at1	9	-21.8%	-39.8%	-25.3%	-20.6%	-14.9%	-13.0%	0.0824
3. ÷sn0	20	-19.1%	-65.5%	-45.8%	-8.4%	+0.0%	+0.2%	0.2528
4. ⌹dq1	3	14.8%	24.2%	24.2%	16.0%	3.4%	3.4%	0.1053

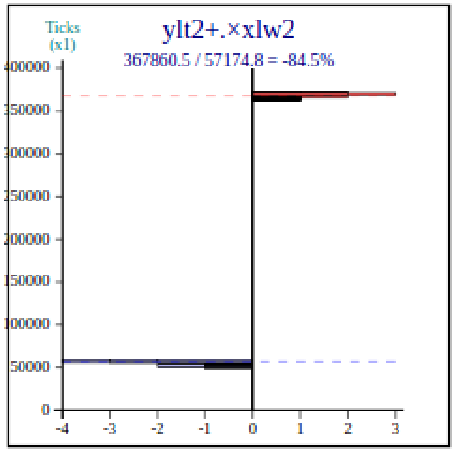
-24.4% : (xbq1+.×ybq1) [#1 of 30 Best Groups]

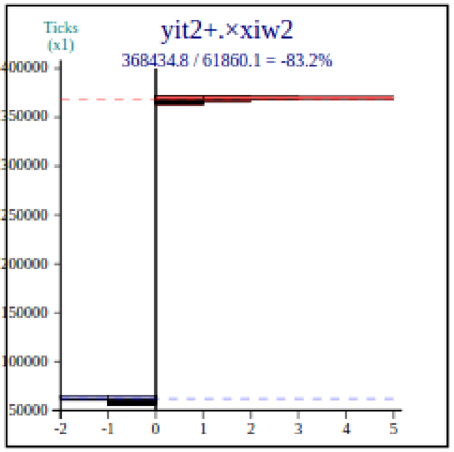


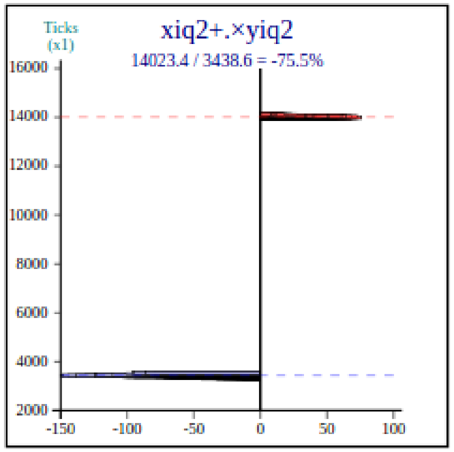
Best: ylt2+.×xlw2 (-84.5%), yit2+.×xiw2 (-83.2%), xiq2+.×yiq2 (-75.5%), xiw4+.×yit4 (-71.9%), xiw2+.×yit2 (-69.9%), xlw4+.×ylt4 (-61.7%), xlw2+.×ylt2 (-59.3%), xiw1+.×yit1 (-57.2%), yit1+.×xiw1 (-55.2%), xzw4+.×yzt4 (-48.4%), xzw2+.×yzt2 (-47.5%), yzt2+.×xzw2 (-46.8%)

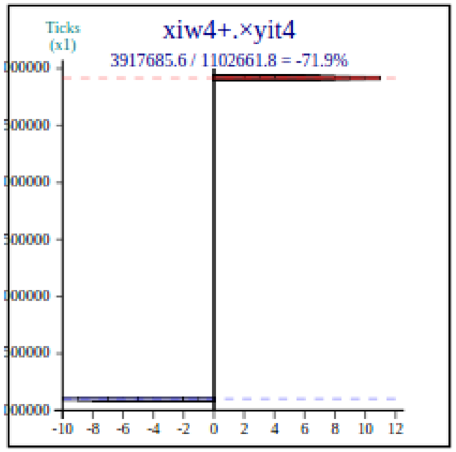
Worst: xsw4+.×yst4 (+19.2%), xsw2+.×yst2 (+18.0%), yst1+.×xsw1 (+11.5%), xsw1+.×yst1 (+10.9%), xsq2+.×ysq2 (+10.7%), xsq1+.×ysq1 (+6.3%), xdq1+.×ydq1 (+5.7%), xbw4+.×ybt4 (+3.0%), xbw2+.×ybt2 (+1.9%), xbq2+.×ybq2 (+0.8%)

