#SHAREorg

SHARE
Technology · Connections · Results

z/OS XL C/C++ V2R1
Enterprise PL/I 4.4

Visda Vokhshoori

IBM Corporation

Session 13789

Insert Custom Session QR if Desired.

SHARE in Boston

AGENDA

- C/C++ New features
- Performance
- In Depth
  - Transaction Execution
  - Debugging Optimized Code
  - 64-bit performance improvements
- PL/I 4.4 Features
  - Decimal-Floating Point to Zoned Facility

New features in z/OS XL C/C++ V2R1

- [Default] Compiler default is now arch=7, z9 machine
- [C1X] New language level EXTC1X builds upon EXTC99
- [C1X] Support for general infinity and NaN initialization for complex types
- [C1X] Generic Type Generics
- [C1X] _Noreturn keyword
- [C1X] static_assert
- [C++11] explicit operator
- [C++11] Name mangeling
- [C++11] Scoped enums
- [C++11] Right angle brackets
- [C++11] Rvalue reference
- [C++11] Generalized Constant Expression
- [C/C++] Include Master Header
- [MetalC] SYSSTATE compiler option
- [MetalC] User nominated main function
- [MetalC] Mixed addressing mode with IPA
- [Builtins] Decimal-Floating-Point Zoned Conversion
- [Builtins] Transactional Memory
- [Builtins] Packed Decimal builtin functions
- [Optimization] __builtin_expect
- [Optimization] OpenMP 3.1 specification
- [Optimization] [NO]THREADED option
- [Optimization] Attribute always_inline
- [Debug] Debugging optimized code
- [Debug] Debugging inlined procedures

New features in z/OS XL C/C++ V2R1

- **[Default] Compiler default is now arch=7, z9 machine**
- [C1X] New language level EXTC1X builds upon EXTC99
- [C1X] Support for general infinity and NaN initialization for comple
- [C1X] Generic Type Generics
- [C1X] _Noreturn keyword
- [C1X] static_assert
- [C++11] explicit operator
- [C++11] Name mangeling
- [C++11] Scoped enums
- [C++11] Right angle brackets
- [C++11] Rvalue reference
- [C++11] Generalized Constant Expression
- [C/C++] Include Master Header
- [MetalC] SYSSTATE compiler option
- [MetalC] User nominated main function
- [MetalC] Mixed addressing mode with IPA
- [Builtins] Decimal-Floating-Point Zoned Conversion
- [Builtins] Transactional Memory
- [Builtins] Packed Decimal builtin functions
- [Optimization] __builtin_expect
- [Optimization] OpenMP 3.1 specification
- [Optimization] [NO]THREADED option
- [Optimization] Attribute always_inline
- [Debug] Debugging optimized code
- [Debug] Debugging inlined procedures

```
#include <stdio.h>
int main() {
printf("ARCH LEVEL %d\n", __ARCH__);
return 0;
}
xlc arch_check.c          > ARCH LEVEL 7
xlc -qtarget=zosv1r13 arch_check.c >ARCH LEVEL 5
```

New features in z/OS XL C/C++ V2R1

- [Default] Compiler default is now arch=7, z9 machine
- **[C1X] New language level EXTC1X builds upon EXTC99**
- [C1X] Support for general infinity and NaN initialization for complex types
- [C1X] Generic Type Generics
- [C1X] _Noreturn keyword
- [C1X] static_assert
- [C++11] explicit operator
- [C++11] Name mangeling
- [C++11] Scoped enums
- [C++11] Right angle brackets
- [C++11] Rvalue reference
- [C++11] Generalized Constant Expression
- [C/C++] Include Master Header
- [MetalC] SYSSTATE compiler option
- [MetalC] User nominated main function
- [MetalC] Mixed addressing mode with IPA
- [Builtins] Decimal-Floating-Point Zoned Conversion
- [Builtins] Transactional Memory
- [Builtins] Packed Decimal builtin functions
- [Optimization] __builtin_expect
- [Optimization] OpenMP 3.1 specification
- [Optimization] [NO]THREADED option
- [Optimization] Attribute always_inline
- [Debug] Debugging optimized code
- [Debug] Debugging inlined procedures

```
struct s{
  struct {
    int a;
   };
 }S;
int main() {
  s.a = 55;
  return s.a;
}
xlc -qlanglvl=extc1x anonymous_struct.c
./a.out
echo $?
55
```

New features in z/OS XL C/C++ V2R1

- [Default] Compiler default is now arch=7, z9 machine
- [C1X] New language level EXTC1X builds upon EXTC99
- **[C1X] Support for general infinity and NaN initialization for complex types**
- [C1X] Generic Type Generics
- [C1X] _Noreturn keyword
- [C1X] static_assert
- [C++11] explicit operator
- [C++11] Name mangeling
- [C++11] Scoped enums
- [C++11] Right angle brackets
- [C++11] Rvalue reference
- [C++11] Generalized Constant Expression
- [C/C++] Include Master Header
- [MetalC] SYSSTATE compiler option
- [MetalC] User nominated main function
- [MetalC] Mixed addressing mode with IPA
- [Builtins] Decimal-Floating-Point Zoned Conversion
- [Builtins] Transactional Memory
- [Builtins] Packed Decimal builtin functions
- [Optimization] __builtin_expect
- [Optimization] OpenMP 3.1 specification
- [Optimization] [NO]THREADED option
- [Optimization] Attribute always_inline
- [Debug] Debugging optimized code
- [Debug] Debugging inlined procedures

```
#include <stdio.h>
double _Complex big_value=CMPLX(1.0/0.0, 1.0/0.0);
int main() {
 float _Complex nan_value=CMPLXF(9.5, 0.0/0.0);
 printf("big value: %e+%e*i\n",__real__(big_value),
        __imag__(big_value);
 printf("nan value: %e+%e*i\n",__real__(nan_value),
       __imag__(nan_value);
return 0;
}
xlc -qfloat=ieee -qlanglv=extc1x complexvalue.c
./a.out
big value: INF+INF*i
nan value: 9.5+NANQ(1)*i;
```

