Transactional memory is a hot topic in parallel programming and many software implementations have been developed, but solutions based on hardware transactional memory have superior performance. IBM® is leading the way with its implementations of transactional memory on its hardware platform, first on IBM Blue Gene® and IBM z/OS®, then recently with IBM AIX® support of hardware transactional memory on the POWER8™ platform. This article introduces the IBM XL compiler hardware transactional memory built-in functions, including its installation and configuration with examples using XL C++ and XL Fortran.

**Introduction to transactional memory**

Transactional memory is a feature first proposed more than twenty years ago in academia as a mechanism to enable atomic operations on an arbitrary set of memory locations. It is intended to simplify parallel programming, specifically for accessing shared data across multiple threads.

Prior to transactional memory, accesses to shared data were synchronized by the use of locks. Threaded code that needs access to shared data must first acquire the data lock, then access the shared data, and then release the lock. On many systems, acquiring locks can be expensive, making the process of accessing shared data vastly more expensive than accessing non-shared data. This additional locking can be especially burdensome when the shared data has low contention between the multiple threads. One of the main uses of transactional memory is the speed up of lock-based programs by using the speculative execution of lock-based critical sections, without first acquiring a lock. This allows applications that have not been carefully tuned for performance to take advantage of the benefits of fine-grain locking. The transactional programming model also provides productivity gains when developing lock-based, shared memory programs.

**Hardware transactional memory support**

Given the high cost of implementing transactional memory in hardware, the research community developed several implementations of software transactional memory. Despite significant progress...
in recent years towards practical and efficient software transactional memory, there is a growing consensus that some hardware support for transactional memory is desirable.

IBM was a pioneer to include a commercial microprocessor using hardware transactional memory. On IBM Blue Gene/Q systems, starting in 2011, the transactional memory model is implemented in the hardware to access all the memory up to the 16 GB boundary. IBM POWER8, the newest IBM Power Systems™ processor (at the time of this publication), contains support for hardware transactional memory as an important performance feature. IBM POWER8 processor supports IBM AIX, IBM i, and Linux.

**Hardware transactional memory intrinsic functions support on IBM XL compiler**

The IBM XL compiler uses the capabilities of the latest POWER8 architecture. In IBM XL Fortran for AIX, V15.1 and XL C/C++ for AIX, V13.1, a set of POWER8 transactional memory built-in functions are introduced to support the hardware transactional memory features in POWER8 processors.

In IBM XL Fortran for AIX, the TRANSACTIONAL_MEMORY intrinsic module has been introduced. This module provides functions that allow you to designate a block of instructions or statements to be treated atomically. Such an atomic block is known as a transaction. When a thread runs a transaction, all of the memory operations within the transaction occur simultaneously from the perspective of the other threads. For some kinds of parallel programs, a transaction implementation can be more efficient than other implementation methods, such as locks. These intrinsic procedures can be used to mark the beginning and end of transactions and to diagnose the reasons for failure.

The TRANSACTIONAL_MEMORY module provides the following procedures:

- Transaction begin and end functions
- Transaction abort functions
- Transaction inquiry functions

The transactional state is entered following a successful call to TM_BEGIN or TM_SIMPLE_BEGIN and ended by TM_END, TM_ABORT, TM_NAMED_ABORT, or by a transaction failure.

A transaction failure occurs when any of the following conditions are met:

- Memory that is accessed in the transactional state is accessed by another thread before the transaction completes.
- The architecture-defined footprint for memory accesses within a transaction is exceeded.
- The architecture-defined nesting limit for nested transactions is exceeded.

Transactions can be nested. TM_BEGIN or TM_SIMPLE_BEGIN can be used in the transactional state. Within an outermost transaction initiated with TM_BEGIN, nested transactions must be initiated with TM_SIMPLE_BEGIN or by TM_BEGIN using the same buffer of the outermost
containing transaction. A nested transaction is subsumed into the containing transaction. Therefore, a failure of the nested transaction is treated as a failure of all containing transactions and the nested transaction completes only when all containing transactions complete.

Similarly, in IBM XL C/C++ for AIX, a set of transactional memory built-in functions are introduced, including transaction BEGIN, END, and ABORT functions.

In the transactional memory built-in functions, the `TM_buff` parameter allows for a user-provided memory location to be used to store the transaction state and debugging information. The transactional state is entered following a successful call to `__TM_begin` or `__TM_simple_begin`, and ended by `__TM_end`, `__TM_abort`, `__TM_named_abort`, or by a transaction failure.