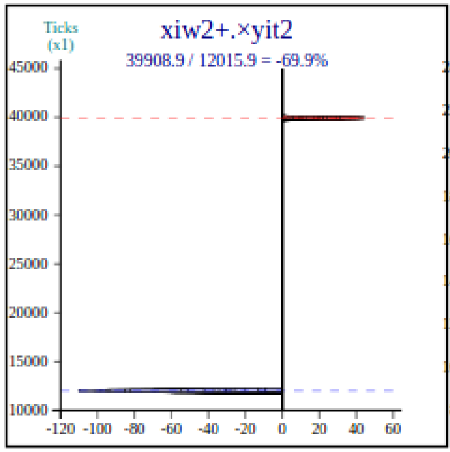
Best Expressions

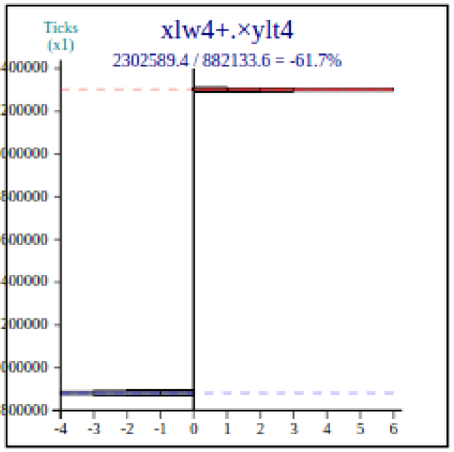


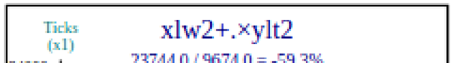


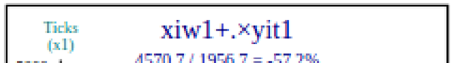




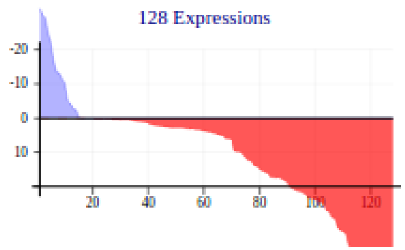








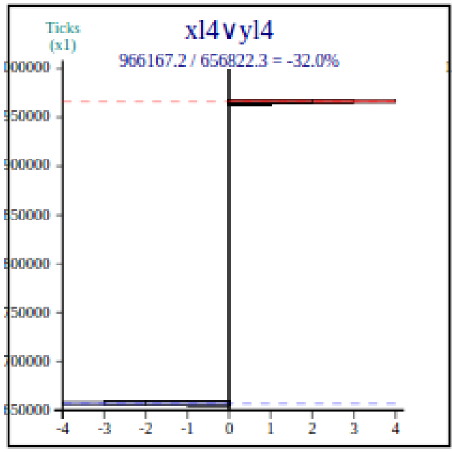
+13.2% : (xb0∧yb0) [#1 of 30 Worst Groups]

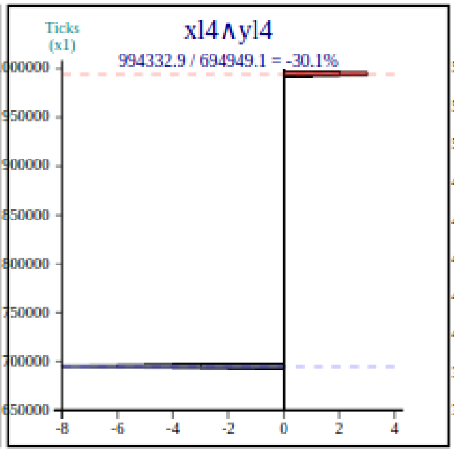


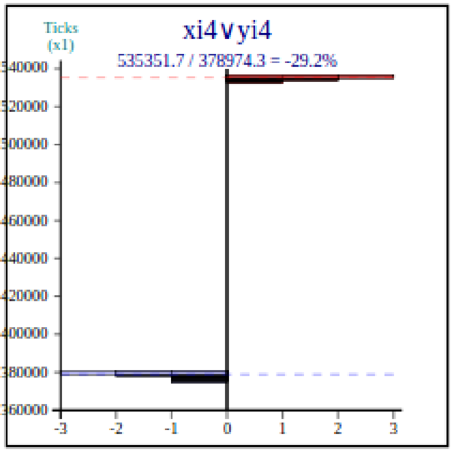
Best: xl4∨yl4 (-32.0%), xl4∧yl4 (-30.1%), xi4∨yi4 (-29.2%), xi4∧yi4 (-24.6%), xs4∨ys4 (-22.1%), xs4∧ys4 (-16.4%), xl4∧yi4 (-13.4%), xi4∧yl4 (-13.3%), xl4∨yi4 (-11.3%), xi4∨yl4 (-10.1%), xi4∨ys4 (-5.0%), xs4∨yi4 (-4.3%)

