Xbox 360 CPU Overview

By Bruce Dawson
Software Design Engineer
Advanced Technology Group (ATG)

Published: April 20, 2004
Updated: October 4, 2005

This paper summarizes the CPU that powers the Xbox 360 console. This paper does not try to cover everything—more detail is available in other papers.

Overview

The Xbox 360 CPU is a custom IBM part based on the PowerPC architecture. It features three identical cores and a 1-MB L2 cache all on one chip. The main features of the Xbox 360 CPU are:

- Three identical custom 64-bit PowerPC cores on a single chip running at 3.2 GHz
- Two instructions per cycle per core
- 32-KB 2-way 51.2 GB/sec L1 instruction cache per core
- 32-KB 4-way 51.2 GB/sec L1 data cache per core
- Shared 1-MB 8-way 102.4 GB/sec L2 cache on the same chip
- Direct read access to the L2 cache by the GPU
- 128-bit SIMD vector unit for float and integer operations
- Scalar floating-point unit, supporting float and double precision
- The following registers, one set for each thread on each core:
    - 32 64-bit integer registers
    - 32 64-bit floating-point registers
    - 128 128-bit vector registers
    - Instruction pointer, link register, count register, condition register, and so on
- 115.2 GFLOPS total power (one vector multiply-add and one vector D3D pack per core per cycle)
- Front side bus bandwidth of 10.8 GB/sec for reads, 10.8 GB/sec for writes
- Big-endian byte ordering
- Custom security engine

Some of the resources, such as the registers, are per-thread resources, meaning there are six copies of them on the chip. Other resources—such as the L1 caches, vector unit, and scalar floating-point unit—are per-core resources, meaning that two hardware threads share each of them.

Because the Xbox 360 CPU is based on the PowerPC architecture there are many sources of information for learning about this processor. A good starting point is the AltiVec PEM and PPC PEM documentation that ships with the Xbox 360 Development Kit. Because there are many sources of generic PowerPC information, this paper concentrates on what makes the Xbox 360 processor different from regular PowerPC processors.

The main architectural difference between the Xbox 360 CPU and the regular PowerPC processor—besides the three cores and the shared L2 cache on one chip—is the enhanced

vector units, discussed in detail below. The integer unit and scalar floating-point unit are virtually unchanged from the PowerPC specification. The scalar floating-point unit includes these optional instructions: square root, reciprocal square root estimate, reciprocal estimate, and select. The integer unit features an enhanced prefetch instruction.

Vector Processing—VMX128

VMX, also known as AltiVec or VelocityEngine, is the PowerPC vector architecture. The large register set and powerful instructions such as vector multiply-add make VMX an excellent architecture for game programmers.

VMX128 is the custom Xbox 360 vector architecture based on VMX. VMX128 features 96 more registers than VMX and even more powerful instructions. These alterations to the already robust PowerPC vector architecture are designed to optimize game performance.

Some of the new instructions for VMX128 include the following.

Instruction	Operation performed	Description
vmsum3fp and **vmsum4fp**	dot product	Performs a three- or four-element dot product on two vectors and stores the result to all elements of the destination.
vpermwi	vector permute word immediate	Rearranges or swizzles words or floats arbitrarily from one vector to another.
vpkd3d and **vupkd3d**	vector unpack D3D datatype	Packs and unpacks data from a Microsoft® Direct3D® API data type, such as **NORMPACKED32** or four **FLOAT16** types to a four float vector.
vmulfp	vector multiply	Performs a vector multiply instruction that is simpler than the VMX solution of doing a vector multiply-add with an add of zero.
lvlx, lvrx, stvlx, stvrx	misaligned vector load and store	Used for misaligned loads and stores of vector data.
vlrimi	vector rotate left immediate and mask insert	Performs a 32-bit-based vector rotation and mask insertion.

For more information on the Xbox 360 additions to VMX, see the white paper "New Instructions in VMX128."