New features in z/OS XL C/C++ V2R1

- [Default] Compiler default is now arch=7, z9 machine
- [C1X] New language level EXTC1X builds upon EXTC99
- [C1X] Support for general infinity and NaN initialization for compl
- **[C1X] Generic Type Generics**
- [C1X] _Noreturn keyword
- [C1X] static_assert
- [C++11] explicit operator
- [C++11] Name mangeling
- [C++11] Scoped enums
- [C++11] Right angle brackets
- [C++11] Rvalue reference
- [C++11] Generalized Constant Expression
- [C/C++] Include Master Header
- [MetalC] SYSSTATE compiler option
- [MetalC] User nominated main function
- [MetalC] Mixed addressing mode with IPA
- [Builtins] Decimal-Floating-Point Zoned Conversion
- [Builtins] Transactional Memory
- [Builtins] Packed Decimal builtin functions
- [Optimization] __builtin_expect
- [Optimization] OpenMP 3.1 specification
- [Optimization] [NO]THREADED option
- [Optimization] Attribute always_inline
- [Debug] Debugging optimized code
- [Debug] Debugging inlined procedures

```
#define cbrt(x) _Generic ((X), \
long double: cbrtl, \
default: cbrt, \
float: cbrtf \
)(X)
Note: cbrtl, cbrt, and cbrtf should be defined.
```

New features in z/OS XL C/C++ V2R1

- [Default] Compiler default is now arch=7, z9 machine
- [C1X] New language level EXTC1X builds upon EXTC99
- [C1X] Support for general infinity and NaN initialization for complex types
- [C1X] Generic Type Generics
- **[C1X] _Noreturn keyword**
- [C1X] static_assert
- [C++11] explicit operator
- [C++11] Name mangeling
- [C++11] Scoped enums
- [C++11] Right angle brackets
- [C++11] Rvalue reference
- [C++11] Generalized Constant Expression
- [C/C++] Include Master Header
- [MetalC] SYSSTATE compiler option
- [MetalC] User nominated main function
- [MetalC] Mixed addressing mode with IPA
- [Builtins] Decimal-Floating-Point Zoned Conversion
- [Builtins] Transactional Memory
- [Builtins] Packed Decimal builtin functions
- [Optimization] __builtin_expect
- [Optimization] OpenMP 3.1 specification
- [Optimization] [NO]THREADED option
- [Optimization] Attribute always_inline
- [Debug] Debugging optimized code
- [Debug] Debugging inlined procedures

```
void _Noreturn end_program(char err_type) {
  if (err_type=='E')
    exit(16);
  else
    exit(0);
}
int main(int argc, char* argv[]) {
  if(argc<5) {
    end_program('E');
    //any code here will never get run
   }
   return 0;
}
xlc -qlanglvl=extc1x aborter.c
./a.out
echo $?
16
```

New features in z/OS XL C/C++ V2R1

- [Default] Compiler default is now arch=7, z9 machine
- [C1X] New language level EXTC1X builds upon EXTC99
- [C1X] Support for general infinity and NaN initialization for complex types
- [C1X] Generic Type Generics
- [C1X] _Noreturn keyword
- **[C1X] static_assert**
- [C++11] explicit operator
- [C++11] Name mangeling
- [C++11] Scoped enums
- [C++11] Right angle brackets
- [C++11] Rvalue reference
- [C++11] Generalized Constant Expression
- [C/C++] Include Master Header
- [MetalC] SYSSTATE compiler option
- [MetalC] User nominated main function
- [MetalC] Mixed addressing mode with IPA
- [Builtins] Decimal-Floating-Point Zoned Conversio
- [Builtins] Transactional Memory
- [Builtins] Packed Decimal builtin functions
- [Optimization] __builtin_expect
- [Optimization] OpenMP 3.1 specification
- [Optimization] [NO]THREADED option
- [Optimization] Attribute always_inline
- [Debug] Debugging optimized code

```
#include <assert.h>
#include <stdlib.h>
static_assert(sizeof(long)==4,"Not in 31bit mode");
int main() {
 long* buffer=(long*)malloc(10*4);
 for(int i=0; i<10; ++i)
    buffer[i] = i;
return (int)buffer[8]+47;
}
xlc -qlangl=extc1x -q64 longBuffer.c
ERROR CCN3865 ./longBuffer.c: 3 Not in 31bit mode
```

New features in z/OS XL C/C++ V2R1

- [Default] Compiler default is now arch=7, z9 machine
- [C1X] New language level EXTC1X builds upon EXTC99
- [C1X] Support for general infinity and NaN initialization for complex types
- [C1X] Generic Type Generics
- [C1X] _Noreturn keyword
- [C1X] static_assert
- **[C++11] explicit operator**
- [C++11] Name mangeling
- [C++11] Scoped enums
- [C++11] Right angle brackets
- [C++11] Rvalue reference
- [C++11] Generalized Constant Expression
- [C/C++] Include Master Header
- [MetalC] SYSSTATE compiler option
- [MetalC] User nominated main function
- [MetalC] Mixed addressing mode with IPA
- [Builtins] Decimal-Floating-Point Zoned Conversion
- [Builtins] Transactional Memory
- [Builtins] Packed Decimal builtin functions
- [Optimization] __builtin_expect
- [Optimization] OpenMP 3.1 specification
- [Optimization] [NO]THREADED option
- [Optimization] Attribute always_inline
- [Debug] Debugging optimized code

```
#include <iostream>
template <class T> struct Ptr {
Ptr():rawptr_(0) {}
Ptr(T* Ptr):rawptr_(T) {}
explicit operator bool() const {return rawptr_ != 0; }
T* rawptr_;
};
int main() {
 int var1, var2;
 Ptr<int> ptr1, ptr2(&var2);
 ptr1 = &var1;  explicit Ptr(T* Ptr):rawptr_(T); - warning
 if (ptr1)  //explicit bool operator provided - good.
   return 66;
cout << "ptr1+ptr2= " << (ptr1+ptr2) << endl; //warning
return 0;
}
xlC -qlanglvl=explicitconversionoperators a.C
./a.C, line 13.40: CCN5218 (S) The call doesn't match any
parameter list for operator+.
./a.C, line 13.40: CCN6283 (I) builtin operator+(int, int) is
not a viable candidate.
```