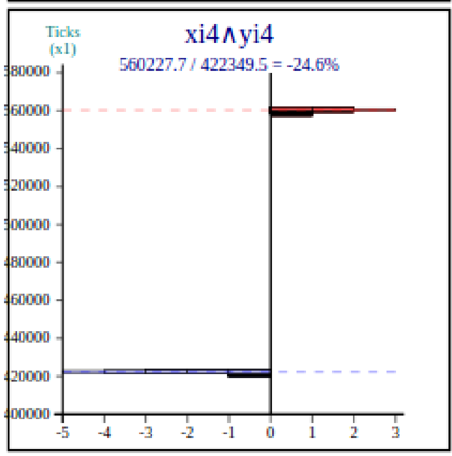
Worst: xi2∨yl2 (+77.3%), xl2∨yi2 (+76.5%), xs2∨yl2 (+74.4%), xl2∨ys2 (+73.2%), xi2∨yi2 (+70.0%), xl2∨yl2 (+68.8%), xi2∨ys2 (+65.4%), xs2∨yi2 (+61.9%), xl2∧yi2 (+47.5%), xi2∧yi2 (+46.4%), xi2∧yl2 (+45.8%), xs2∧yi2 (+45.2%)

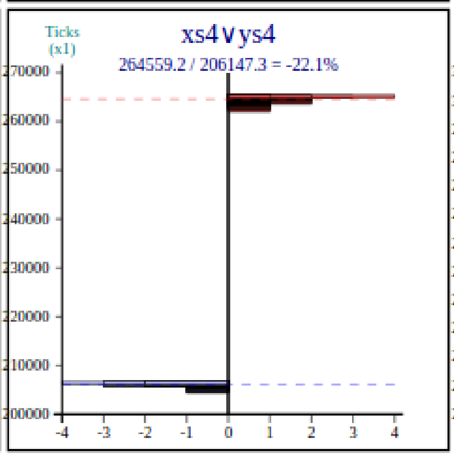