Transactions can be nested. `__TM_begin` or `__TM_simple_begin` can be used in the transactional state. Within an outermost transaction initiated with `__TM_begin`, nested transactions must be initiated with `__TM_simple_begin`, or by `__TM_begin` using the same buffer of the outermost containing transaction. A nested transaction is subsumed into the containing transaction. Therefore, a failure of the nested transaction is treated as a failure of all containing transactions and the nested transaction completes only when all containing transactions complete.

### Hardware transactional memory built-in samples

This section provides samples with Fortran and C/C++ languages. The examples illustrate how to use hardware transactional memory built-in functions for the beginning and end of transactions, and how to identify the reasons for the possible failure.

**IBM XL Fortran**

The following is the sample code for hardware transactional memory built-in functions in Fortran language:

#### Listing 1. Sample code for Fortran hardware transactional memory built-in functions

```fortran
! HTM_demo.f: an example of using HTM built-in
program main
  use transactional_memory
  implicit none
  type(tm_buff_type) tstatus
  integer :: arr1(5), arr2(5), i
  integer(8) :: rc_failcode

  ! Initialize both array with same element
  do i = 1, 5
    arr1(i) = i
    arr2(i) = i
  end do

  if (tm_begin(tstatus) == tm_success) then
    ! Transaction started successfully
    arr1(3) = 0
    arr2(3) = 0
  if (tm_end() /= tm_success) then
    print *, 'transaction end fail'
    error stop 1
  end if
end if
```

IBM XL compiler hardware transactional memory built-in functions
for IBM AIX on IBM POWER8 processor-based systems
When a program encounters a failure within a series of operations, it might leave the function in a half-completed state, therefore, leaving the data in an unexpected state. In this example, both arrays are initialized with the same element. With hardware transactional memory built-in functions, the consistent state of both arrays is ensured, no matter whether the transaction succeeds or fails. Every element would be identical after the transaction.

To compile and run the above code, you must first ensure that XL Fortran for AIX V15.1 is properly installed and configured on the AIX system with POWER8 processors. For more information, refer to the [XL Fortran Installation Guide](https://www.ibm.com/developerWorks/ibmตำแหน่ง). Run the following command to compile the sample code:

```bash
xlf2003 HTM_demo.f
```

The executable file `a.out` is generated. To run the program, enter `a.out` on the command line and you will see the result as shown in Figure 1.

**Figure 1. Output of the Fortran sample code**

```
$ ./a.out
1 2 0 4 5
1 2 0 4 5
$  
```

**IBM XL C/C++**

The following is the sample code for hardware transactional memory built-in functions in C language:

**Listing 2. Sample code for C++ hardware transactional memory built-in functions**

```c
#include <builtins.h>
#include <stdio.h>
char buf[256] __attribute__((aligned(8)));
unsigned long long zero = 0;
```
int main(void)
{
    long rc_begin, rc_end, i;
    long ar1[5], ar2[5];
    unsigned long long rc_buf;

    rc_begin = rc_end = -1;

    // Initialize both array with same element
    for (i = 0; i < 5; i++)
    {
        ar1[i] = ar2[i] = i;
    }

    if ( (rc_begin = __TM_begin(buf)) == 0 )
    {
        // Transaction started successfully
        ar1[2] = 0;
        ar2[2] = 0;
        if ( (rc_end = __TM_end()) != 0 )
        {
            printf("the transaction failed\n");
            return 1;
        }
    }
    else
    {
        if (__TM_failure_code(buf) == zero)
        {
            printf("The transaction failed, but an abort is not reported in the buffer\n");
            return 2;
        }
    }

    for (i = 0; i < 5; i++) printf("arr1[%ld] = %ld; ", i, ar1[i]);
    printf("\n");

    for (i = 0; i < 5; i++) printf("arr2[%ld] = %ld; ", i, ar2[i]);
    printf("\n");

    // Verify each element of this array is still identical
    for (i = 0; i < 5; i++)
    {
        if (ar1[i] != ar2[i])
        {
            printf("No. %ld element is not identical", i);
            return 3;
        }
    }
}

Similarly, to compile and run the above code, you must first ensure that XL C/C++ for AIX V13.1 is properly installed and configured on the AIX system with POWER8 processors. For more information, refer to the XL C/C++ installation guide.

Run the following command to compile the sample code:

```bash
xlc HTM_demo.c
```

The executable file `a.out` is generated. To run the program, enter `a.out` on the command line and you will see the result as shown in Figure 2.
Figure 2. Output of the C/C++ sample code

```
$ ./a.out
```

Resources

Here are a few useful resources that are relevant to this article.

- IBM Knowledge Center:
  - IBM XL C/C++ for AIX, V13.1 documentation
  - IBM XL Fortran for AIX, V15.1 documentation

© Copyright IBM Corporation 2014
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)