New features in z/OS XL C/C++ V2R1

- [Default] Compiler default is now arch=7, z9 machine
- [C1X] New language level EXTC1X builds upon EXTC99
- [C1X] Support for general infinity and NaN initialization for complex types
- [C1X] Generic Type Generics
- [C1X] _Noreturn keyword
- [C1X] static_assert
- [C++11] explicit operator
- **[C++11] Name mangeling**
- [C++11] Scoped enums
- [C++11] Right angle brackets
- [C++11] Rvalue reference
- [C++11] Generalized Constant Expre
- [C/C++] Include Master Header
- [MetalC] SYSSTATE compiler option
- [MetalC] User nominated main function
- [MetalC] Mixed addressing mode with IPA
- [Builtins] Decimal-Floating-Point Zoned C
- [Builtins] Transactional Memory
- [Builtins] Packed Decimal builtin functions
- [Optimization] __builtin_expect
- [Optimization] OpenMP 3.1 specification
- [Optimization] [NO]THREADED option
- [Optimization] Attribute always_inline
- [Debug] Debugging optimized code
- [Debug] Debugging inlined procedures

```
template<typename Element> struct Iterator {
  Iterator() {}
  friend Iterator<Element>& operator+(Iterator<Element> it, long d) {
   it+=d;
   return it;
 }
};
int main()
{
  Iterator<int> iter;
  Iterator<const int> c_iter;
  c_iter = c_iter+10;
  iter = iter+10;
  return 0;
}
xlC -qnamemangling=zosv2r1_ansi -c a.C //compiles OK
xlC -qnamemangling=zosv1r2_ansi -c a.C //fails
./a.C, line 14.26 CCN5704 (S) The definitions of "Iterator<const int>
operator+(Iterator<const int>, long) and "Iterator<int>
operator+(Iterator<int>, long)" have the same linkage signature.
"__pl__F8IteratorXTi_l"
```

New features in z/OS XL C/C++ V2R1

- [Default] Compiler default is now arch=7, z9 machine
- [C1X] New language level EXTC1X builds upon EXTC99
- [C1X] Support for general infinity and NaN initialization for complex types
- [C1X] Generic Type Generics
- [C1X] _Noreturn keyword
- [C1X] static_assert
- [C++11] explicit operator
- [C++11] Name mangeling
- **[C++11] Scoped enums**
- [C++11] Right angle brackets
- [C++11] Rvalue reference
- [C++11] Generalized Constant Expression
- [C/C++] Include Master Header
- [MetalC] SYSSTATE compiler option
- [MetalC] User nominated main function
- [MetalC] Mixed addressing mode with IPA
- [Builtins] Decimal-Floating-Point Zoned Convers
- [Builtins] Transactional Memory
- [Builtins] Packed Decimal builtin functions
- [Optimization] __builtin_expect
- [Optimization] OpenMP 3.1 specification
- [Optimization] [NO]THREADED option
- [Optimization] Attribute always_inline
- [Debug] Debugging optimized code
- [Debug] Debugging inlined procedures

```
enum class Color {orange, green, purple};
void foo(Color){}
int main() {
  foo(Color::green);
  Color color = orange;  //strongly typed enum should be
                         //accompanied with enum name
  return 0;
}
xlC -qlanglv=scopedenum a.C
```

New features in z/OS XL C/C++ V2R1

- [Default] Compiler default is now arch=7, z9 machine
- [C1X] New language level EXTC1X builds upon EXTC99
- [C1X] Support for general infinity and NaN initialization for complex types
- [C1X] Generic Type Generics
- [C1X] _Noreturn keyword
- [C1X] static_assert
- [C++11] explicit operator
- [C++11] Name mangeling
- [C++11] Scoped enums
- **[C++11] Right angle brackets**
- [C++11] Rvalue reference
- [C++11] Generalized Constant Expression
- [C/C++] Include Master Header
- [MetalC] SYSSTATE compiler option
- [MetalC] User nominated main function
- [MetalC] Mixed addressing mode with IPA
- [Builtins] Decimal-Floating-Point Zoned Conversion
- [Builtins] Transactional Memory
- [Builtins] Packed Decimal builtin functions
- [Optimization] __builtin_expect
- [Optimization] OpenMP 3.1 specification
- [Optimization] [NO]THREADED option
- [Optimization] Attribute always_inline
- [Debug] Debugging optimized code
- [Debug] Debugging inlined procedures

```
vector<vector<int>> v;

xlC -qlanglvl=rightanglebracket a.C
```

New features in z/OS XL C/C++ V2R1

- [Default] Compiler default is now arch=7, z9 machine
- [C1X] New language level EXTC1X builds upon EXTC99
- [C1X] Support for general infinity and NaN initialization for complex types
- [C1X] Generic Type Generics
- [C1X] _Noreturn keyword
- [C1X] static_assert
- [C++11] explicit operator
- [C++11] Name mangeling
- [C++11] Scoped enums
- [C++11] Right angle brackets
- **[C++11] Rvalue reference**
- [C/C++] Include Master Header
- [MetalC] SYSSTATE compiler option
- [MetalC] User nominated main function
- [MetalC] Mixed addressing mode with IPA
- [Builtins] Decimal-Floating-Point Zoned Conversion
- [Builtins] Transactional Memory
- [Builtins] Packed Decimal builtin functions
- [Optimization] __builtin_expect
- [Optimization] OpenMP 3.1 specification
- [Optimization] [NO]THREADED option
- [Optimization] Attribute always_inline
- [Debug] Debugging optimized code
- [Debug] Debugging inlined procedures

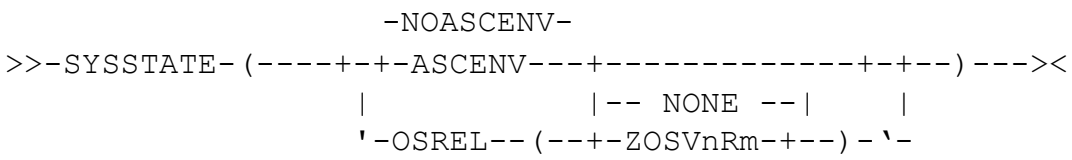
```
string &string::operator=(string &&parm_str);
string a,b,c;
a = b+c;
```

New features in z/OS XL C/C++ V2R1

- [Default] Compiler default is now arch=7, z9 machine
- [C1X] New language level EXTC1X builds upon EXTC99
- [C1X] Support for general infinity and NaN initialization for complex types
- [C1X] Generic Type Generics
- [C1X] _Noreturn keyword
- [C1X] static_assert
- [C++11] explicit operator
- [C++11] Name mangeling
- [C++11] Scoped enums
- [C++11] Right angle brackets
- [C++11] Rvalue reference
- **[C/C++] Include Master Header**
- [MetalC] SYSSTATE compiler option
- [MetalC] User nominated main function
- [MetalC] Mixed addressing mode with IPA
- [Builtins] Decimal-Floating-Point Zoned Conversion
- [Builtins] Transactional Memory
- [Builtins] Packed Decimal builtin functions
- [Optimization] __builtin_expect
- [Optimization] OpenMP 3.1 specification
- [Optimization] [NO]THREADED option
- [Optimization] Attribute always_inline
- [Debug] Debugging optimized code
- [Debug] Debugging inlined procedures

```
xlc t.c -qinclude=stdio.h -qinclude=myhdr.h
```

MetalC – SYSSTATE option



Default is SYSSTATE(NOASCENV,OSREL(NONE))

- OSREL=NONE|ZOSVnRmOSREL provides the value for the OSREL parameter on the SYSSTATE macro generated by the compiler. The value provided must be in the form of ZOSVnRm as described in the "z/OS MVS Programming: Assembler Services Reference". The default is NONE with which no OSREL parameter will appear on the SYSSTATE macro.
- ASCENV | NOASCENVASCENV indicates to the compiler to automatically generate additional SYSSTATE macros with the ASCENV parameter to reflect the ASC mode of the function. The default is NOASCENV with which no ASCENV parameter will appear on the SYSSTATE macro.

