Widening

Ian McIntosh
Toronto IBM Lab

October 6, 2004
CASCON 2004

What is Widening?

- An optimization in the latest versions of the IBM XL compiler family
  - ƒ C and C++ version 7
  - ƒ Fortran version 9
- Active at opt level 3 or higher
- Some aspects depend on -qarch, -qtune and other options

What is Widening?

- Purpose is to replace multiple "narrow" instructions with fewer "wider" ones
- Narrow means smaller than a register can hold
- Wide means as wide as a register can hold
- Different kinds of registers are different widths and allow different operations
- Also known as "Short Vector Auto SIMDization"

What does Widening do?

Finds "narrow" stores into contiguous addresses, fed by:

- literals
- loads from contiguous addresses
- parallelizable expressions

and if possible replaces them with widened moves, loads, operations and stores.

Complements -O5 Loop SIMDization.

Examples

These examples show instructions similar to what would appear in listings, but before register allocation and mapping to hardware instructions.

All are taken from actual compiler output, compressed for clarity.

Example 1 - Initializing

```
typedef  struct
    { short a; short b; char c; char d; short e; } s;
s  s1;

main ( )
{
  s1.a = 1;  s1.b = 2;  s1.c = 3;  s1.e = 5;  s1.d = 4;
}
```

Note: Field sizes vary, and assignments are not all in order.

Example 1 Without Widening

```
L8      gr548=.s1(gr2,0)
ST2Z    s1.a(gr548,0)=1
ST2Z    s1.b(gr548,2)=2
ST1Z    s1.c(gr548,4)=3
ST2Z    s1.e(gr548,6)=5
ST1Z    s1.d(gr548,5)=4
```

Example 1 With Widening

```
L8      gr548=.s1(gr2,0)
ST8     s1(gr548,0)
        =0x0001000203040005
```

Example 2 - Copying

```
typedef  struct
    { short a; short b; char c; char d; short e; } s;
s  s1, s2;

main ( )
{
  s2.a = s1.a; s2.b = s1.b;
  s2.c = 6; s2.e = 8; s2.d = 7;
}
```

Note: Some s2 fields are copied from s1, some set to literals.

Example 2 Without Widening

```
L8     gr549=.s1(gr2,0)
L2Z    gr550=s1.a(gr549,0)
L8     gr552=.s2(gr2,0)
ST2Z   s2.a(gr552,0)=gr550
L2Z    gr553=s1.b(gr549,2)
ST2Z   s2.b(gr552,2)=gr553

ST1Z   s2.c(gr552,4)=6
ST2Z   s2.e(gr552,6)=8
ST1Z   s2.d(gr552,5)=7
```

Example 2 With Widening

```
    L8   gr549=.s1(gr2,0)
    L4Z  gr555=s1.|a|b
(gr549,0)
    L8   gr552=.s2(gr2,0)
    ST4Z s2.|a|b(gr552,0)
=gr555
    ST4Z s2.|c|d|e(gr552,4)
         =0x06070008
```

Example 3 - Bit Expressions

```
typedef struct  {char a; char b; char c; char d;} s;
s  s1={...}, s2={...}; s3={...};
main ( )  {
  s  s4;
  s4.a = s1.a | (s2.a & s3.a);
  s4.b = s1.b | (s2.b & s3.b);
  s4.c = s1.c | (s2.c & s3.c);
  s4.d = s1.d | (s2.d & s3.d);
}
```

Note: For expressions, fields must all be the same size.

Example 3 Without Widening 1/4

```
L8      gr548=.s1(gr2,0)
L1Z     gr549=s1.a(gr548,0)
L8      gr550=.s2(gr2,0)
L1Z     gr551=s2.a(gr550,0)
L8      gr552=.s3(gr2,0)
L1Z     gr553=s3.a(gr552,0)
N       gr554=gr551,gr553
O       gr555=gr549,gr554
ST1Z    s4.a(grauto,0)=gr555
```

Example 3 Without Widening 2/4

```
... L1Z   gr557=s1.b(gr548,1)
    L1Z   gr558=s2.b(gr550,1)
    L1Z   gr559=s3.b(gr552,1)
    N     gr560=gr558,gr559
    O     gr561=gr557,gr560
    ST1Z  s4.b(grauto,1)=gr561
...
```

Example 3 Without Widening 3/4

```
... L1Z   gr563=s1.c(gr548,2)
    L1Z   gr564=s2.c(gr550,2)
    L1Z   gr565=s3.c(gr552,2)
    N     gr566=gr564,gr565
    O     gr567=gr563,gr566
    ST1Z  s4.c(grauto,2)=gr567
...
```

Example 3 Without Widening 4/4

```
... L1Z    gr569=s1.d(gr548,3)
    L1Z    gr570=s2.d(gr550,3)
    L1Z    gr571=s3.d(gr552,3)
    N      gr572=gr570,gr571
    O      gr573=gr569,gr572
    ST1Z   s4.d(grauto,3)=gr573
```

Example 3 With Widening

```
L8     gr548=.s1(gr2,0)
L4A    gr578=s1(gr548,0)
L8     gr550=.s2(gr2,0)
L4A    gr576=s2(gr550,0)
L8     gr552=.s3(gr2,0)
L4A    gr575=s3(gr552,0)
N      gr577=gr575,gr576
O      gr579=gr577,gr578
ST4A   s4(grauto,0)=gr579
```

Example 4 - Integer Expressions

```
typedef struct  {short a; short b; . . . short h;} s;
s  s1={...}, s2={...}; s3={...};
main ( )  {
  s  s4;
  s4.a = s1.a - (s2.a + s3.a + 1);
  s4.b = s1.b - (s2.b + s3.b + 2);
        . . .
  s4.h = s1.h - (s2.h + s3.h + 8);
}
```

Example 4 Without Widening 1/8

```
L8    gr516=.s1(gr515,0)
L2A   gr517=s1.a(gr516,0)
L8    gr518=.s2(gr515,0)
L2A   gr519=s2.a(gr518,0)
L8    gr520=.s3(gr515,0)
L2A   gr521=s3.a(gr520,0)
A     gr522=gr519,gr521
AI    gr523=gr522,1
S     gr524=gr517,gr523
ST2A  s4.a(grauto,0)=gr524...
```

Example 4 Without Widening 2-7/8

• • •

• • •

• • •

• • •

• • •

• • •

Example 4 Without Widening 8/8

```
...L2A  gr568=s1.h(gr516,14)
   L2A  gr569=s2.h(gr518,14)
   L2A  gr570=s3.h(gr520,14)
   A    gr571=gr569,gr570
   AI   gr572=gr571,8
   S    gr573=gr568,gr572
   ST2A s4.h(grauto,14)=gr573
```

Example 4 With VMX Widening

```
L8       gr516=.s1(gr515,0)
VLQ      vr517=s1(gr516,0)
L8       gr518=.s2(gr515,0)
VLQ      vr519=s2(gr518,0)
L8       gr520=.s3(gr515,0)
VLQ      vr521=s3(gr520,0)
VADDUHM  vr522=vr519,vr521
VADDUHM  vr523=vr522,0x0001...08
VSUBUHM  vr524=vr517,vr523
VSTQ     s4(grauto,0)=vr524
```

Notes

- Widening may also use floating point registers for 8 byte data movement.
- In addition to 1, 2 and 4 byte signed and unsigned integers, VMX handles single precision floating point.
- VMX Widening does not handle all operations (eg, integer divide). It also does not handle interesting VMX operations like saturated arithmetic.

Questions and Answers

?