Best Expressions

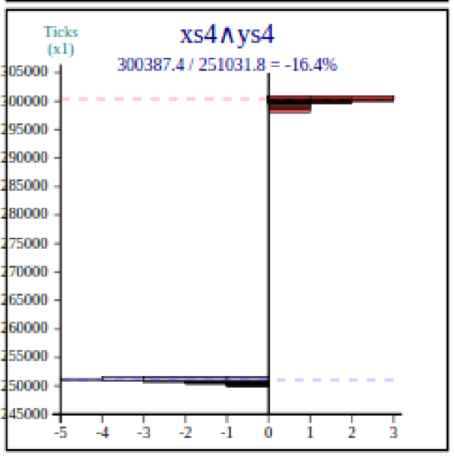










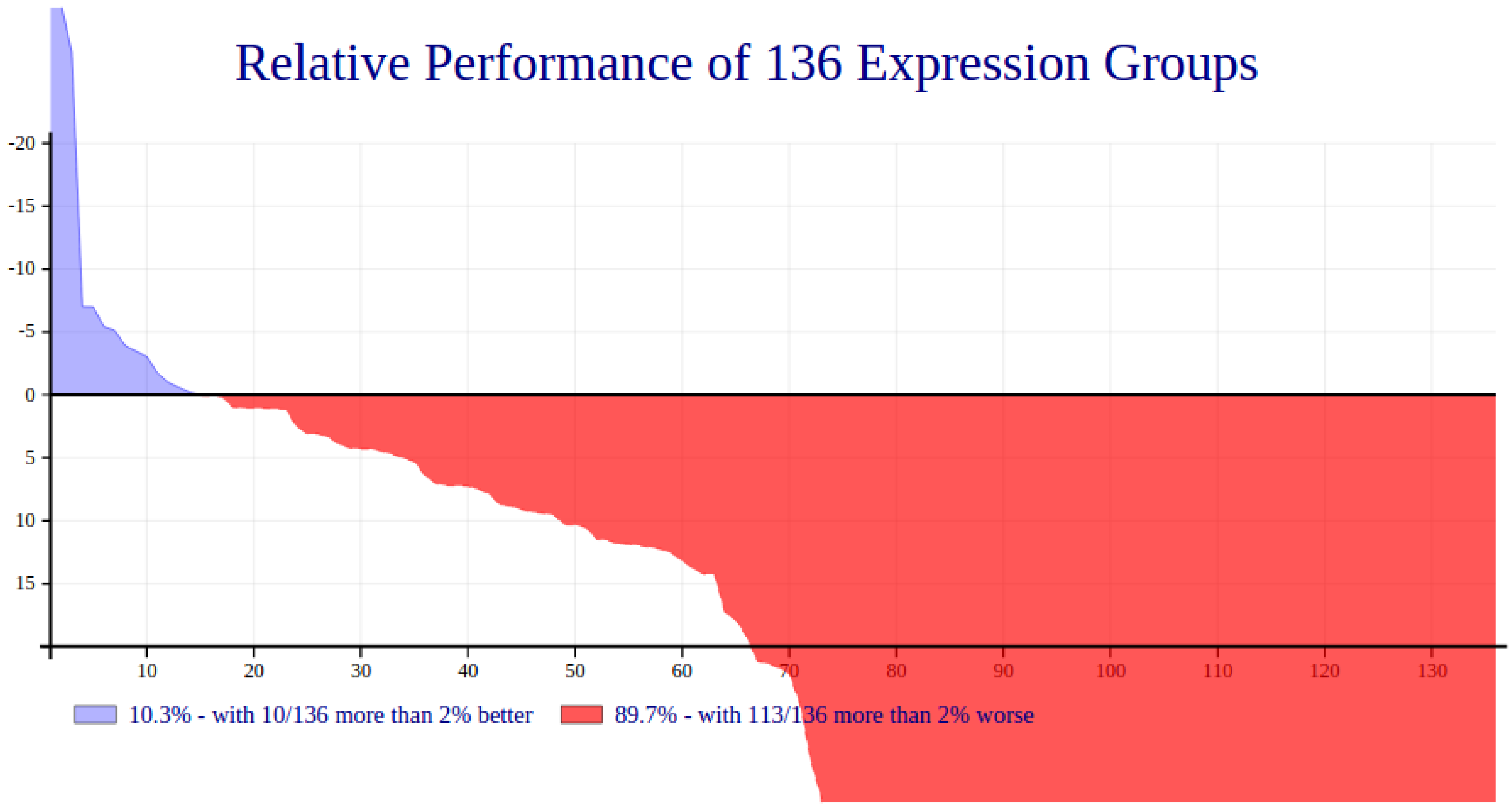


Windows: Visual Studio

- 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

Performance Comparison
Between Windows-64 15.0.25086.0 W Development vs2015.pqa and vs2005.pqa



