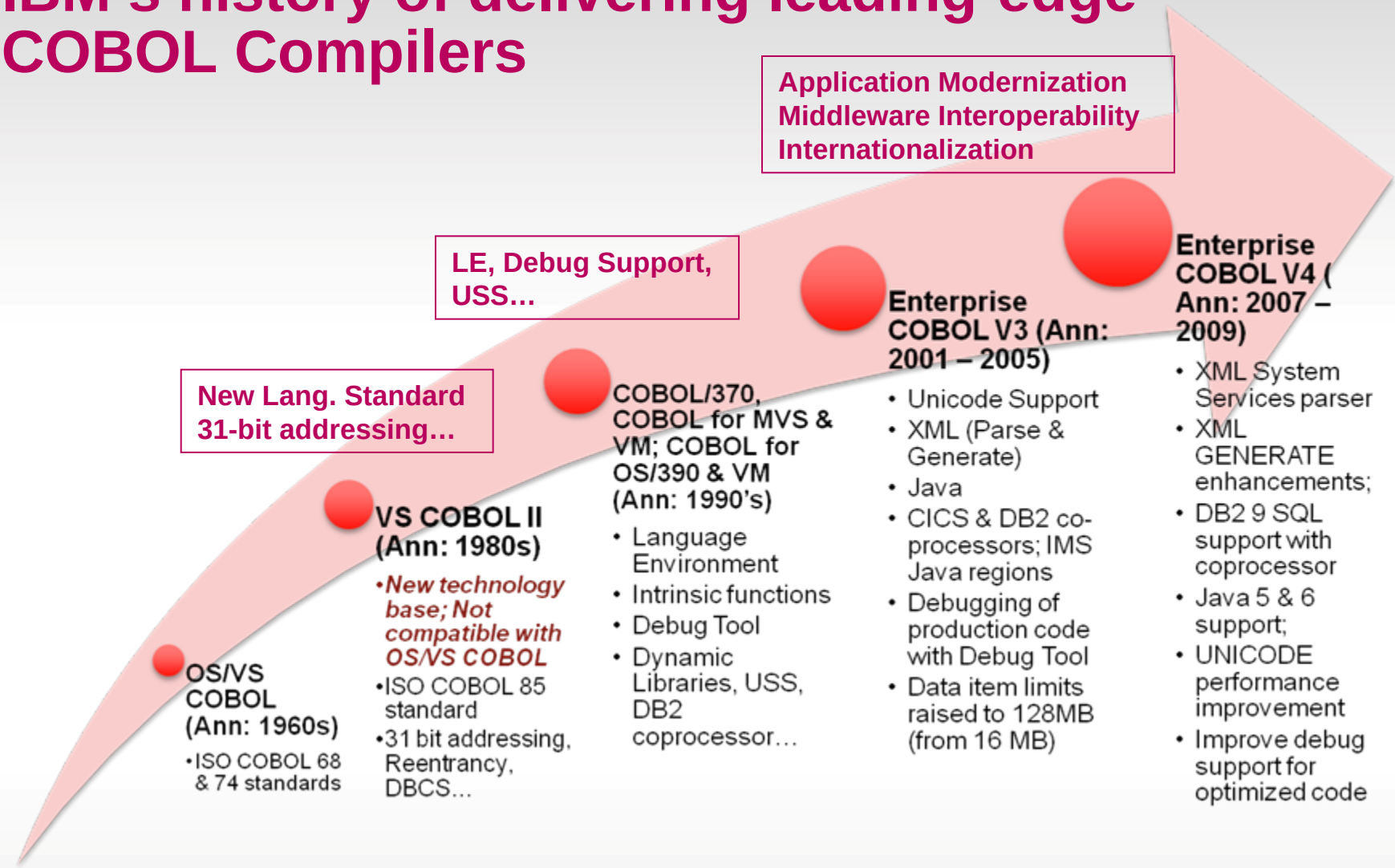
What's current in compilers for the enterprise to improve application performance?

Speaker Name and Title

Agenda

- Enterprise COBOL for z/OS v5.1
- z/OS XL C/C++ v2.1
- Enterprise PL/I for z/OS v4.4

