#SHAREorg

SHARE
Technology · Connections · Results

12340: Make Your C/C++ and PL/I Code FLY With the Right Compiler Options

Visda Vokhshoori
IBM
Peter Elderon
IBM

SHARE in San Francisco 2013

What does good application performance mean to you?

What?

- Fast execution time
- Short compile time
- Small load module size

How do you achieve good application performance?

How?

- Use the best compiler options
- Write good code
- Install newer hardware

The best compiler options

Compiler options

- Most compiler options have no effect on performance
- But some have a major impact, particularly
  - ARCH (and TUNE)
  - OPT
- Others are also important, e.g.
  - FLOAT(AFP)
  - HGPR
  - etc

The ARCH option

- The ARCH option specifies the level of the hardware on which the generated code must run

- ARCH(6) is the PL/I default – it produces code that will run on old z990 machines

- ARCH(5) is the C/C++ default – it produces code that will run on the even older z900

ARCH

- The ARCH option is dangerous
  – but only if misused

- If you specify ARCH(n) and run the generated code on an ARCH(n-1) machine, you will most likely get an operation exception

- So you must set ARCH to the lowest level machine where your generated code will run

ARCH option overview

	Machine supported	Hardware features exploited
ARCH(6)	z990	Long displacement, Load Bbyte,…
ARCH(7)	z9	Decimal Floating Point, Extended Immediate, Extended Translate, …
ARCH(8)	z10	Compare and Branch, Add Logical with Signed Immediate, …
ARCH(9)	z196	Conditional Load/Store, Non-destructive ops, Population count, …
ARCH(10)	ec12	DFP-Zoned Conversions

ARCH(6)

- Both PL/I and C/C++ exploit the

- The long-displacement facility
- The load byte instruction
- Additional floating point load and store instructions

- The long-displacement provides a huge improvement over the code generated under ARCH(5) – for this reason alone, unless your code runs on machines older than a z9, don't use the ARCH(5) default.

ARCH(7)

- Both PL/I and C/C++ exploit some of the

- Extended immediate facility
- Extended translation facility
- Decimal floating point (DFP) instruction set

ARCH(7)

- On the z9, the DFP instructions were in software (aka millicode)
- On z10, they are in the hardware
- So, they are much faster on z10 (or arch(8)) machines
- This is often true of some significantly new instructions - sometimes the compiler will defer using them until they are in the hardware and actually make for faster code

ARCH(7)

- PL/I, for example, uses

- TRTR to inline some verifyr and searchr
- SRSTU to inline some widechar index
- CU12 and CU21 to inline some UTF valid functions
- CU12 and CU21 to inline UTF-8 <-> UTF-16

ARCH(8)

- Both PL/I and C/C++ exploit some of the

- Compare-and-branch
- General instruction extensions facility
- More decimal floating point (DFP) instructions
  - For example to convert DFP to HEX or BINARY float

ARCH(8)

- General instruction extensions facility includes

- ADD LOGICAL WITH SIGNED IMMEDIATE
- COMPARE AND BRANCH (RELATIVE)
- COMPARE (HALFWORD) RELATIVE LONG
- COMPARE IMMEDIATE AND BRANCH (RELATIVE)
- COMPARE LOGICAL AND BRANCH (RELATIVE)
- COMPARE LOGICAL IMMEDIATE AND BRANCH (RELATIVE)
- COMPARE LOGICAL RELATIVE LONG
- LOAD HALFWORD RELATIVE LONG
- LOAD LOGICAL (HALFWORD) RELATIVE LONG
- LOAD RELATIVE LONG
- MULTIPLY SINGLE IMMEDIATE
- STORE (HALFWORD) RELATIVE LONG

ARCH(9)

- Both PL/I and C/C++ exploit some of the

- Load/store-on-condition facility
- Distinct-operands facility
- Population-count facility