---

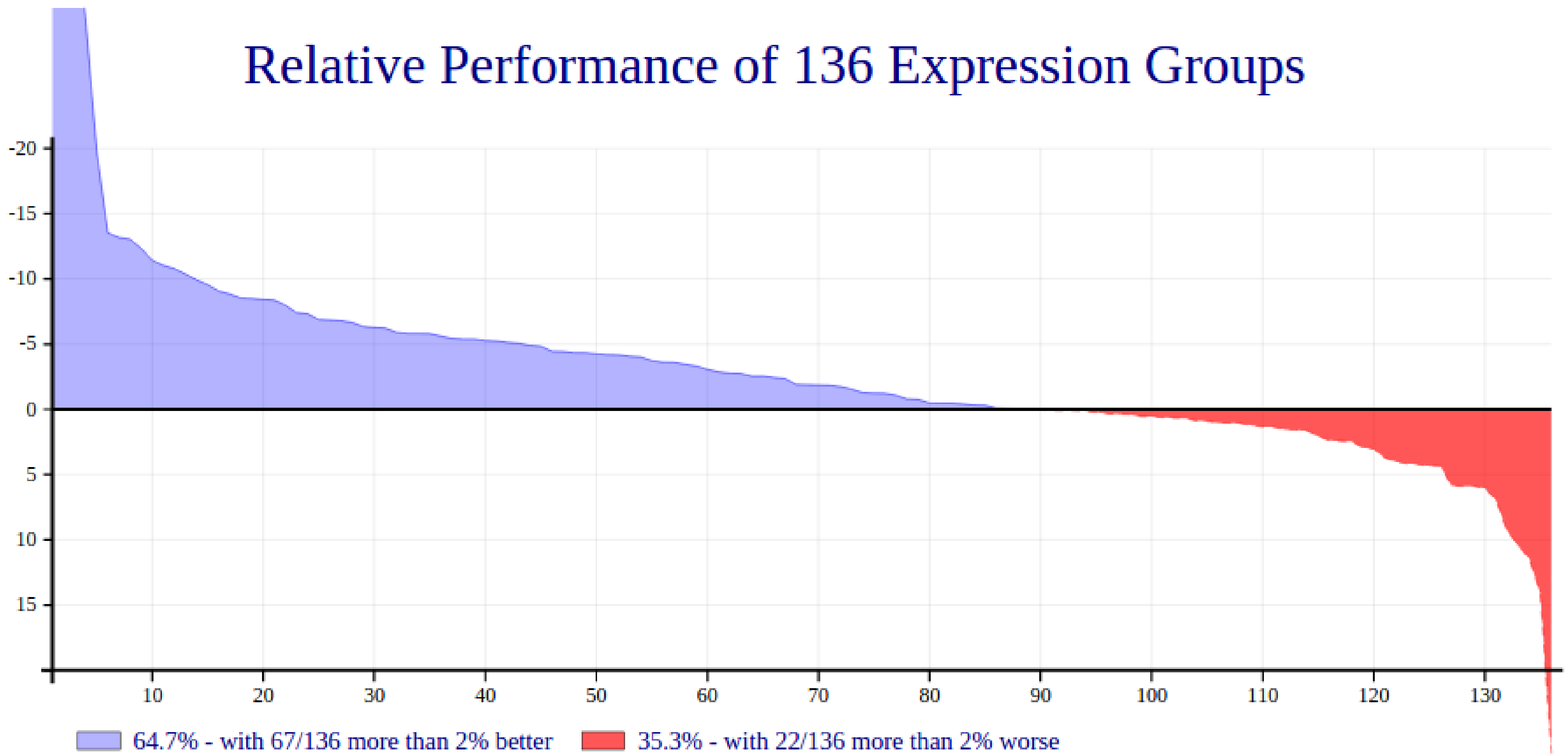
30 Best Groups

Group	#	mean	min	25%	median	75%	max	stddev
1. ⌸dq1	3	-35.8%	-61.8%	-61.8%	-44.4%	-1.2%	-1.2%	0.3120
2. xsp0⍟ysp0	68	-31.1%	-48.4%	-44.0%	-33.3%	-10.8%	-2.9%	0.1510
3. dv2⌸dt2	2	-27.2%	-29.4%	-29.4%	-27.2%	-25.0%	-25.0%	0.0308
4. [illegible]at1	24	-7.0%	-21.6%	-14.3%	-8.0%	+0.7%	+6.0%	0.0828

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

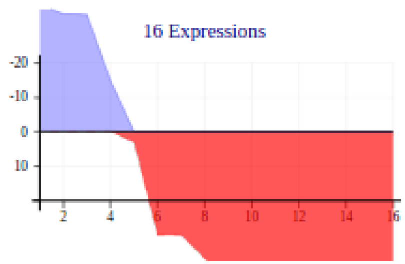


---

30 Best Groups

Group	#	mean	min	25%	median	75%	max	stddev
1. xbq1+.×ybq1	42	-38.8%	-75.8%	-62.8%	-54.3%	-6.6%	+4.9%	0.2933
2. ⌹dq1	3	-38.5%	-61.8%	-61.8%	-47.4%	-6.1%	-6.1%	0.2891
3. xsp0⍟ysp0	68	-37.4%	-49.4%	-46.8%	-39.1%	-25.4%	-12.0%	0.1055
4. dv2⌹dt2	2	-30.0%	-32.3%	-32.3%	-30.0%	-27.8%	-27.8%	0.0314

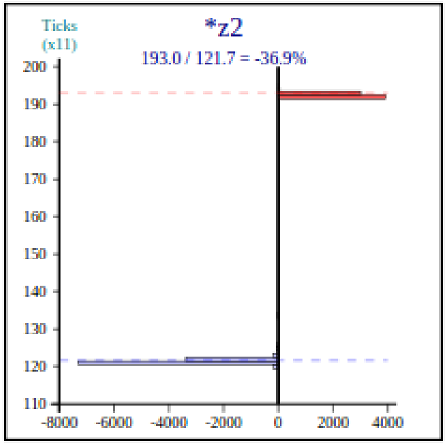
+26.4% : (*b0) [#1 of 30 Worst Groups]