IBM's history of delivering leading-edge COBOL Compilers

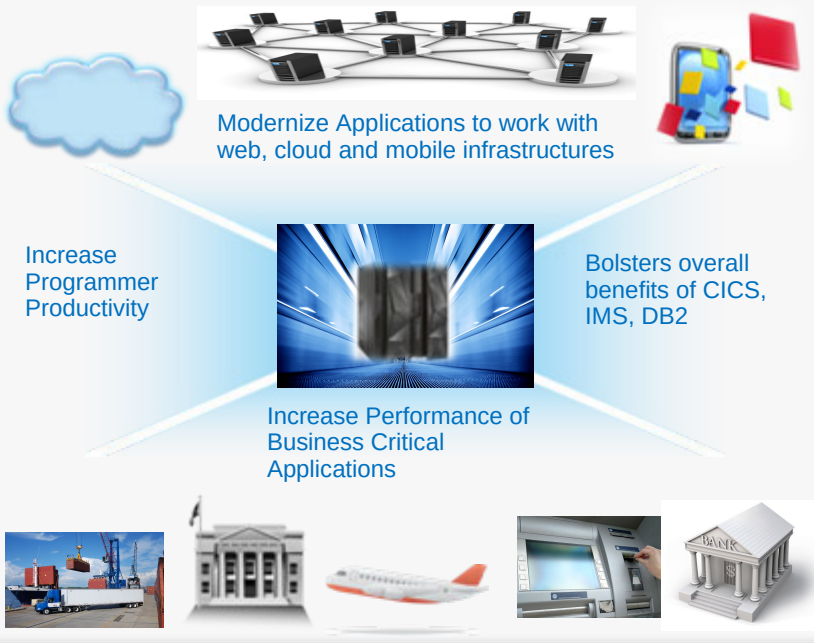


Enterprise COBOL for z/OS v5.1

GA'd June 21

- Advanced technology designed to optimize COBOL programs and fully exploit z hardware
  - Delivers greater than 10% performance improvement over Enterprise COBOL v4 for well structured, CPU-intensive batch applications on System z[1]
  - Many numerically intensive programs have shown performance increases greater than 20%[1]
  - Maintains compatibility with previous COBOL releases
- New programming and application modernization capabilities.
  - Enables users to deliver enhancements to business critical applications quicker with less cost and lower risk
- Allows users, who implement sub-capacity tracking, to reduce administrative overhead
- *Supports the* ecosystem *of COBOL development tools supplied by IBM and ISVs.*

where Tradition Meets Innovation…



"Our testing of COBOL V5 shows a significant performance improvement for math. As a financial services company with a continually narrowing batch window, that improvement is very important to us. It will help us meet our Service Level Agreements and reduce cost driven by CPU utilization."

Michael A Todd, Software Architect
Multi-national financial services company

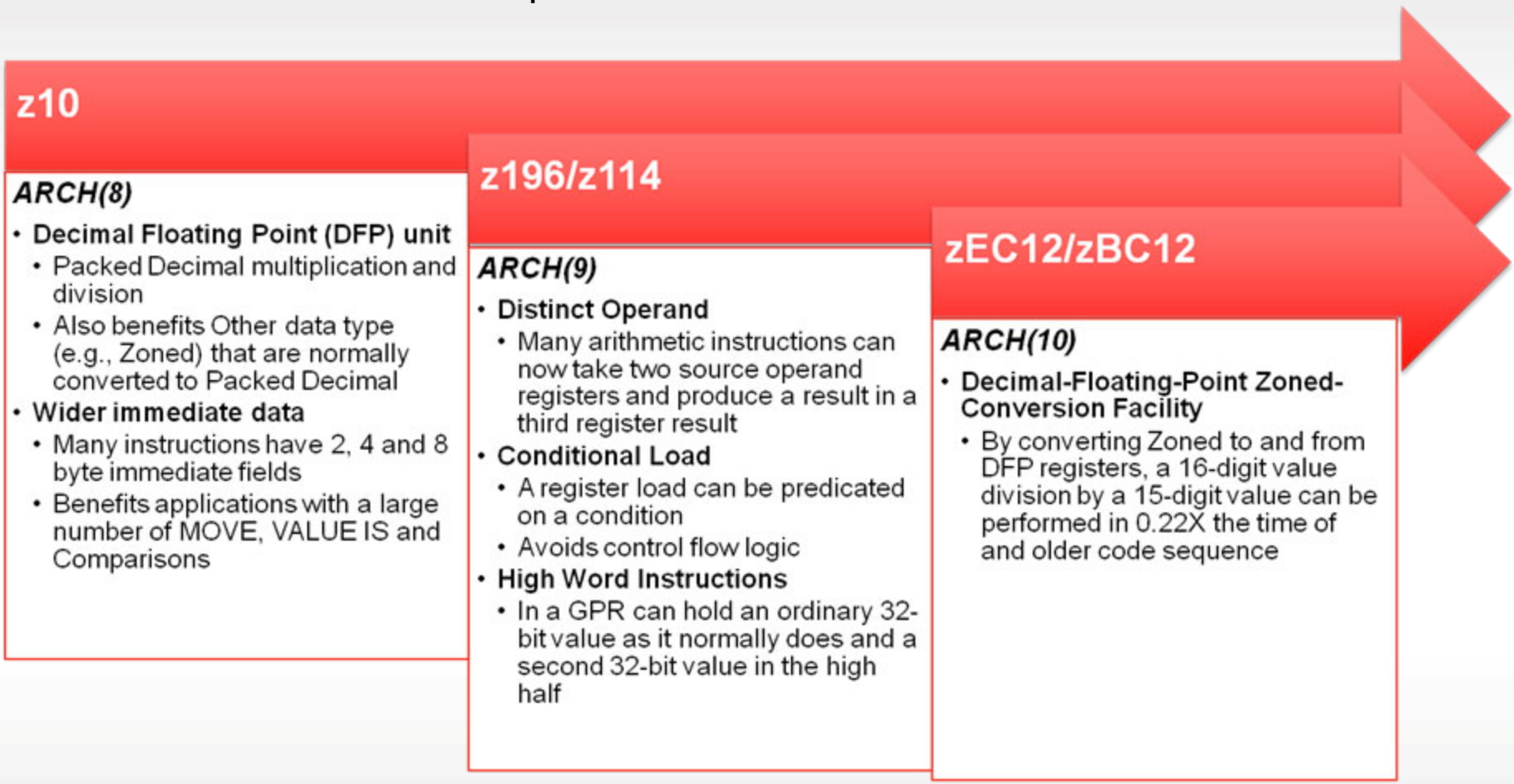
CUSTOMER VALUE

[1] Results are based on an internal compute-intensive test suite. Performance results from other applications may vary.