MetalC – SYSSTATE option

```
int main()
 {
  int lv=256, addr;

  __asm("  GETMAIN RC,LV=(%1),Key=(%2),LOC=(%3,%4) \n"
        "  ST 1,%0\n"
        :"=m"(addr)
        :"r"(lv),"I"(6),"I"(31),"I"(31)
        :"r1","r15");
  __asm(" FREEMAIN RU,LV=256,A=(1) \n");

  return 55;
}

>xlc -qMETAL -S -qSYSSTATE=ASCENV test.c
   SYSSTATE ASCENV=P     appears before function entry point marker
if function compiled w/ -qARMODE or has __attribute__((armode))
   SYSSTATE ASCENV=AR     appears before function entry point marker
```

MetalC – User Nominated Main

```
#pragma map(main, "ANEWMAIN")
void dosomething(char *);

int main(int argc, char *argv[]) {
  int i;   for (i=1; i<argc; i++) {
    dosomething(argv[i]);
  }
return 0;
}
```

- When a Metal C program is built with the RENT option, it needs a “main” function to anchor the Writable Static Area (WSA) creation process. However a Metal C program may not have a function called “main” as the entry point thus not having the opportunity to be built with the RENT option.
- The entry point name in the generated code will be ANEWMAIN.
- When you link your program, you'll need to tell the binder that the entry point name is ANEWMAIN, such as this:

  /bin/ld -o a.out a.o -e ANEWMAIN

MetalC – Mixed Addressing Mode with IPA

```
xlc -qmetal -q32 -qipa -c 32.c
xlc -qmetal -q64 -qipa -c 64.c
xlc -qmetal -qipa -S 32.o 64.o
```

Metal C applications with AMODE-switching requirements can take advantage of inter-procedural analysis optimization

Built-ins - Decimal Floating Point-Zoned Conversion

- When compiled with the DFP and ARCH(10) options the following new hardware built-in functions are available:

- For Zoned to DFP conversions:

_Decimal128 __cxzt(void* source, unsigned char length, const unsigned char mask);
_Decimal64 __cdzt(void* source, unsigned char length, const unsigned char mask);

- For DFP to Zoned conversions:

int __czxt(_Decimal128 source, void* result, unsigned char length, const unsigned char mask);
int __czdt(_Decimal64 source, void* result, unsigned char length, const unsigned char mask);

Built-ins - Transactional Memory

- Multi-threaded applications can benefit from processors' “opportunistic locking” of memory blocks. This could result in fast lock-free execution where there is no conflict.
- Support for this hardware feature is via a set of intrinsics as well as a set of built-ins for error cause detection
- Reduce overhead of obtaining and orchestrating a lock and improve run-time performance of multi-threaded programs
- 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 functions:**
- 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);
- long __TM_is_conflict(void* const TM_buff);
- long __TM_is_failure_persistent(long const result);
- long __TM_is_failure_address(void* const TM_buff);
- long __TM_failure_code();
- void __TM_abort_assist(unsigned int num_aborts);

Built-ins - Packed Decimal

- C++ is missing support for packed decimal intrinsic type
- Decimal instructions are not normally generated by XL compilers
- Six buitin functions prototyped in builtins.h:
  - Compare Decimal – CP
  - Add Decimal – AP
  - Subtract Decimal – SP
  - Multiply Decimal – MP
  - Divide Decimal – DP
  - Shift and Round Decimal - SRP
- C++ and Metal C users can directly utilize packed-decimal instructions

Optimization - __builtin_expect

```
if(__builtin_expect(x, 0)) {
error();
...
}
```

- Here, we expect that x will be equal 0, and we will not execute the statement for this branch very frequently
- Providing this additional information to the compiler can be exploited for optimization

Optimization - OpenMP 3.1

- A widely used industry standard to construct task parallelism
- Support for 3.1 OpenMP specification has been added
- A new option, SMP, to allow OpenMP parallelization directives to be recognized
- Only supported in 64-bit mode, generated executable must run in USS, thread safe version of standard library must be used inside the parallel regions

Optimization - [NO]THREADED Option

- A new NOTHREADED option for user to assert their application is single-threaded. This will allow for more aggressive optimization and can potentially reduce compile- and run-time performance

```
xlc -qnothreaded single-threaded.c
xlc -qthreaded multi-threaded.c
```

Optimization - __attribute__((always_inline))

- An IBM Extension available at OPT for functions identified as inline
- These functions will be inlined by the compiler
- By passes the compiler heuristic of inlining, puts the user in charge of telling the compiler which functions are important to be inlined

Debugging Optimized Code

- It is hard to debug optimized code because:
  - The debugger doesn't know where to find the value of a variable, i.e. it can be in a register, not in memory
  - The code generated ordering may not match the source code ordering

- The Debug Optimize Code feature:
  - Creates different levels of snapshots of objects at selected source locations
  - Makes the program state available to the debugging session at the selected source locations
  - When stopping at the 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) sub-option:
  - DEBUG(LEVEL(2)): No snapshot points inserted
  - DEBUG(LEVEL(5)): Snapshot points inserted before and after (1) if-endif, (2) function, (3) loop, and (4) the first executable line of a function
  - DEBUG(LEVEL(8)): Snapshot points inserted at every executable statement
  - The line number table will only contain entries for the snapshot points
  - The debugger can only stop at snapshot points when doing source view debugging
  - Applies to DWARF format and O2

  e.g. xlc -Wc,"DEBUG(FORMAT(DWARF),LEVEL(8))" -O2 a.c
  e.g. xlc -Wc,"DEBUG(FORMAT(DWARF))" -O2 -g8 a.c

Debugging Inlined Procedures

- In v1r13, we added debug information for inline procedures
  - Set entry breakpoint for all inline instances of a procedure
  - No debug information is provided for the parameters and local variables of the inline instances
    - *Debugger cannot show the value of these objects*
- V2R1 provides debug information for parameters and local variables of each inline instance of a procedure
- The debugger is able to set show the values of the parameters and locals of an inline instance

Performance - statement

- A suite of CPU intensive C/C++ integer benchmarks compiled with the V2R1 compiler and 31-bit addressing demonstrated more than 6% improved performance compared to the same benchmarks compiled with the V1R13 compiler.