ARCH(9): Load-on-condition example

consider this small program:

```
 2.0 | test: proc returns( fixed bin(31) );
 3.0 |
 4.0 |   exec sql include sqlca;
 5.0 |
 6.0 |   dcl c fixed bin(31);
 7.0 |
 8.0 |   exec sql commit;
 9.0 |
10.0 |   if sqlcode = 0 then
11.0 |     c = 0;
12.0 |   else
13.0 |     c = -1;
14.0 |
15.0 |   return( c );
16.0 | end;
```

Load-on-condition example …

- Under OPT(3) ARCH(8), the instructions after the call are:

```
0000CA  0DEF            000008 |              BASR    r14,r15
0000CC  5800  D0F4      000010 |              L       r0,<a1:d244:l4>(,r13,244)
0000D0  A718  FFFF      000010 |              LHI     r1,H'-1'
0000D4  EC06  0005 007E 000010 |              CIJNE   r0,H'0',@1L8
0000DA  4110  0000      000010 |              LA      r1,0
0000DE                  000010 |       @1L8  DS      0H
0000DE  58E0  2000      000015 |              L       r14,_addrReturns_Value(,r2,0)
0000E2  5010  E000      000015 |              ST      r1,_shadow1(,r14,0)
```

Load-on-condition example …

- Under OPT(3) ARCH(9), the instructions after the call are:

```
0000CA  0DEF          000008 |        BASR    r14,r15
0000CC  A718  FFFF    000010 |        LHI     r1,H'-1'
0000D0  BF0F  D0F4    000010 |        ICM     r0,b'1111',<a1:d244:l4>(r13,244)
0000D4  58E0  2000    000015 |        L       r14,_addrReturns_Value(,r2,0)
0000D8  4100  0000    000010 |        LA      r0,0
0000DC  B9F2  8010    000010 |        LOCRE   r1,r0
0000E0  5010  E000    000015 |        ST      r1,_shadow1(,r14,0)
```

Load-on-condition example …

- So, under ARCH(8), the code sequence was:

- Load SQLCODE into r0
- Load -1 into r1
- Compare r0 (SQLCODE) with 0 and branch if NE to @1L8
- Load 0 into r1
- @1L8
- Store r1 into the return value

Load-on-condition example …

- But under ARCH(9), the code has no label and no branch:

- Load -1 into r1
- Load SQLCODE into r0 via ICM (so that CC is set)
- Load 0 into r0
- Load-on-condition r1 with r0 if the CC is zero (i.e. if SQLCODE = 0)
- Store r1 into the return value

ARCH(10)

- Both PL/I and C/C++ exploit some of the

- The execution-hint facility
- The load-and-trap facility
- The miscellaneous-instructions-extension facility
- The transactional-execution facility

ARCH(10): Zoned to DFP example

consider this small program:

```
pic2int: proc( ein, aus )
 options(nodescriptor);

  dcl ein(0:100_000) pic'(9)9' connected;
  dcl aus(0:100_000) fixed bin(31) conn;
  dcl jx  fixed bin(31);

  do jx = lbound(ein) to hbound(ein);
    aus(jx) = ein(jx);
  end;
end;
```

Zoned to DFP example …

- Under ARCH(9), the heart of the loop consists of these 8 instructions

```
0058   F248   D098   1000      PACK    #pd580_1(5,r13,152),_shadow2(9,r1,0)
005E   C020   0000   0021      LARL    r2,F'33'
0064   D204   D0A0   D098      MVC     #pd581_1(5,r13,160),#pd580_1(r13,152)
006A   4110   1009             LA      r1,#AMNESIA(,r1,9)
006E   D100   D0A4   200C      MVN     #pd581_1(1,r13,164),+CONSTANT_AREA(r2,12)
0074   F874   D0A8   D0A0      ZAP     #pd582_1(8,r13,168),#pd581_1(5,r13,160)
007A   4F20   D0A8             CVB     r2,#pd582_1(,r13,168)
007E   502E   F000             ST      r2,_shadow1(r14,r15,0)
```