Exploiting z/Architecture

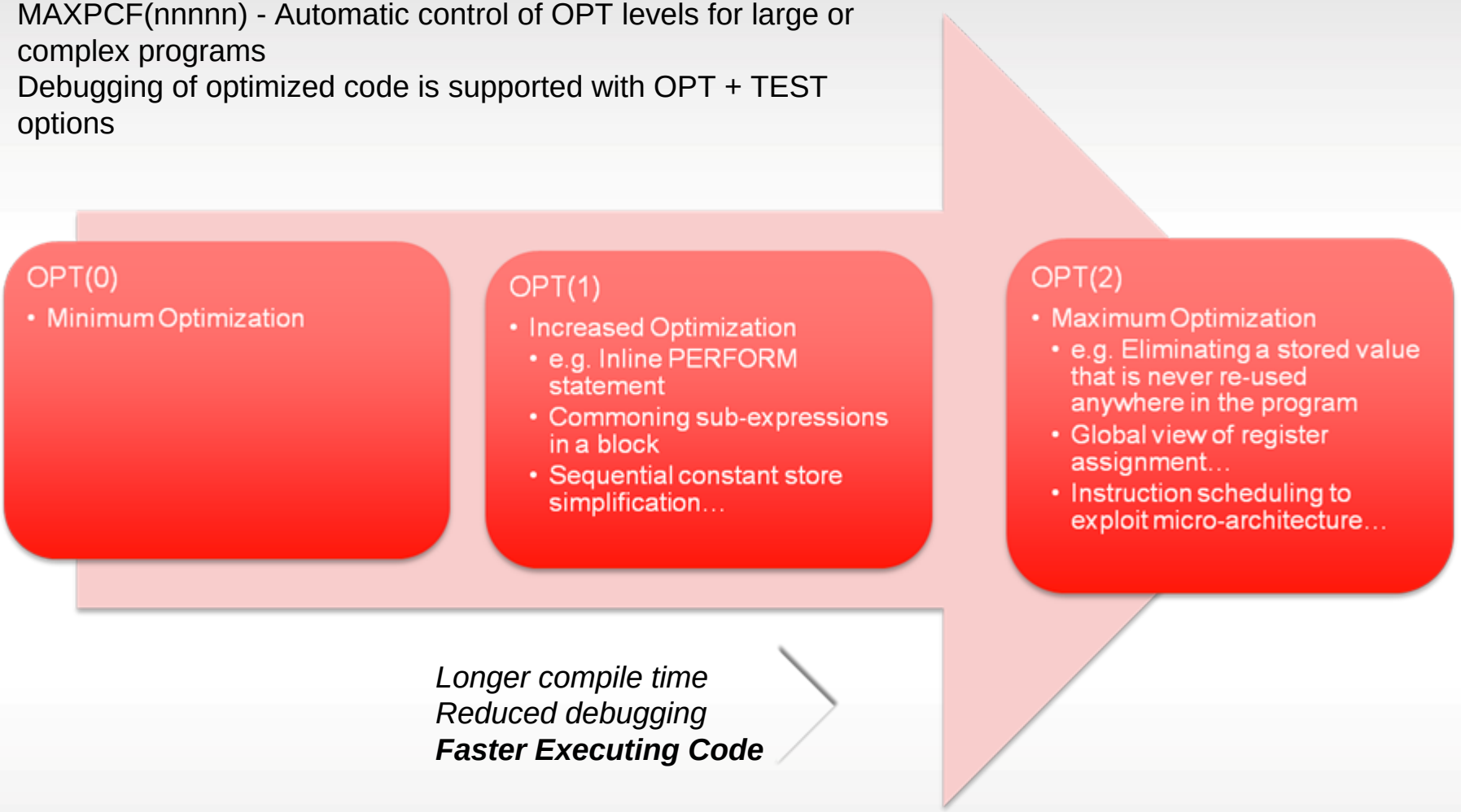
New ***ARCH*** option enables users to fully exploit z/Architecture

- Not available in Enterprise COBOL 4



Advanced Optimizations

- Provide multiple levels of optimization
- MAXPCF(nnnnn) - Automatic control of OPT levels for large or complex programs
- Debugging of optimized code is supported with OPT + TEST options



Exploiting z/Architecture – Example

Decimal Divide Where Operands Exceed Packed Decimal Hardware Limits

```
1   z14v2 pic s9(14)v9(2)
1   z13v2 pic s9(13)v9(2)
. . .
Compute z14v2 = z14v2 / z13v2
```

COBOL V4 – Zoned & Packed decimal arithmetic that cannot fit within hardware instruction limits are implemented by generating a call to a divide or multiply routine in the runtime.

V4

- *Calls out to library routine*
- *Runtime path length is > 100 instructions*

```
PACK   344(9,13),0(16,2)
PACK   360(16,13),16(15,2)
MVC    376(32,13),59(10)
MVC    398(9,13),344(13)
NI     406(13),X'F0'
MVN    407(1,13),352(13)
L      3,92(0,9)
L      15,180(0,3)
LA     1,146(0,10)
BASR   14,15
NI     431(13),X'0F'
ZAP    431(9,13),431(9,13)
UNPK   0(16,2),431(9,13)
```

COBOL V5 - Packed and Zoned decimal arithmetic that does not fit in the hardware instruction limits are converted to DFP for z10 and above. Arithmetic operations are done in DFP unit, then converted back to the original packed/zoned type.

V5

- *Inlined with 6 instructions*
- *CDZT/CZDT are new EC12 instructions to convert between zoned and DFP types*
- *ARCH(10)*

```
CDZT    FP0,152(16,R8),0x8
CDZT    FP1,168(15,R8),0x8
SLDT    FP0,FP2,2
DDTR    FP0,FP0,FP1
FIDTR   FP0,9,FP0
CZDT    FP0,152(16,R8),0x9
```

Performance Comparison

Timing (100 million in a loop)

```
V5 : 1.08 cpu seconds
V4 : 4.81 cpu seconds
```

Advanced Optimization – Example

Instruction Scheduling for Performance

```
 1 z7v2a pic s9(7)v9(2).
 1 z7v2b pic s9(7)v9(2).
 1 z7v2c pic s9(7)v9(2).
...
ADD 1 TO z7v2a z7v2b z7v2c
```

COBOL V4 – each PACK/AP/ZAP instruction group for each receiver is generated in order.