- A suite of CPU intensive C/C++ integer benchmarks compiled with the V2R1 compiler and 64-bit addressing demonstrated more than 11% improved performance compared to the same benchmarks compiled with the V1R13 compiler.

- The performance improvements are based on internal IBM lab measurements. All benchmarks were built using the XPLINK, HGPR, O3, HOT, and IPA(LEVEL(2) with PDF compiler options. The benchmarks compiled with the V1R13 compiler were built using the ARCH(9) TUNE(9) options; the benchmarks compiled with the V2R1 compiler used ARCH(10) TUNE(10). Performance results for specific applications will vary, depending on the source code, the compiler options specified, and other facto

Performance – applied

- By just setting ARCH and TUNE [C/C++ only] to 10 get the best performance running on zEC12
- Utilize the features that enables the following zEC12 facilities
  - Transaction Execution Facility
    - Up to 4X speed up over coarse locking of shared memory via pthread_mutex[1]
  - Decimal-Floating-Point Zoned-Conversion Facility
    - Up to 4X speed up FLOAT DEC to PICTURE conversion [2]
- Improved Performance of LP64 applications
  - Making the type of data the center logic of instruction selection

Transaction Execution Facility

- Facilitates parallel programming, have multiple paths of execution to work together to complete the task the program has to perform
- The instructions between TBEGIN and TEND will execute in isolation, and atomically
- If the transaction aborts, due to conflict, illegal instruction, footprint exceeded, hardware interrupt or other abort conditions, the memory state and register content is rolled back to before transaction
- If the transaction succeeds all the results are committed to memory at the end of the transaction

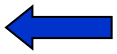
Transaction Execution Facility-Sample

```
      LHI R0,0                        *initialize retry count=0
loop TBEGIN                           *begin transaction
      JNZ       abort                 *go to abort code if CC!=0
      LT        R1,lock               *load&test the fallback lock
      JNZ       lckbzy                *branch if lock busy
    ...perform operation...
      TEND                            *end transaction
    ...
lckbzy TABORT                         *abort if lock busy; this resumes after

abort  JO       fallback              *no retry if CC=3
       AHI      R0,1                  *increment retry count
       CIJNL    R0,6,fallback         *give up after 6 attempts
       PPA      R0,TX                 *random delay based on retry count
     ... potentially wait for lock to become free
       J        loop                  *jump back to retr

fallback
    OBTAIN lock *using Compare&Swap
    ...perform operation...
    RELEASE lock
    ...
```

[3]

Transaction Execution Facility-Sample

```
TM_BEGIN(myId);                  ← Maps to a function that manages the transaction start
 long numTotalParent = (long)TM_SHARED_READ(learnerPtr->numTotalParent);
 TM_SHARED_WRITE(learnerPtr->numTotalParent, (numTotalParent + 1));
TM_END(myId);                    ← Maps to a functions that manages the transaction end

 #define TM_BEGIN(myID)          tm_begin(myID)

void tm_begin(long threadId) {
 while(!is_lock_free(threadId))
  {
   idle(threadId);
  }

  if (global_barrierPtr[threadId].tmState==TRANSITION_TO_LOCK_STATE)
  {
     acquire_a_lock(threadId);
     return;
  }

  call_tm_begin(threadId);

  if (global_barrierPtr[threadId].tmState==TRANSITION_TO_LOCK_STATE)
    acquire_a_lock(threadId);
  return; }
```



Compare and swap to get a global lock

Transaction Execution Facility-Sample

```
call_tm_begin(threadId) {
 long cc;
 long retry = 0;
  while(retry<RETRY_FACTOR) {
  cc = __TM_simple_begin();
  if (cc == 0)
      break;
    else if (cc == 3)
      break;

    ++retry;
 }
 GET_TM_CC(cc);
 if (cc != 0) {
   global_barrierPtr[threadId].tmState = TRANSITION_TO_LOCK_STATE;
   SET_TM_ROLLBACK;
 } else {
   global_barrierPtr[threadId].tmState = TRANSITION_TO_TRANSACTIONAL_STATE;
 }
 return;
}
```

Transaction Execution Facility-Sample

```
void tm_end(long threadId) {
 /*Issue TM_END if transactional*/
 if (global_barrierPtr[threadId].tmState == TRANSITION_TO_LOCK_STATE ||
     global_lock.tmLock == BUSY)
   release_a_lock(threadId);
 else  if (global_barrierPtr[threadId].tmState == TRANSITION_TO_TRANSACTIONAL_STATE)
   __TM_end();
 reset(threadId);
 return;
}
```

Transaction Execution Facility-Sample

```
# define TM_START(tid, ro)                    {
unsigned long retry = 0; unsigned long err; \
        char TM_buff[256] __attribute__((aligned(8)));\
        unsigned long line = 0;\
        restart:\
        err = __TM_begin(TM_buff);\
        if (__builtin_expect(err != 0, 0)) { \
          void *ptr;\
          int ii; retry++;\
          d->nb_aborts[err-1]++;\
          /*ptr = __TM_failure_address(TM_buff);*/ \
          ptr = (void *)line;\
          for (ii=0;ii<16;ii++) { if(d->failure_addr[ii] == NULL) d->failure_addr[ii] = ptr; if(d-
    >failure_addr[ii] == ptr) { d->failure_ctr[ii]++; break; } }\
          if (__TM_is_conflict(TM_buff)){\
            d->nb_aborts_conflict++;\
          }\
          if (__TM_is_illegal(TM_buff))\
            d->nb_aborts_illegal++;\
          if (__TM_is_footprint_exceeded(TM_buff))\
            d->nb_aborts_fe++;\
          udelay_simple((retry+d->id)%16);\
          goto restart; }
```