Zoned to DFP example …

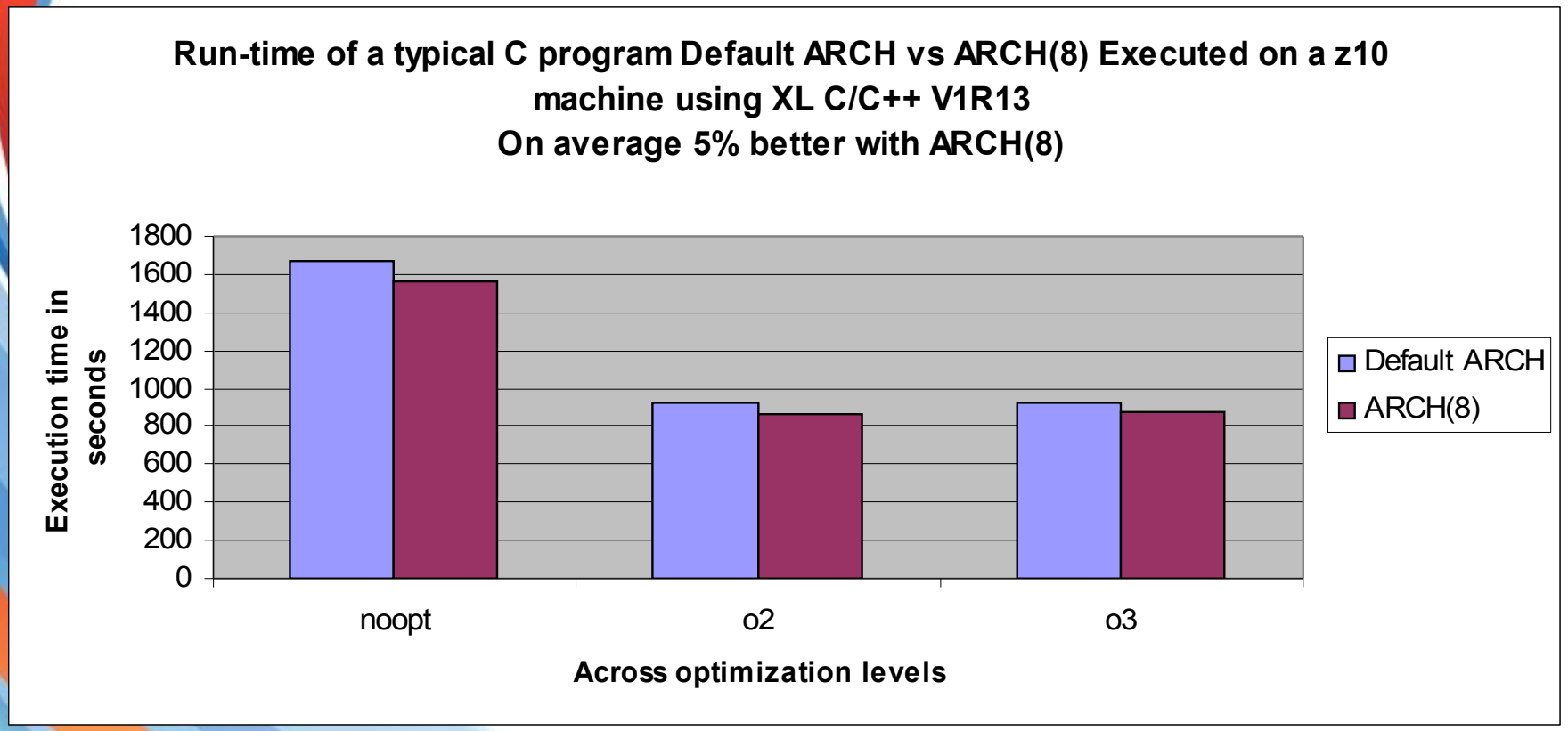
- While under ARCH(10), it consists of 9 instructions and uses DFP in several of them – but since only the ST and the new CDZT refer to storage, the loop runs more than 66% faster

```
0060   EB2F   0003   00DF          SLLK     r2,r15,3
0066   B9FA   202F                 ALRK     r2,r15,r2
006A   A7FA   0001                 AHI      r15,H'1'
006E   B9FA   2023                 ALRK     r2,r3,r2
0072   ED08   2000   00AA          CDZT     f0,#AddressShadow(9,r2,0),b'0000'
0078   B914   0000                 LGFR     r0,r0
007C   B3F6   0000                 IEDTR    f0,f0,r0
0080   B941   9020                 CFDTR    r2,b'1001',f0
0084   5021   E000                 ST       r2,_shadow1(r1,r14,0)
```