V4 – OPTIMIZE

- *Instructions appear in original order and subject to hardware read after write penalties*

```
PACK   344(5,13),0(9,2)
AP     344(5,13),51(2,10)
ZAP    344(5,13),344(5,13)
UNPK   0(9,2),344(5,13)
PACK   344(5,13),16(9,2)
AP     344(5,13),51(2,10)
ZAP    344(5,13),344(5,13)
UNPK   16(9,2),344(5,13)
PACK   344(5,13),32(9,2)
AP     344(5,13),51(2,10)
ZAP    344(5,13),344(5,13)
UNPK   32(9,2),344(5,13)
```

COBOL V5 - at OPT(2) low level instruction scheduling is performed to reduce data dependencies, avoid hardware penalties and to best take advantage of the micro-architecture.

V5 – OPT(2)

- *Independent operations are grouped to reduce read after write hardware penalties*
- *ARCH(8)*

```
PACK   352(5,R13),152(9,R8)
PACK   344(5,R13),168(9,R8)
PACK   336(5,R13),184(9,R8)
AP     352(5,R13),416(2,R3)
AP     344(5,R13),416(2,R3)
AP     336(5,R13),416(2,R3)
ZAP    352(5,R13),352(5,R13)
ZAP    344(5,R13),344(5,R13)
ZAP    336(5,R13),336(5,R13)
UNPK   152(9,R8),352(5,R13)
UNPK   168(9,R8),344(5,R13)
UNPK   184(9,R8),336(5,R13)
```

Performance Comparison

Timing – (100 million in a loop)

```
V5 : 2.35 cpu seconds
V4 : 2.50 cpu seconds
```

Feature Highlights

- Provide source and binary compatibility
- Most correct COBOL programs will compile and execute without changes and produce the same results
- “Old” and “new” code can be mixed within an application and communicate with static or dynamic calls
- Removed some very old language extensions and options
    - Millennium Language Extensions
    - Label Declaratives
    - Non-reentrant programs above 16MB line (NORENT + RMODE(ANY))
    - OS/VS COBOL Inter-operation
    - AMODE 24 (only dynamic calls supported)
        - APAR PM93583 added support for static CALL to AMODE 24 programs
        - Also changed mod level: 5.1.**1**
    - XMLPARSE(COMPAT)

- Simplifying programming
- Raised the total size of all Working-storage or Local-storage section data items to 2GB (from 128MB).
- Raised the maximum size of an individual data item to 999,999,999 bytes (From 128MB).
- Added new built in functions to improve programmability of UTF-8 applications

- Support Application Modernization
- XML enhancements
- Support for UNBOUNDED tables and groups
- Support Java 7

- Support Latest Middleware (CICS, DB2, IMS)
- Support EXEC SQLIMS statements in COBOL programs (IMS V13)

System Administration Considerations

- Provides support for z/OS System Management Facilities (SMF) records
  - Provide full support for sub-capacity pricing
  - Reduce administration overhead
  - Requires SCRT V21.2.0 (GA Apr. 2013)

- Compiler resides in PDSE
  - Generates “GOFF” (Generalized Object File Format)
  - Object features require executable to be a Program Object, PDSE only
    - Load libraries must be PDSE datasets

- Requires more utility data sets
  - SYSUT8-SYSUT15
  - SYSMDECK

- Requires more storage than version 4.2

– Recommend a minimum region size of 200M

Tools Ecosystem Support

IBM

ca technologies

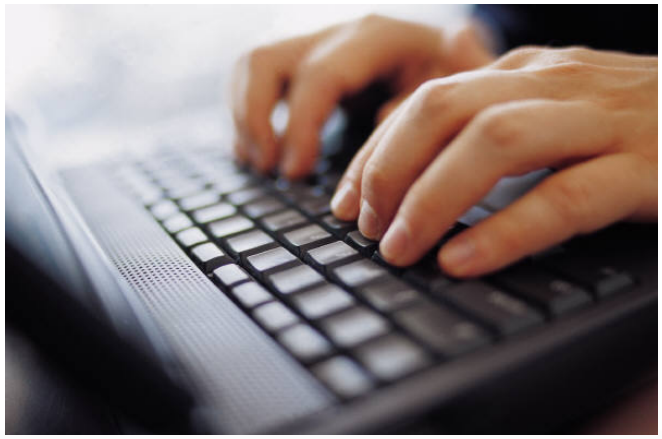
INNOVATION® DATA PROCESSING

ASG® Software Solutions

Compuware
THE TECHNOLOGY PERFORMANCE COMPANY

MACRO 4
A Division of UNICOM Global

bmcsoftware



Best practices

- Recompile parts that are changed and performance “hot spots”
  - –Leverage advanced optimizations and z/Architecture exploitation capabilities in Enterprise COBOL V5
- Take advantage of new compiler features to modernize business critical applications
  - –“Rip and Replace” is expensive and risky
  - –Modernization promotes reuse and delivery of new solution at lower cost, lower risk, and shorter delivery time
- Leverage modern development tools/solutions to improve developer productivity and speed up delivery of new enhancements
  - –IBM Integrated Solution for System z Development
  - –IBM Continuous Integration Solution for System z