Transaction Execution Facility facts

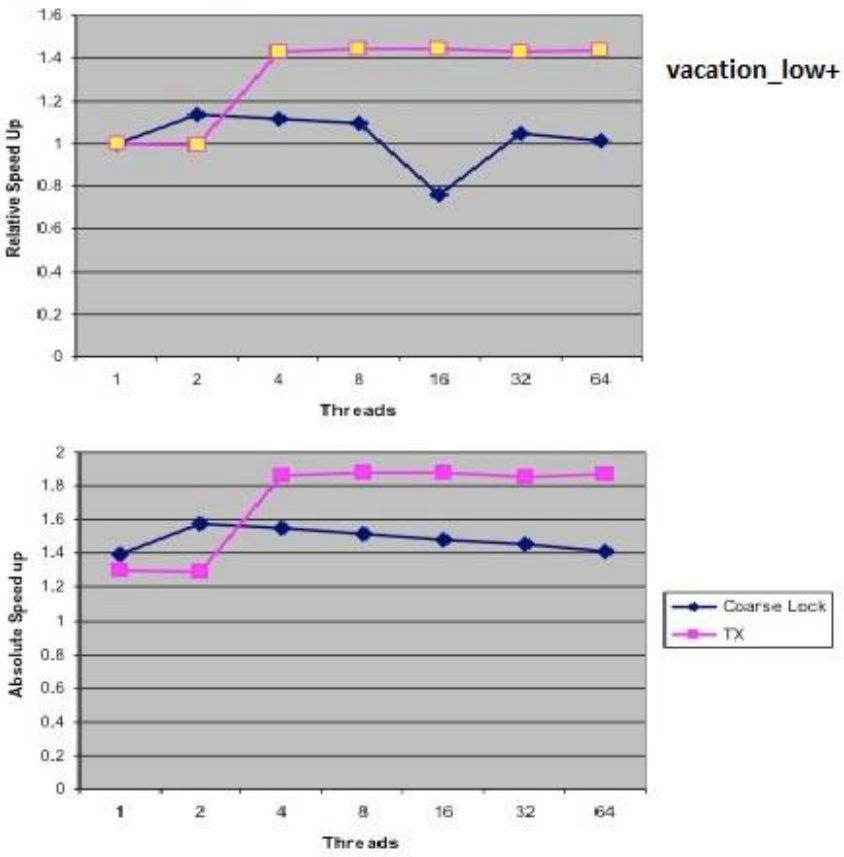
- Compiler built-ins require additional user code to handle aborts
- Use the expensive begin built-in, __TM_begin(void const*) and the debug helper built-ins, e.g. __TM_is_xxx to find good candidates for transaction memory execution
- Best to get a lock if, CC=3 due to any of the following:
  - long __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);
- Worth to TX if, CC=0, 2
- __TM_abort_assist will insert a delay based on the number of times the TX aborted; this improves-

Transaction Execution Facility facts

- ..this improves chances of success next time TBEGIN is issued
- Do the writes ahead of reads to the same variable in the transaction
- Avoid putting a large chunk of code in the TX

Transaction Execution Facility in action

- Shows the vacation benchmark from STAMP suite with low run set [4]
- vacation is an OLTP reservation system
- It has low/medium contention, and medium TX length
- Two measurements:
  - Relative- TX vs TX single thread and Coarse vs Coarse single thread
  - Absolute- TX/Coarse vs best single thread sequential run
- Lock is performing better than TX for up to 2-threads (due to different memory management for sequential version)
- TX performs up to 2X better for four threads on a 4-way system
- The locking mechanism shows degraded throughput



Debugging Optimized Code

- Programmers view of source different than the compiler view of source
- Optimized code, in order to be fast, move code around, inline functions, as well as do a lot of in register operation
- With optimization it is hard to map between source and generated code

Debugging Optimized Code - example

```
#include <stdio.h>

int foo(int input)
   {
   int a;
      {
      a = input * 2;
      }
      printf("a = %d\n", a);
      return a;
   }

int bar(int i)
   {
   i++;
   printf("i = %d\n", i);
   return i+foo(i);
   }

int main()
   {
   int i = foo(7);   foo gets inlined in main -
   int j = foo(7);
   int k = bar(7);
   return i+j+k+3;
   }
```

Debugging Optimized Code – session then

```
w/ zOS V1R13 XL C/C++
xlc -qdebug=symbol:nohook -O2 a.c
dbx a.out

(dbx64) st in main
[1] stop in 'int main()'
(dbx64) r
FDBX6432: Processing load module "a.out"
FDBX9997: The loaded module does not contain module map which may lead to bad
   performance. Suggest to use dbgld to create module map to the executable before
   debugging.
FDBX6421: Loaded debug data from "/home/visdav/FixRTC/51878/a.dbg"
[1] stopped in main at line 7 in file "a.c"  ($t2)
    7         a = input * 2;
(dbx64) print input
FDBX0082: "input" is not active
```

Debugging Optimized Code – session now

```
w/ zOS V2R1 XL C/C++
xlc -qdebug=symbol:nohook:level=8 -O2 a.c
dbx a.out

(dbx64) st in main
[1] stop in 'int main()'        File TOROLABA:/home/visdav/FixRTC/51878/a.c, Line 3.
(dbx64) r
FDBX6432: Processing load module "./a.out"
FDBX9997: The loaded module does not contain module map which may lead to bad
   performance. Suggest to use dbgld to create module map to the executable before
   debugging.
FDBX6421: Loaded debug data from "/home/visdav/FixRTC/51878/a.dbg"
[1] stopped in inline int foo(int input).$b8 at line 3 in file "a.c"  ($t2)
    3   int foo(int input)
(dbx64) print input
7
```

64-bit performance

- Instruction selection based on the operand type, no need to use the Grande version just because –q64, ALG vs AL
- Sign extend only if the operation requires the 64-bit value, e.g. type is long, long long in 64-bit and long long in 32-bit
- Reduced the path length, improved the run-time
  - 12.1% was observed on a set of CPU intensive integer based programs when compared to the same programs with the V1R13 compiler*

*
For 64-bit: LP64, XPLINK, HGPR, OPT(3), HOT, IPA(LEVEL(2)),PDF, ARCH(10), TUNE(10)