In addition, VMX128 supports most of the instructions from the regular VMX vector architecture, in most cases extended to support 128 registers. Some of the powerful VMX instructions that have been extended for VMX128 include the following.

Instruction	Operation performed	Description
vmaddfp	vector multiply-add	Multiplies two vectors and adds a third
vnmsubfp	vector negate multiply-subtract	Multiplies two vectors, subtracts a third, and negates the result
vrefp	vector reciprocal estimate	Calculates a 12-bit accurate reciprocal
vrsqrtefp	vector reciprocal square root estimate	Calculates a 12-bit accurate reciprocal square root

A few VMX vector instructions were removed from VMX128 in order to make room for more useful instructions. The main instructions cut were the vector integer multiply instructions.

VMX doesn't have a divide instruction or a square root instruction. However, it does have **vrefp** (vector reciprocal estimate) and **vrsqrtefp** (vector reciprocal square root estimate). These instructions give a 12-bit accurate estimate, which is often enough. The accuracy can be doubled by doing a Newton-Raphson iteration. This refinement requires just two instructions for reciprocal or four instructions for reciprocal square root—**vnmsubfp** is very useful for this process.

The vector and scalar floating-point math instructions have long latencies—12 cycles for **vmaddfp**, for instance. This means that if a vector instruction uses the result of a previous **vmaddfp** instruction, the second instruction won't start executing until 12 cycles after the first instruction. However, the vector and scalar units are pipelined so they can issue a floating-point math instruction every cycle, as long as data dependencies don't force a stall. Therefore, efficiently scheduling code—including unwinding loops—is important in order to realize the full potential of the Xbox 360 CPU.

VMX128 instructions can be accessed through Xbox 360 compiler intrinsics. Using these frees programmers from worrying about register allocation and instruction reordering. VMX programming using the intrinsics looks like this:

```
#include <ppcintrinsics.h>
__vector4 Transform( __vector4 Vector, __vector4 Matrix[] )
{
    // __vmsum4fp == 4 element dot product
    __vector4 x = __vmsum4fp( Vector, Matrix[0] );
    __vector4 y = __vmsum4fp( Vector, Matrix[1] );
    __vector4 z = __vmsum4fp( Vector, Matrix[2] );
    __vector4 w = __vmsum4fp( Vector, Matrix[3] );
    // __vmrglw is Vector MeRGe Low Word
    __vector4 t1 = __vmrglw( x, z );    // t1 = { x, z, x, z };
    __vector4 t2 = __vmrglw( y, w );    // t2 = { y, w, y, w };
    __vector4 result  = __vmrglw( t1, t2 ); // { x, y, z, w };
    return result;
}
```

The compiler maps variables of type `__vector4` to VMX registers. This particular routine transforms a vector by a row major matrix, using the VMX128 dot product instruction **vmsum4fp**.

Caches and Memory

Fast CPUs have to wait a long time when accessing memory, and the Xbox 360 CPU is no exception. Therefore, an understanding of the caches and memory is crucial to getting maximum performance.

Front-Side Bus

The front side bus connects the Xbox 360 CPU to the GPU. The front side bus is used by the CPU for reading and writing memory and is also used when the GPU reads from the CPU's L2 cache.

The front side bus has a raw bandwidth of 10.8 GB/sec in each direction. After subtracting some signaling overhead, there is about 10.2 GB/sec available for data. The CPU can use a maximum of about 5.6 GB/sec of this bandwidth when reading from memory. The full bandwidth is available when writing and when the GPU is reading from the CPU's L2 cache.

L1 Caches

Each core has a 32-KB four-way L1 data cache and a 32-KB two-way L1 instruction cache, both with 128-byte cache lines. The instruction cache latency is four cycles, but this latency is normally hidden by the pipelines. The data cache latency is five cycles, but again this is mostly hidden by the pipelines—for example, you can issue a VMX load on one cycle and a VMX math instruction that uses the result two cycles later.