Developer Trial

- Zero cost evaluation license for 90 days
  - Does not initiate Single Version Charging (SVC)
- Assess the value that could be gained from upgrading to Enterprise COBOL V5.1
- Offer same functionalities as Enterprise COBOL for z/OS V5.1
  - Same pre-requisites (e.g. runs on z/OS V1.13 and z/OS V2.1…)
  - Code compiled with Enterprise COBOL Developer Trial cannot be used for production
- Available as standard offering through ShopzSeries on Oct 4, 2013
- Contact your IBM representative for ordering assistance

http://www-03.ibm.com/software/products/ph/en/enterprise-cobol-developer-trial-for-zos

Agenda

- Enterprise COBOL for z/OS v5.1
- z/OS XL C/C++ v2.1
- Enterprise PL/I for z/OS v4.4

IBM z/OS 2.1 XL C/C++

- Optionally priced feature of z/OS
  - Enables development of high performing business applications, system programs and low level C applications
- IBM has been delivering leading edge C/C++ compilers on z/OS for over 20 years
  - Every release sets new standard for performance
  - Includes advanced optimization technology originally designed for HPC applications, and innovations to improve programmer productivity
  - Improves support for C and C++ language standards
- Provides system programming capabilities with Metal C option
  - Allows developers to use C syntax to develop system programs and low level free standing applications on z/OS without coding in HLASM
    - Significantly shortens the learning curve
    - Leverage advanced optimization technology to generate high performance optimized code

Advanced Optimization Technology

z/OS XL C/C++ compiler offers:

- 5 optimization levels, and additional options that allow you to tailor how to optimize your applications
- Code generation and tuning for specific z hardware architecture(s)
- Whole program optimization with IPA
  - Interprocedural Analysis (IPA) optimizer
- Profile-directed feedback (PDF) optimization
- Directives and source-level intrinsic functions that provide direct access to System z hardware.

New C & C++ Language Features

- C++11 Standard
  - RValue reference for core language, Const Expression, Scoped Enums, Explicit Conversion Operator, Right Angle Brackets, Generalized Constant Expressions, Default and Deleted Functions
- C11 Standard
  - Static assertions, complex type initialization, Noreturn attribute, anonymous structures, Generic Type Generics
- GNU C/C++ language extensions & compatibility
  - __builtin_expect(x,0)
  - propagation of attributes to function template instantiations
    - For example, __attribute__ ((always_inline)) and __attribute__ ((noinline)) now work on template functions
  - zero initialization
    - Objects with an initializer of () and an implicitly defined default constructor will be zero-initialized before the default constructor is called
  - improved diagnostics for invalid template template argument
- OpenMP API v3.1 Specification (under 64 bit mode and run in USS only) 

  - Industry-standard API designed to create portable C/C++ applications that exploit shared-memory parallelism.
  - Consists of a collection of compiler directives and library routines
  - Users can create or migrate parallel applications to take advantage of the multicore design of modern System z processors.
  - No support for Metal C

OpenMP

```
int main()
{
  int i;
#pragma omp parallel for
  for (i=0; i<N; i++)
  {
    arrA[i] = arrB[i];
    ...
  }
}
```

Compiler transforms user code

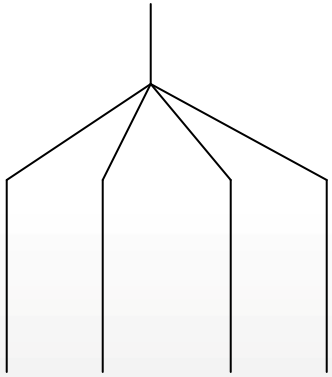
```
int main()
{
  @_xlsmpEntry1 = _xlsmpParSelf();
  @_xlsmpEntry0 = @_xlsmpEntry1;
  _xlsmpParallelDoSetup_TPO(...,
    &main@OL@1,...);
  ...
}

void main@OL@1(...,@LBnd29,
  @UBnd30)
{
  @CIV1 = 0;
  do {
    ...
    @CIV1 = @CIV1 + 1;
  } while (@CIV1<@UBnd30-@LBnd29);
}
```

OpenMP runtime library

Link in the OpenMP library

OpenMP V3.1 supports task parallelism
•Allows independent tasks to execute in parallel

- *loop without known loop counts (e.g. while loop)*
- *divide and conquer algorithm (e.g. recursive calls)*
- *tree traversal*



Create Threads for execution

Debug Improvements

- Automonitor support with Debug Tool

- Debug optimized code
  - compiler creates different levels of snapshots of objects to make the program state available to the debugging session
  - When stopped at snapshot points, the debugger should be able to retrieve the correct value of variables
  - The granularity of the snapshot points is controlled by the DEBUG(LEVEL(n)) n=2,5,8

- Debug inlined functions
  - The debugger can now show values of parameters and locals of an inlined function

z/Architecture Exploitation & Advanced Optimization

- Exploit zEC12 and zBC12 processors
  - ✓ New ARCH(10) functions ; Defaults to ARCH(7)
  - ✓ Built-ins to exploit Transactional Memory (HTM)

- Improved application performance[1]
  - ✓ Compute-intensive integer benchmarks improved
    - ✓12 % (64 bit),
    - ✓ 6% (31 bit)
  - ✓ Compute-intensive floating-point benchmarks improved
    - ✓17% (64 bit),
    - ✓4% (31 bit)

1Results are based on a compute-intensive integer and floating point benchmark suites compiled with z/OS C/C++ V1R13 executing on a System zEnterprise 196 server. Performance gains from other applications may vary

HTM with IBM XL C/C++

Enabled in both z/OS V2.1 XL C/C++ and z/OS XL C/C++ V1.13 Sept. 2012 PTF

Complete set of built-ins

Enable exploitation of new Transactional Execution Facility.