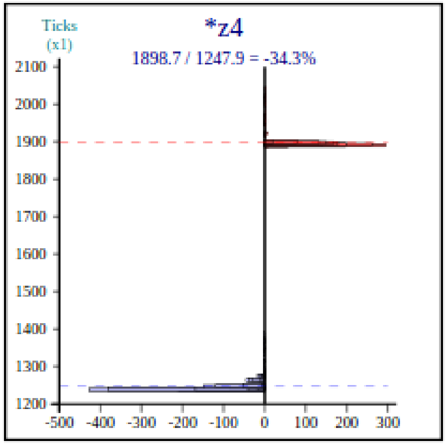


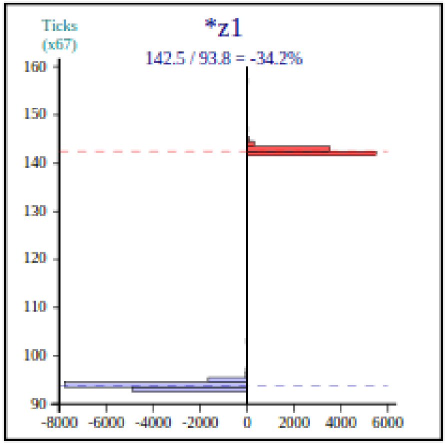
Best: *z2 (-36.9%), *z4 (-34.3%), *z1 (-34.2%), *z0 (-15.0%)

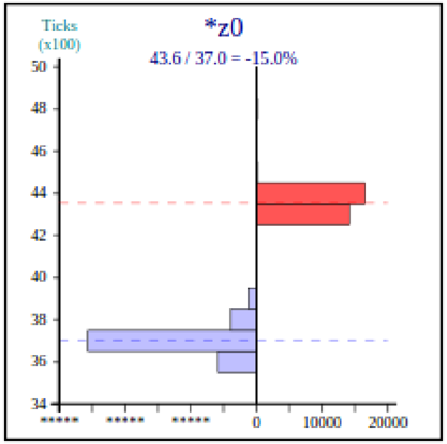
Worst: *b2 (+71.4%), *b4 (+66.0%), *s4 (+55.5%), *s2 (+53.1%), *d2 (+53.1%), *d4 (+52.4%), *s1 (+49.1%), *d1 (+41.9%)

Best Expressions









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 (`+.×`)
- Inner loop in C:

```
for(k=0; k<p; ++k)
    *zp++ += x * *rargp++;
```

Auto-vectorisation

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

```
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)
```

Auto-vectorisation

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

```
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

intel Intrinsics Guide

Technologies

- MMX
- SSE
- SSE2
- SSE3
- SSSE3
- SSE4.1
- SSE4.2
- AVX
- AVX2
- FMA
- AVX-512
- KNC
- SVML
- Other

Categories

- Application-Targeted
- Arithmetic
- Bit Manipulation
- Cast
- Compare
- Convert
- Cryptography
- Elementary Math Functions
- General Support
- Load
- Logical
- Mask
- Miscellaneous
- Move

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. ×

addpd × ?

`__m128d _mm_add_pd (__m128d a, __m128d b)` addpd

Synopsis

```
__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

```
FOR j := 0 to 1
        i := j*64
        dst[i+63:i] := a[i+63:i] + b[i+63:i]
ENDFOR
```

Performance

Architecture	Latency	Throughput
Haswell	3	0.8
Ivy Bridge	3	1
Sandy Bridge	3	1
Westmere	3	1
Nehalem	3	1

`__m128d _mm_mask_add_pd (__m128d src, __mmask8 k, __m128d a, __m128d b)` vaddpd

`__m128d _mm_maskz_add_pd (__mmask8 k, __m128d a, __m128d b)` vaddpd

Manual vectorisation

- Reductions: `⌈/ ⌊/ +/`
- Intel & AMD:
  SSE2, SSE4.1
  AVX, AVX2 (to come)
- Factor of 1 to 30 for `⌈/ ⌊/`
- Factor of 2 to 5 for `+/`

Manual vectorisation

- Reductions: ⌈/ ⌊/ +/

- POWER architecture: AltiVec, VSX
- Factor of 2 to 20 for ⌈/ ⌊/
- 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