Then

Now

OFFSET	OBJECT CODE			LINE#	FILE#	PSEUDO ASSEMBLY LISTING				OFFSET	OBJECT CODE			LINE#	FILE#	PSEUDO ASSEMBLY LISTING		
0042F8				000091	25	+@3L350	DS	0H		004900				000091	25	+@3L354	DS	0H
0042F8	E330	48F8	0004	000091	25	+	LG	r3,#SPILL12(,r4,2296)		004900	E360	4968	0004	000091	25	+	LG	r6,#SPILL39(,r4,2408)
0042FE	E350	4900	0004	000091	25	+	LG	r5,#SPILL44(,r4,2304)		004906	E350	4948	0004	000091	25	+	LG	r5,#SPILL38(,r4,2376)
004304	E3A0	4908	0004	000091	25	+	LG	r10,#SPILL13(,r4,2312)		00490C	E3F0	4970	0004	000091	25	+	LG	r15,#SPILL40(,r4,2416)
00430A	E381	3000	0014	000091	25	+	LGF	r8,(*)int.rns34.(r1,r3,0)		004912	5837	6000		000091	25	+	L	r3,(*)int.rns24.(r7,r6,0)
004310	E391	5000	0014	000091	25	+	LGF	r9,(*)int.rns24.(r1,r5,0)		004916	58A7	5000		000091	25	+	L	r10,(*)int.rns22.(r7,r5,0)
004316	E3D0	4910	0004	000091	25	+	LG	r13,#SPILL45(,r4,2320)		00491A	5807	F000		000091	25	+	L	r0,(*)int.rns34.(r7,r15,0)
00431C	E3B1	A000	0014	000091	25	+	LGF	r11,(*)int.rns34.(r1,r10,0		00491E	E360	4930	0004	000091	25	+	LG	r6,#SPILL37(,r4,2352)
004322	E3C1	D000	0014	000091	25	+	LGF	r12,(*)int.rns22.(r1,r13,0		004924	E3F0	4978	0004	000091	25	+	LG	r15,#SPILL42(,r4,2424)
004328	E3E0	4918	0004	000091	25	+	LG	r14,#SPILL14(,r4,2328)		00492A	E3C0	4980	0004	000091	25	+	LG	r12,#SPILL45(,r4,2432)
00432E	E331	E000	0014	000091	25	+	LGF	r3,(*)int.rns34.(r1,r14,0)		004930	5827	6000		000091	25	+	L	r2,(*)int.rns20.(r7,r6,0)
004334	E3F0	4920	0004	000091	25	+	LG	r15,#SPILL43(,r4,2336)		004934	5E37	F000		000091	25	+	AL	r3,(*)int.rns34.(r7,r15,0)
00433A	B90A	0089		000091	25	+	ALGR	r8,r9		004938	5EA7	C000		000091	25	+	AL	r10,(*)int.rns34.(r7,r12,0
00433E	E351	F000	0014	000091	25	+	LGF	r5,(*)int.rns20.(r1,r15,0)		00493C	EC50	009F	2051	000091	25	+	RISBLG	r5,r0,H'0',H'159',H'32'
004344	E3A0	4928	0004	000091	25	+	LG	r10,#SPILL11(,r4,2344)		004942	5867	E000		000091	25	+	L	r6,(*)int.rns29.(r7,r14,0)
00434A	B90A	00BC		000091	25	+	ALGR	r11,r12		004946	19A3			000091	25	+	CR	r10,r3
00434E	E391	A004	0014	000091	25	+	LGF	r9,(*)int.rns34.(r1,r10,4)		004948	B9F2	40A3		000091	25	+	LOCRL	r10,r3
004354	E3D0	4930	0004	000091	25	+	LG	r13,#SPILL42(,r4,2352)		00494C	E330	4988	0004	000091	25	+	LG	r3,#SPILL41(,r4,2440)
00435A	19B8			000091	25	+	CR	r11,r8		004952	E3C0	4990	0004	000091	25	+	LG	r12,#SPILL43(,r4,2448)
00435C	B9E2	40B8		000091	25	+	LOCGRL	r11,r8		004958	5E27	3000		000091	25	+	AL	r2,(*)int.rns34.(r7,r3,0)
004360	B90A	0035		000091	25	+	ALGR	r3,r5		00495C	192A			000091	25	+	CR	r2,r10
004364	E380	4950	0004	000091	25	+	LG	r8,#SPILL26(,r4,2384)		00495E	B9F2	402A		000091	25	+	LOCRL	r2,r10
00436A	E351	8004	0014	000091	25	+	LGF	r5,(*)int.rns34.(r1,r8,4)		004962	5E57	8004		000091	25	+	AL	r5,(*)int.rns34.(r7,r8,4)
004370	B914	008B		000091	25	+	LGFR	r8,r11		004966	1952			000091	25	+	CR	r5,r2
004374	193B			000091	25	+	CR	r3,r11		004968	B9F2	4052		000091	25	+	LOCRL	r5,r2

64-bit performance - differences

- Less instructions in the “now” than the “then” version of the code
- No redundant sign extension
- Better usage of limited resource, register, by keeping one operand in memory, AL vs ALGR

Enterprise PL/I 4.4 Highlights

- Improved performance of PL/I applications
  - Improved exploitation of zEC12 and zBC12 hardware
  - New optimization features (e.g. improve code for UTF-16; improve code generation for Decimal-Floating-Point Zoned-Conversion Facility…)
  - 4X improvement in listing generation time
- Improved Middleware support
- Better error messages when compiling SQL programs
- New features to support IMS
  - Sparse arrays, XML cleaning and normalization; base 64 encoding/decoding
  - Significantly reduced size of IMS convertor, allowing more convertors to run in same addressing space
- Support for the latest IBM Middleware: CICS, DB2 and IMS
- UTF 16 PICTURE support
- New program diagnostics features
- Increase programmer productivity and application modernization

Decimal-Floating Point Zoned-Conversion Facility

- This facility adds a new set of instructions for converting between decimal floating-point (DFP) and zoned decimal
- Few customers are currently using DFP
- So the usefulness of these new instructions might seem limited
- But the compiler can exploit these for you – even in programs that use no floating-point data!

