Memory Controller: Page Operations

Abstract: This is a short description of possible memory controller extensions. This is a modification of the old usenet article below.
Author: Benjamin Kalytta (benjamin@kalytta.com)
Url: http://www.kalytta.com
Date: November 2008, based on original idea of March 2004
Revision: 0.4

Further discussion about this topic can be found in alt.os.development newsgroup (http://groups.google.de/groups/search?q=kalytta+pset)

1 Scope

This document describes possible extensions for hardware based memory controllers in x86 based system architecture. These extensions are namely **page fill** and **page copy** operations.

Most modern operating systems require that new allocated memory regions are zeroed for security reason before usage. Zeroing of unused memory regions is mostly done in background or on memory request. Zeroing is a significant sink of processor resources even by using SIMD instruction optimized code. Above all it is a very stupid operation which shouldn't be performed by the CPU. Another kind of often used operation is memory region copy, especially copy-on-write operations. Copy-on-write is often used in process creation operation to map dynamic libraries and certain data regions into new process space. As soon as this region is modified (written to) this region need to be copied to a new location.

The idea is, to outsource some operation to the memory controller which is integrated in modern central processing units like AMD Athlon 64™ and Intel "Nehalem" Architecture which guarantees low latencies.

1.1 Terminology

Description of some abbreviations used in this document.

Unit	Smallest by memory controller addressable unit i.e. 4KiB. Unit size may be changeable by the CPU for example to support various kinds of page translation mechanisms.
Copy-On-Write	Is a mechanism to support delay time copy. Memory region will not be copied until first time it is written to it
CPU	Central Processing Unit
MCOC	Memory Controller Operation Cache

2 Operations

Memory controller operations need to be accessible by the CPU in some way. This can be done by either through some memory mapping mechanism or by adding new CPU instructions. I'll discuss the later one and will call them **PSET** and **PCPY**. These operations perform on one unit. They should be implemented as non privileged instructions and should perform on logical addresses instead of physical memory addresses. Access to memory pages is restricted by usual way through various supported memory address translation and protection mechanisms (Paging, Protected Mode).

2.1 PSET

Start memory controller bus transaction and fill specified page with value which is specified either in CPU register or as immediate value. Filling more than one page requires multiple calls of PSET. Memory addresses should be a multiple of the memory controller unit boundary, if not address will automatically be rounded down to next unit boundary.

Mnemonic	Description
PSET *mem, imm*	Zeroing page at address [mem], Fill value is specified in immediate value
PSET *mem, reg*	Zeroing page at address [mem], Fill value is specified in register

The memory controller contains a kind of n-way associative cache (Memory Controller Operation Cache) to store each PSET operation requests. This cache will save the address of the operation, the kind of operation and various operations specific parameters like the fill pattern. This data stays in this cache as long as the operation is not completed. If during that time a memory read request to one of the addresses in cache is made, the memory controller can answer this request without any delay. It is not required to read some datum from memory. However if a write request is made before the operation is completed, the write request will be delayed.