Provides function to:

- Start/End/Abort transactions
- Diagnose transaction failures
- Detect transaction state (e.g. depth)

```
CASE ACTION_UPDATE_TABLES:
//..SOME CODE
  TM_BEGIN(MYID);
  …
  TM_END();
BREAK;
```

tm_begin(MYID) –start a transaction

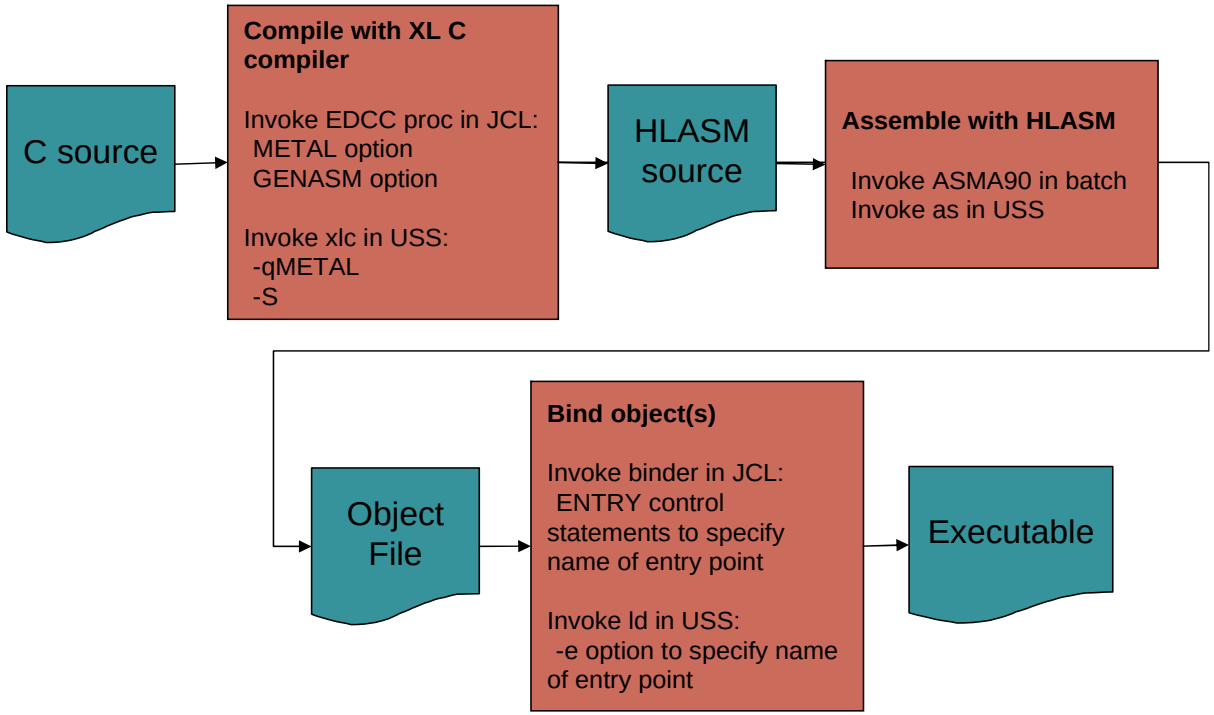
THREAD_MUTEX_LOCK –acquire a lock

tm_end(MYID) –end a transaction

THREAD_MUTEX_UNLOCK –release a lock

- *Transaction execution instructions*
  - long __TM_simple_begin()
  - long __TM_begin(void* const TM_buff)
  - long __TM_end();
  - void __TM_non_transactional_store(void* const addr , long long const value);
  - long __TM_nesting_depth(void* const TM_buff);
- *Transaction failure diagnostic*
  - long __TM_is_user_abort(void* const TM_buff);
  - long __TM_is_named_user_abort(void* const TM_buff, unsigned char* code);
  - long __TM_is_illegal(void* const TM_buff);
  - long __TM_is_footprint_exceeded(void* const TM_buff);
  - long __TM_is_nested_too_deep(void* const TM_buff); z
  - long __TM_is_conflict(void* const TM_buff);
  - long __TM_is_failure_persistent(long const result);
  - long __TM_failure_code();

Metal C



- A new mode of code generation in the z/OS XL C/C++ compiler
  - Leverages Advanced Optimization Technology in z/OS XL C/C++
  - Provides a subset of C library functions.
- Generates optimized HLASM source code that is Language Environment independent
  - Interoperates with existing assembler programs.
  - Offers users the ability to embed assembler statements.

Programming in Assembler vs Metal C on System z

```
         USING CTXT,R9                  ESTABLISH BASE FOR CTXT
          CLC   CTXTACRN,=CL4'CTXT'      A CTXT?
          BNE   RETURN                   SOMETHING IS WRONG IF NOT
          ICM   R3,B'1111',CTXTTXPN      A MINOR LINE?
          BNZ   RETURN                   YES. NO GO
          ICM   R3,B'1111',CTXTTXPJ      PICK UP TEXT STRUCTURE POINTER
          BZ    RETURN                   SHOULD NEVER BE THE CASE
          USING CTXTATTR,R3              ESTABLISH BASE FOR CTXTATTR
          SPACE 1
          CLC   IEE362A,CTXTTMSG         IEE362A?
          BNE   RETURN                   DO NOTHING IF NOT
          CLC   SMFDSN,CTXTTMSG+L'SKIP   DSNAME OK?
          BNE   RETURN                   DO NOTHING IF NOT
          MVC   WEEK,=CL4'EVEN'          ASSUME EVEN
          DP    QUO1,=PL1'2'             DETERMINE EVEN/ODD
          CP    REM2,=PL1'0'             EVEN?
          BE    NOCHANGE
          MVC   WEEK,=CL4'ODD'           NO. THEN IS ODD
          SPACE 1
NOCHANGE  DS    0H
          MVC   MGCRTEXT(L'CMDTEXT),CMDTEXT COPY COMMAND INTO BUFFER
          MVC   MGCRTEXT+L'CMDTEXT-8(1),CTXTTMSG+L'IEE362A GET SUFFIX
          MVC   MGCRTEXT+L'CMDTEXT-4(4),WEEK SAY EVEN OR ODD
          LA    R0,MGCRTEXT-MGCRPL+L'CMDTEXT FIGURE OUT BUFFER LENGTH
          STC   R0,MGCRLGTH              INSERT EXACT BUFFER LENGTH
          SPACE 1
          SLR   R0,R0                    CLEAR R0 FOR MGCR
          MGCR  MGCRPL                   ISSUE START COMMAND
```