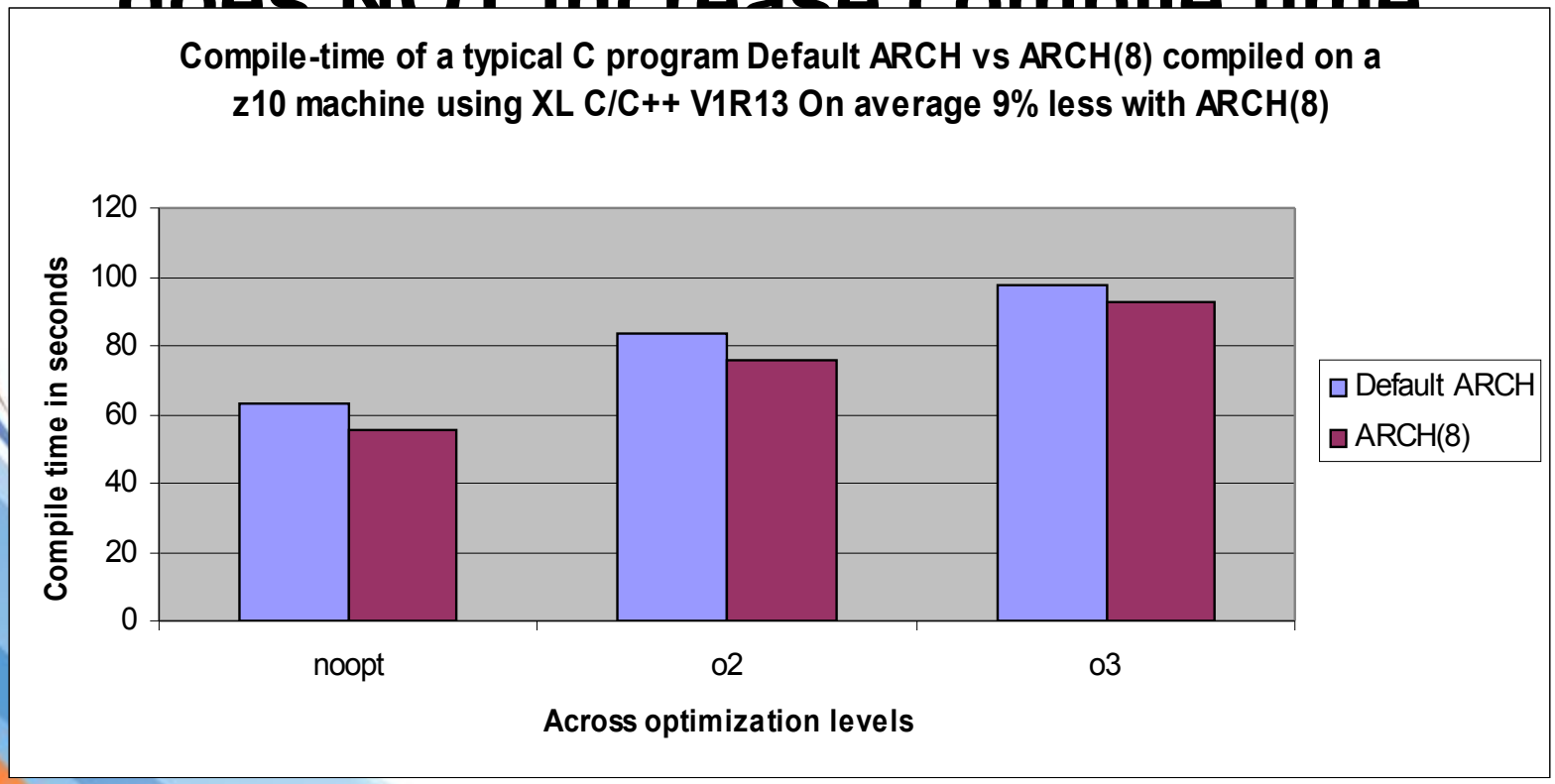
Zoned to DFP example …

- Some lessons from this example:
  - A longer set of instructions may be faster than a shorter set
  - Reducing storage references helps performance
  - Eliminating packed decimal instructions can help performance
  - Using decimal-floating-point may improve your code's performance even in program's that have no floating-point data
- And best-of-all: the compiler will figure out this for you

The right ARCH option does decrease run time



The right ARCH option does NOT increase compile time



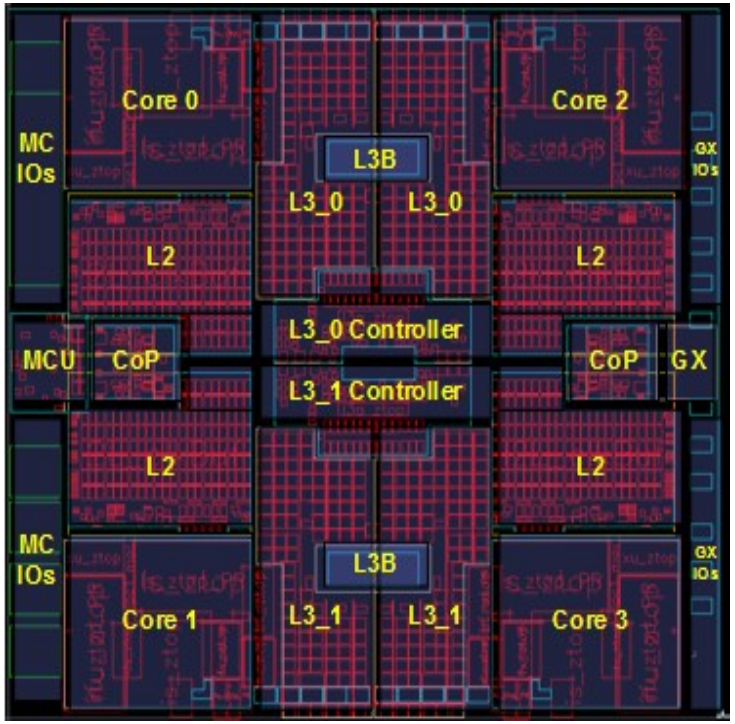
TUNE

- This option controls instruction scheduling etc to produce the best code with the given ARCH option but to be run on the TUNE machine

- E.g. TUNE(9) ARCH(5) is safe and produces code that will run on an ARCH(5) machine but perform best on an ARCH(9) machine

- C/C++ default is TUNE(5)

- PL/I no longer has an external TUNE option and instead sets TUNE equal to ARCH



Recommended values for ARCH and TUNE options

Choose the highest value for the ARCH option that will produce code that can still run on your oldest machine (and, if you are a vendor, on the oldest machine of your customers).

For C/C++ applications, we recommend setting the TUNE option to be at least equal to the ARCH option value.