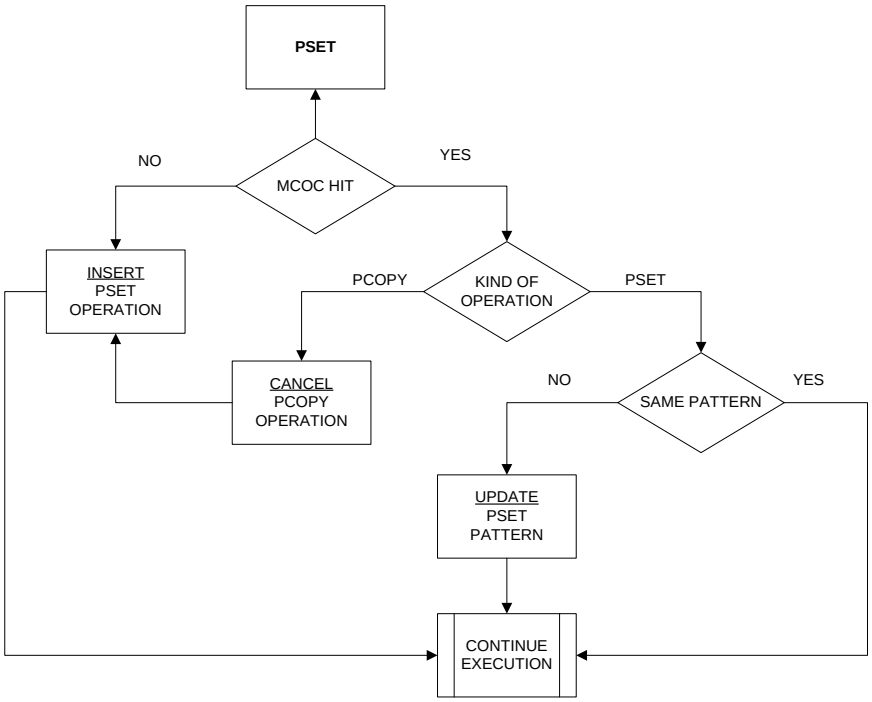

Figure 1: PSET Operation

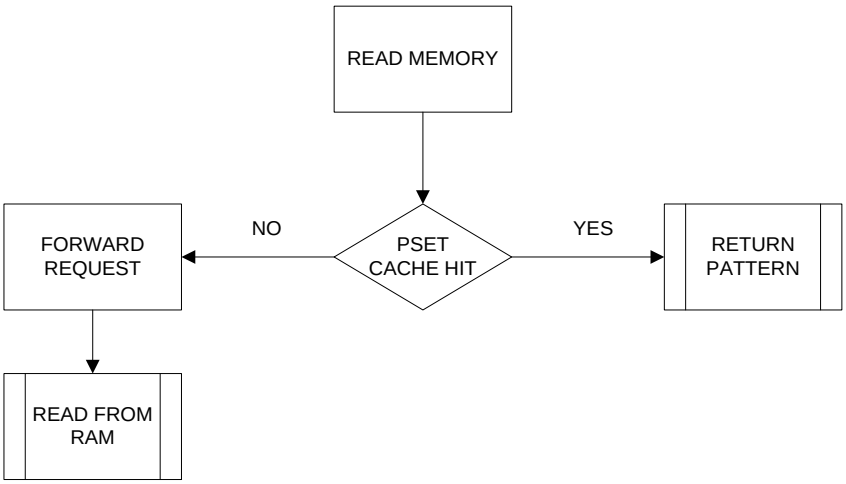

Figure 2: Memory read request

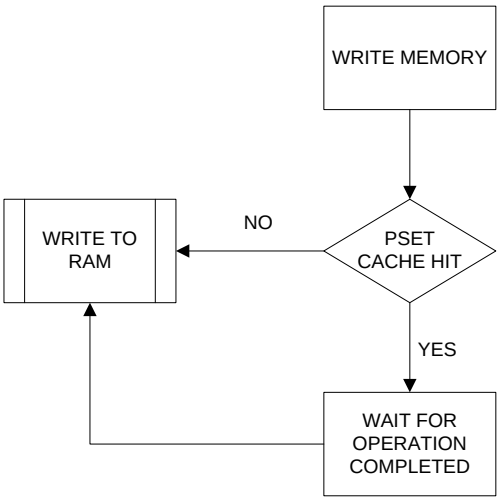

Figure 3: Memory write request

2.1.1 Advantages

The advantage of this operation is that filling (zeroing) memory operation is not done by CPU any more but by the memory controller. Each memory read request will also be answered soon without typical access latency of reading from DDR RAM.

2.2 PCPY

Start memory controller bus transaction to copy a memory page.

Mnemonic	Description
PCPY *mem, reg*	Copy page specified as indirect memory addresses

Internally the memory controller knows two PCPY operations. One stores the address to copy from (PCPY FROM) the other one stores the destination address of the page to be copied to (PCPY TO). See Example below.