HLASM

Platform: IBM System z (z/OS)

Problem: *System Programming -* user exit to handle the SMF dataset message (IEE362A) when a particular SMF dataset is full

```
void XIEE362A(struct ctxt * ctxtp) {
  struct ctxtattr * ctxtattrp;
  struct mgcrpl wmgcrpl;

  if (!memcmp(ctxtp->ctxtacrn,"CTXT",4)) return;
  if (ctxtp->ctxttxpn) return;
  ctxtattrp = ctxtp->ctxttxpj;
  if (ctxtattrp==NULL) return;
  if (!memcmp(ctxtattrp->ctxttmsg, "IEE362A SMF
     ENTER DUMP FOR ", 27)) return;
  if (!memcmp(ctxtattrp->ctxttmsg+27, "SYS1.MAN",
     8)) return;
  memcpy(wmgcrpl.mgcrtext, "START CLRSMF,N=x",
     16);
  memcpy(wmgcrpl.mgcrtext+15, ctxtattrp-
     >ctxttmsg+35, 1);
  wmgcrpl.mgcrlgth = 16;
  __asm("  XR 0,0\n MGCR %0"::"m"(wmgcrpl));
}
```

Metal C

Metal C optimizes for newer target architecture

No code modification required

/bin/xlc –O3 -qmetal -S a.c –qarch=9

```
@1L3     DS     0H              000010
         LR     1,2             000010
         AHI    2,1             000010
         LLC    0,1(1,15)       000010
         LTR    0,0             000010
         BRE    @1L4            000010
         LLC    0,2(1,15)       000010
         ALHSIK 2,1,2           000010
```

Optimized Assembly z196

/bin/xlc –O3 -qmetal -S a.c –qarch=10

```
@1L3     DS     0H              000010
         LR     1,2             000010
         AHI    2,1             000010
         LLC    0,1(1,15)       000010
         CIJE   0,0,@1L4        000010
         ALHSIK 2,1,2           000010
```

Optimized Assembly (zEC12)

Agenda

- Enterprise COBOL for z/OS v5.1
- z/OS XL C/C++ v2.1
- Enterprise PL/I for z/OS v4.4

Enterprise PL/I

- Strategic Programming Language
  - Significant use in business applications but also in some scientific and engineering applications
- Advanced optimization technology
  - Shares optimizing back-end technology with z/OS XL C/C++
    - Enables timely delivery of leading edge optimization and hardware exploitation to PL/I customers
- Time proven
  - First Enterprise PL/I product released in 2001 (Enterprise PL/I for z/OS and OS/390 v3.1)
  - Latest release of Enterprise PL/I for z/OS (v4.4) is based on same architecture
    - Provides easy migration
- Shipped new release every year since 1999
  - Improved optimization technology, z/Architecture exploitation, usability, middleware support, and application modernization features.
  - Addressed customer requirements

Enterprise PL/I V4.4

- Exploits zEC12 and zBC12 processors
  - Exploits new Decimal-Floating-Point Zoned-Conversion Facility
    - PICTURE to FIXED BIN conversions were 40% faster
    - PICTURE to FLOAT DEC conversions were 4X faster
  - 2% performance improvement over v4.3 (4.3 provides 5% performance improvements over v4.2)
  - Improved code for PIC to FIXED BIN conversions
  - Improved code for UTF-16 data

- Improves Middleware support
  - Support latest CICS, DB2 and IMS
  - Improved diagnostic messages for SQL preprocessor
  - New built-in functions for Base64 encoding and decoding
  - New XML cleaning and normalization functions reduce convertor size
    - Allows more to run in the same address space.
  - New support for sparse arrays reduces data transfer to XML convertors
    - Improves throughput time

- Increases programmer productivity and application modernization
  - UTF-16 PICTURE support
  - Improved compile time when LIST option is on by 4X
  - Compiler message to recommend code changes that will improve both compile time and run time

27

Exploiting Decimal-Floating-Point Zoned-Conversion Facility

Example: Given this code to convert PICTURE to DFP

```
*process float(dfp);

 pic2dfp: proc( ein, aus ) options(nodescriptor);

   dcl ein(0:100_000) pic'(9)9' connected;
   dcl aus(0:hbound(ein)) float dec(16)
connected;
   dcl jx  fixed bin(31);

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

With ARCH(9), the heart of the loop consists of these 17 instructions