Each L1 cache can do a 128-bit read or write every cycle, so each L1 cache has a maximum of 51.2 GB/sec of bandwidth, shared between reads and writes. The L1 data cache is *store-through* to the L2 cache, which means that when you start writing data it immediately starts moving towards the L2 cache. The L1 cache does not allocate on write misses, which means that an L1 cache line is not allocated for writes—the L1 cache is reserved for data already read, which is an efficient change from Xbox®, where writes can pollute the L1 cache. On Xbox 360, a write only goes to the L1 cache if the operation writes to an address already cached there.

L2 Cache

The 1-MB eight-way L2 cache also has 128-byte cache lines. It is on-chip and is shared between the three cores. The L2 cache runs at half of the core frequency, and the path between the L1 caches and the L2 is 64 bytes (512 bits) wide. This gives the L2 cache a maximum total bandwidth of 102.4 GB/sec.

The maximum L2 bandwidth is only achieved if the full 64-byte data path is used. The full path is automatically used on read operations, where a 128-byte cache line can be read in two L2 cache clock cycles. Although the cores can only write 16 bytes at a time, each core

has eight write-gathering buffers. These buffers allow multiple writes to be gathered together and then sent down as one 64-byte block. Write gathering is important for bandwidth-intensive processing.

The L2 cache is a *write-back* cache, which means that data is not written back to memory until it is flushed or forced out by other data.

Cache Locking

Part of the L2 cache can be locked down, either to allow the GPU to read CPU-written data without it going through memory, or just to ensure that important data stays in the cache. The L1 caches cannot be locked.

Cache Control Instructions

Three main cache control instructions are used for optimizing memory accesses: **dcbf**, **dcbz128**, and **dcbt**.

The **dcbf** instruction flushes a cache line to memory and frees it from the L1 and L2 caches, making those cache lines available for future use.

The **dcbz128** instruction zeroes the specified cache line in the L2 cache—invalidating that address in the L1 if it was cached there. This optimization is important when writing large blocks of data, because without it the CPU would have to fetch the line of data just to have it overwritten. Using **dcbz128** avoids these fetches, conserving bandwidth and avoiding latency related stalls—it's like a very fast call to **memset(ptr, 0, 128);**. For best performance, you should use **dcbz128** approximately 140 cycles ahead of when you will write to the cache line.

The **dcbt** instruction prefetches the specified cache line. If you do this prefetch far enough in advance of when you need the data, you can hide memory latency. This is important, because when a memory access misses both caches, the latency is approximately 610 cycles. There can be up to eight prefetches running in parallel. This means that when processing streaming data you should prefetch eight cache lines (1KB) ahead of where you are working. Each cache line load still takes about 610 cycles, but the pipelined fetches mean the total memory throughput is much greater.

The data can be prefetched into both L1 and L2 or just into L1. The option to skip the L2 cache was added to the Xbox 360 CPU to avoid polluting the L2 cache with transient data. Skipping the L2 cache can improve performance when used carefully.

It is important to understand that the cache control instructions are optimizations. They are not needed for sharing data between cores. In order to share data between cores, you just need to use synchronization primitives, such as critical sections or semaphores—this guarantees coherency.

The Xbox 360 CPU does not support *automatic* data prefetch. Automatic data prefetch can be problematic because it brings in data that may not be used. The Xbox 360 CPU also does not have an instruction to prefetch more than one cache line of data. Instead, multiple

prefetch instructions can be issued and are processed in parallel. The Xbox 360 CPU does perform automatic instruction prefetch.

Careful use of the cache control instructions can dramatically improve Xbox 360 performance.

Summary

Careful use of the caches is important for getting maximum performance from Xbox 360. This care includes designing your data structures to minimize random memory access and to fit as many structures in a cache line as possible.

Dealing with instruction latencies is important for getting maximum performance from the Xbox 360 CPUs. Consider unwinding critical loops or otherwise structuring your code to minimize these dependencies.

The six hardware threads and custom vector units give the Xbox 360 CPU tremendous computing power, and this power is optimized for game programming. We look forward to seeing the new possibilities and great games that this CPU will facilitate.