2.3 Examples

This example shows a PSET operation which fills 2 page units with value 0xFFEEDDCC at start address 0x10000000 followed by a PCPY operation to copy 1 page unit from address 0x20000000 to 0x10002000. We assumes a **32 bits x86** architecture with a page unit size of 4 KiB.

```
    MOV EDI, 0x10000000
    MOV EAX, 0xFFEEDDCC
    MOV ECX, 2
LFILL:
    PSET [EDI], EAX
    ADD EDI, 0x00001000
    DEC ECX
    JNZ LFILL
    …
    MOV ESI, 0x20000000
    PCPY [EDI], ESI
```

In this example we assume that each memory controller can access no more than 2^{32} logical memory addresses and has an full associative MCOC which results in 1048576 required cache lines. Each cache line is responsible for 4096 bytes of data (one page unit). One page line could be very small since the only data saved is the operation (PSET/PCOPY) and operation specific data like fill pattern or source address and may be some flags i.e. “is locked”. Say 5 bytes are required per cache line.

Tag	Operation	Data
0x00000		
...		
0x10000		
0x10001		
0x10002		
0x10003		
...		
0xFFFFF		

Figure 4: Cache before PSET is still empty

Tag	Operation	Data
0x00000		
...		
0x10000	PSET	0xFFEEDDCC
0x10001		
0x10002		
0x10003		
...		
0xFFFFF		

Figure 5: After first PSET iteration

Tag	Operation	Data
0x00000		
...		
0x10000	PSET	0xFFEEDDCC
0x10001	PSET	0xFFEEDDCC
0x10002		
0x10003		
...		
0xFFFFF		

Figure 6: After second PSET iteration

Tag	Operation	Data
0x00000		
...		
0x10000	PSET	0xFFEEDDCC
0x10001	PSET	0xFFEEDDCC
0x10002	PCPY FROM	0x20000000
0x10003		
...		
0x20000	PCPY TO	0x10002000
...		
0xFFFFF		

Figure 7: After PCPY operation

You will notice that there are 2 PCPY operations; one specifies the source and the other the destination address. This is important in case when to one of this address is written to before PCPY FROM is executed.

Consider following cases:

a. A write request is made to address 0x10002000 before the PCPY operation is executed
b. A write request is made to address 0x20000000 before the PCPY operation is executed
c. A PSET/PCPY request to address 0x20000000 is made before the PCPY operation is executed
d. A PSET/PCPY request to address 0x10002000 is made before the PCPY operation is executed
e. A read request is made to address 0x10002000 before the PCPY operation is executed

Case a. is the typical copy-on-write case where a shared page is modified by another process or thread.

```
…
MOV ESI, 0x20000000
PCPY [EDI], ESI
…
MOV [EDI], 0
```

In this case we need to copy page at source address 0x20000000 to 0x10001000 first before we can continue. This source address is stored as operation specific data in the operation cache. To prevent that any modification is made to both of the cache lines during execution the cache lines will be locked until page is copied. After that the PCPY operations will be removed from both cache lines.

Case b. is the case where the source page has been modified before it has been copied.

```
…
MOV ESI, 0x20000000
PCPY [EDI], ESI
…
MOV [ESI], 0
```

The last MOV instruction will start the PCPY operation in this case because this operation was still queued.

Case c. is similar to case b. because we try to write to an address whose page has not been copied yet.

```
…
MOV ESI, 0x20000000
PCPY [EDI], ESI
…
PSET [ESI], 0
```

The PCPY operation needs to be executed first before PSET can be queued.

The last case d. is uncritical. The page 0x20000000 has not been copied yet to 0x10002000 but since PSET will fill the entire page this PCPY operation can be safely canceled. In fact both PCPY operation need to be removed (PCPY HINT and PCPY FROM).

```
…
MOV ESI, 0x20000000
PCPY [EDI], ESI
…
PSET [EDI], 0
```

Case e. is also not critical. There are 2 approaches: Either each read request is forwarded to address 0x20000000 or we will follow steps of case a.

3 Consideration

Both of these operations may be delayed (execute-on-write mechanism) to support faster transactions especially when multiple page operations are executed instead of direct executing. It is also uneconomic to use a full associative cache, may be an n-way associative cache is more appropriate.

4 CPU cache consideration

Since the CPU caches don't know anything about of these memory controller specific operations, each cache line corresponding to any of the operation specific addresses has to be invalidated. If we assume a cache line size of 64 bytes the CPU needs to invalidate 64 cache lines in worst case scenario for each memory controller specific instruction.