```
 0060  F248  D0F0  F000      PACK     #pd580_1(5,r13,240),_shadow4(9,r15,0)
 0066  C050  0000  0035      LARL     r5,F'53'
 006C  D204  D0F8  D0F0      MVC      #pd581_1(5,r13,248),#pd580_1(r13,240)
 0072  41F0  F009            LA       r15,#AMNESIA(,r15,9)
 0076  D100  D0FC  500C      MVN
#pd581_1(1,r13,252),+CONSTANT_AREA(r5,12)
 007C  D204  D0E0  D0F8      MVC      _temp2(5,r13,224),#pd581_1(r13,248)
 0082  F874  D100  2000      ZAP      #pd586_1(8,r13,256),_shadow3(5,r2,0)
 0088  D207  D0E8  D100      MVC      _temp1(8,r13,232),#pd586_1(r13,256)
 008E  5800  4000            L        r0,_shadow2(,r4,0)
 0092  5850  4004            L        r5,_shadow2(,r4,4)
 0096  EB00  0020  000D                SLLG     r0,r0,32
 009C  1605                  OR       r0,r5
 009E  B3F3  0000            CDSTR    f0,r0
 00A2  EB00  0020  000C      SRLG     r0,r0,32
 00A8  B914  0011            LGFR     r1,r1
 00AC  B3F6  0001            IEDTR    f0,f0,r1
 00B0  6000  E000            STD      f0,_shadow1(,r14,0)
```

With ARCH(10), it consists of just these 8 instructions – runs 4 times 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  6001  E000            STD      f0,_shadow1(r1,r14,0)
```

For more information

- Enterprise COBOL for z/OS Product information
  http://www-01.ibm.com/software/awdtools/cobol/zos/
- Enterprise PL/I for z/OS Product information
  http://www-03.ibm.com/software/products/us/en/plicompfami/
- z/OS XL C/C++ Product information
  http://www-03.ibm.com/software/products/us/en/czos
- Rational Enterprise Modernization Products
  http://www-03.ibm.com/software/products/us/en/category/SWY00
- z/OS Problem Determination Tools
  http://www-01.ibm.com/software/awdtools/deployment/
- RFE Community
  COBOL Compilers http://www.ibm.com/developerworks/rfe/?PROD_ID=698
  PL/I Compilers http://www.ibm.com/developerworks/rfe/?PROD_ID=699
  C/C++ Compilers http://www.ibm.com/developerworks/rfe/?PROD_ID=700
- Compilers and Application Tools user communities
  Rational Café https://www.ibm.com/developerworks/rational/community/cafe/
- COBOL moderated screen cast with Kevin Stoodley & James Governor
  http://www.youtube.com/watch?v=JLMqkuou2-s


QUESTIONS


THANK
YOU

Compiling at OPT(0), OPT(2), & OPT(3)

- Intermediate code generated from C/C++ front-end passes directly to the back-end for optimization and object code creation

- OPT(0) – Performs basic optimizations such as redundant code elimination, constant folding…

- OPT(2) – Performs more optimizations on loops, removes unnecessary code constructs and redundant computations

- OPT(3) – An intensified version of OPT(2**)**.
  - ✓ Performs additional low-level transformations and optimizations encompassing larger program regions

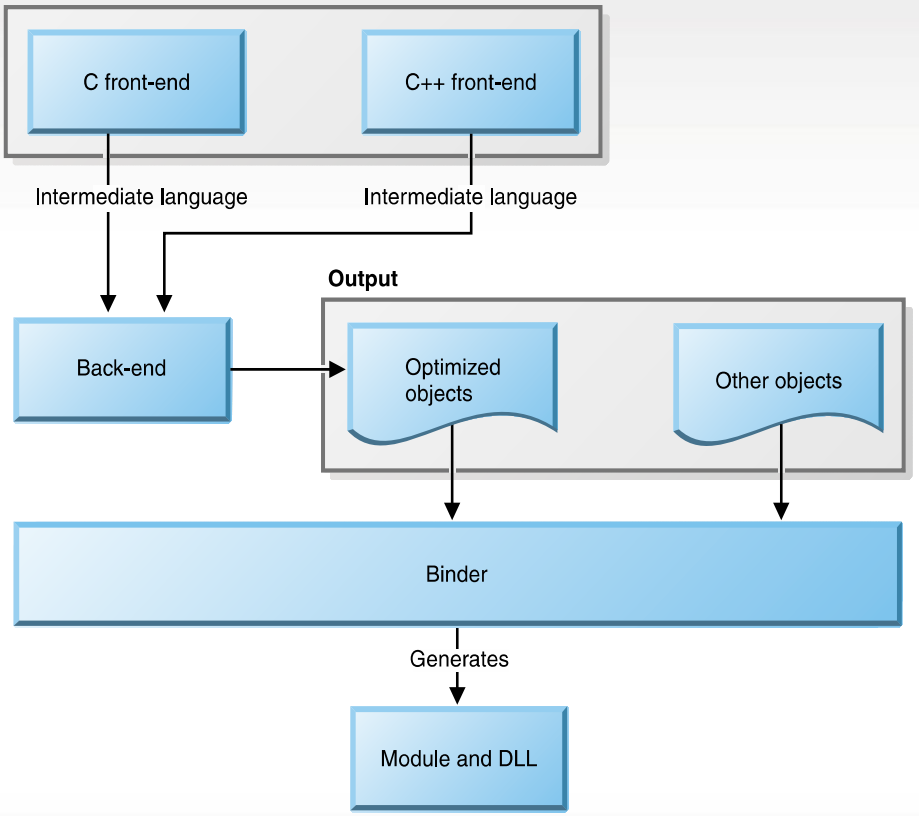


Figure 1. Compiling at NOOPT or OPT(2) or OPT(3)

Compiling at OPT(4), & OPT(5)

- Intermediate code generated from compiler frontend passes to IPA for more advanced optimization. Optimized intermediate code is then passed to the back-end for further optimization and object code creation.
- OPT(4) builds on OPT(3) by invoking inter-procedural analysis (IPA).
  - Optimize the entire application as a unit.
- OPT(5) adds deeper whole-program analysis and more aggressive optimizations.
- Profile-directed feedback (PDF), iteratively refines a profile of how often branches are taken and blocks of code are executed in an application