Terminology Review

- Zoned decimal data consists of bytes where the leftmost 4 bits are called the zone bits and the rightmost 4 bits are the decimal or numeric bits.
- Most commonly, these are the byte values representing the numbers 0-9
- Zoned decimal data is suitable for input, editing, and output of numeric data in human-readable form
- There are no arithmetic instructions that operate directly on zoned decimal
- Zoned decimal is represented in PL/I by the PICTURE data type

zEC12 Zoned-to/from-DFP Facility

- This new facility in the zEnterprise EC12 hardware adds instructions to convert from zoned decimal to DFP (and back)
- And there are already arithmetic instructions that operate on DFP as well as instructions to convert between DFP and binary integer
- Also: DFP data can be held in registers, and that helps optimization
- These new instructions will clearly help in programs that use PICTURE and DFP data

Example

- **So, for example, when 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;
```

Example

- **Under 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)
```

Example

- **While under ARCH(10), it consists of just these 8 instructions – and the loop runs more than 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)
```

zEC12 Zoned-to/from-DFP Facility

- But, more importantly, the Enterprise PL/I 4.3 compiler exploits this new facility in the zEnterprise EC12 hardware to help programs that don't even use DFP !

- For programs that convert PICTURE to FIXED BIN (or the reverse) the compiler has traditionally used packed decimal as an intermediary.

- Now it can use DFP instead, and in many cases this is faster

Example: Picture to FIXED Bin (31)

- **So, for example, when given this code to convert PICTURE to FIXED BIN**

```
pic2int: proc( ein, aus ) options(nodescriptor);
   dcl ein(0:100_000) pic'(9)9' connected;
   dcl aus(0:hbound(ein)) fixed bin(31) connected;
   dcl jx fixed bin(31);

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

Example: Picture to FIXED Bin (31)

- **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)
```

Example: Picture to FIXED Bin (31)

- **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)
```

zEC12 Zoned-to/from-DFP Facility

- In converting PICTURE to FIXED BIN, the compiler uses the new CDZT instruction that converts zoned-decimal to DFP
- In converting from FIXED BIN(31) to PICTURE, the compiler could use the new instruction CZDT that does the reverse
- However, our tests showed that this set of instructions performed slightly worse than the old set
- This is another example of the strength of the compiler: it will exploit new instructions exactly when they help you - and as another example of this, consider conversions of UNSIGNED FIXED BIN(32) to PICTURE

Example: Unsigned Fixed Bin(32) to Picture

- **So, when given this code to convert UNSIGNED FIXED BIN to PICTURE**

```
uint2pic: proc( ein, aus ) options(nodescriptor);
   dcl ein(0:100_000) unsigned fixed bin(32) connected;
   dcl aus(0:hbound(ein)) pic'(10)9' connected;
   dcl jx fixed bin(31);

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

Example: Unsigned Fixed Bin(32) to Picture

- **Under ARCH(9), the heart of the loop consists of these 10 instructions**

```
005C 586E F000          L r6,_shadow2(r14,r15,0)
0060 4140 1000          LA r4,#AMNESIA(,r1,0)
0064 C050 0000 0026     LARL r5,F'38'
006A 41E0 E004          LA r14,#AMNESIA(,r14,4)
006E C067 8000 0000     XILF r6,F'-2147483648'
0074 4E60 D0A0          CVD r6,#pd579_1(,r13,160)
0078 D207 D0A8 D0A0     MVC #pd581_1(8,r13,168),#pd579_1(r13,160)
007E FA75 D0A8 5000 AP  #pd581_1(8,r13,168),+CONSTANT_AREA(6,r5,0)
0084 D207 D098 D0A8     MVC _temp1(8,r13,152),#pd581_1(r13,168)
008A F397 4000 2000     UNPK _shadow1(10,r4,0),_temp1(8,r2,0)
```

Example: Unsigned Fixed Bin(32) to Picture

- **While under ARCH(10), it consists of only 8 instructions (but uses DFP in several of them), and combined with the facts that only the L and the new CZDT refer to storage and that packed decimal is avoided, the loop runs more than 2 times faster**

```
005C 5851 E000            L r5,_shadow1(r1,r14,0)
0060 EB30 0003 00DF       SLLK r3,r0,3
0066 EB40 0001 00DF       SLLK r4,r0,1
006C 1E34                 ALR r3,r4
006E 4110 1004            LA r1,#AMNESIA(,r1,4)
0072 B953 0005            CDLFTR f0,r5
0076 B9FA 303F            ALRK r3,r15,r3
007A ED09 3000 00A8       CZDT f0,#AddressShadow(10,r3,0),b'0000'
```

zEC12 Zoned-to/from-DFP Facility

- To summarize some of the lessons from these examples:
  - 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
  - The 4.4 PL/I compiler knows when these new ARCH(10) instructions will help and will exploit them appropriately for you

- This is just one of many Enterprise PL/I 4.4 compiler for more details about this or other 4.4 new feature you can contact
  - Peter Elderon, elderon@us.ibm.com
  - Attend the next What's New … session at SHARE

Connect With Us

Cafes

C/C++
http://ibm.com/rational/community/cpp

COBOL
http://ibm.com/rational/community/cobol

Fortran
http://ibm.com/rational/community/fortran

PL/I
http://ibm.com/rational/community/pli

Feature Requests

C/C++
http://ibm.com/developerworks/rfe/?PROD_ID=700

COBOL
http://ibm.com/developerworks/rfe/?PROD_ID=698

Fortran
http://ibm.com/developerworks/rfe/?PROD_ID=701

PL/I
http://ibm.com/developerworks/rfe/?PROD_ID=699

Like IBM Compilers on Facebook

Follow IBM Compilers on Twitter

References:

- [1] V. Vokhshoori, M. Mitran, Evalutating the zEC12 Transactional Execution Facility, IBM System Magazine, Oct. 2012
- [2] Peter Elderon, What's New in Enterprise PL/I V4R3 and C/C++ V1.13, SHARE San Francisco, Feb. 2013
- [3] C. Jacobi, T. Slegel, D. Greiner, Transactional Memory Architecture and Implementation for IBM System z, ACM, 2012
- [4] STAMP: Stanford Transactional Applications for Multi-Processing Chi Cao Minh, JaeWoong Chung, Christos Kozyrakis, Kunle Olukotun In IISWC '08: Proceedings of The IEEE International Symposium on Workload Characterization, Sept. 2008.


THANK
YOU SO
MUCH!

Two-Column Slide (Type Size=28)

- Topic A (Type Size=24)
  - Subtopic 1 (Type Size=22)
  - Subtopic 2 (Type Size=22)
  - Subtopic 3 (Type Size=22)
  - Subtopic 4 (Type Size=22)
- Topic B (Type Size=24)

- Topic C (Type Size=24)
  - Subtopic 1 (Type Size=22)
  - Subtopic 2 (Type Size=22)
  - Subtopic 3 (Type Size=22)
    - Sub-subtopic 1 (Type Size=20)
    - Sub-subtopic 2(Type Size=20)
- Topic D (Type Size=24)

Slide with Table

Slide with Text & Graphic