(For PL/I, the compiler sets TUNE equal to the ARCH option)

The OPTIMIZE option

- The OPTIMIZE, or OPT, option controls how much, or even if at all, the compiler tries to optimize your code
- Under the default OPT(0) option, the compiler simply translates your code into machine code source line by source line
- The code generated under OPT(0) will be large and very slow
- OPT(0) is the best choice when you want to use a debugger
- OPT(0) also requires the least compile-time - by far
- But OPT(0) is a terrible choice if you care about your runtime performance

The OPTIMIZE option

- When optimizing, the compiler will improve, often vastly, the code it generates by, for example
  - Keeping intermediate values in registers
  - Moving code out of loops
  - Merging statements
  - Reordering instructions to improve the instruction pipeline
  - Inlining functions

- All this can make debugging much harder

- And OPT(n) with n > 0 can require **much more** CPU and REGION during the compilation

The OPTIMIZE option

- The PL/I and C++ compilers use the same optimizing backend, but there are differences in what OPT suboptions they support and what they mean:

- PL/I's OPT(2) is a crippled version of C's OPT(2)
  - It helps compile-time be reasonable for large programs
  - But it does produce less than optimal code

- PL/I's OPT(3) is the same as C's OPT(2)
  - It can use a lot of CPU and REGION
  - But it will produce very good and safe code

- C/C++ has an OPT(3) that is even more aggressive

The OPTIMIZE option

- Some of the additional optimizations under C/C++ OPT(3) include:
  - Aggressive code motion and scheduling on computations that have the potential to raise an exception
  - Conformance to IEEE rules are relaxed
  - Floating-point expressions may be rewritten
  -
- You can use the STRICT option to turn off the aggressive optimizations that might change the semantics of a program
- This optimization level will consumer even more cpu and region during compile time

The OPTIMIZE option

- C/C++ also allows you to compile your optimized code HOT
- Under the HOT option, the compiler performs
  - **H**igh-**O**rder loop analysis and **T**ransformations
  -
- It may improve the code generated for some of the loops in your apps
- But it will consume yet more cpu and region
- It is not supported with the OPT(0) option

Other compiler options important for performance

- FLOAT(AFP(NOVOLATILE))
- HGPR
- UNROLL
- Inlining
  - C/C++: INLINE
  - PL/I: DFT(INLINE)

Other compiler options important for performance

- These options must be used with intelligence
- For example, UNROLL can make your code not only faster, but also bigger
- Inlining can also greatly increase your object size and sometimes even make the code slower. We have seen a C program that performed 4X worse with INLINE

Other PL/I compiler options important for performance

- REDUCE
- RESEXP
- RULES(NOLAXCTL)
- DEFAULT(REORDER NOOVERLAP CONNECTED)

Other C/C++ compiler options important for performance

- XPLINK
- ANSIALIAS
- IPA
- COMPACT
- STRICT
- STRICT_INDUCTION

XPLINK

- A modern linkage convention that is 2.5 times more efficient than the conventional linkage convention
- We have seen some programs improve by 30%
- You cannot statically link non-XPLINK with XPLINK
- You can call non-XPLINK DLLs from XPLINK DLLs and vice-versa but you must tell the compiler about this so that it can insure the (expensive) switching code gets executed
- If your application contains few switches (as is true of the PL/I compiler where the frontend is not XPLINK and the backend is), then mixing will be beneficial; otherwise it may be very costly

Choosing the right options is important for performance, but

- Even more important is
- Writing good code

Warning

- Attempts to be clever and produce “optimal” code have produced:
  - Code that is unreadable
  - Code that cannot be maintained
  - Code that performs worse than less clever solutions
  - Code that fails!
- Readability trumps speed

Warnung

- Wegen des Versuchs klug zu erscheinen und optimalen Code zu schreiben habe ich zu oft folgendes gesehen:
  - Programme, die keiner verstehen kann
  - Programme, die keiner reparieren kann
  - Programme, die langsamer laufen als einfachere Loesungen
  - Programme, die einfach abbrechen
- Lesbarkeit vor Schnelligkeit !

Install newer hardware

New hardware

- Requires **no**
  - Recompilation
  - Relinking
  - Migration to a new release
- But can make your code run much faster
- Often the performance boost from moving to new hardware is greater than that from recompiling with the corresponding new ARCH level

Feedback

- We are collecting feedback for future sessions - which topics are you interested in?
- Looking forward to hearing from you!!
- Please email Visda or me

#SHAREorg

SHARE
Technology · Connections · Results

SHARE
in San Francisco
2013