PAT (Page Attribute Table)

x86 Page Attribute Table (PAT) allows for setting the memory attribute at the page level granularity. PAT is complementary to the MTRR settings which allows for setting of memory types over physical address ranges. However, PAT is more flexible than MTRR due to its capability to set attributes at page level and also due to the fact that there are no hardware limitations on number of such attribute settings allowed. Added flexibility comes with guidelines for not having memory type aliasing for the same physical memory with multiple virtual addresses.

PAT allows for different types of memory attributes. The most commonly used ones that will be supported at this time are:

WB	Write-back
UC	Uncached
WC	Write-combined
WT	Write-through
UC-	Uncached Minus

PAT APIs

There are many different APIs in the kernel that allows setting of memory attributes at the page level. In order to avoid aliasing, these interfaces should be used thoughtfully. Below is a table of interfaces available, their intended usage and their memory attribute relationships. Internally, these APIs use a reserve_memtype()/free_memtype() interface on the physical address range to avoid any aliasing.

API	RAM	ACPI,...	Reserved/Holes
ioremap	--	UC-	UC-
ioremap_cache	--	WB	WB
ioremap_uc	--	UC	UC
ioremap_wc	--	--	WC
ioremap_wt	--	--	WT
set_memory_uc, set_memory_wb	UC-	--	--
set_memory_wc, set_memory_wb	WC	--	--
set_memory_wt, set_memory_wb	WT	--	--
pci sysfs resource	--	--	UC-
pci sysfs resource_wc is IORESOURCE_PREFETCH	--	--	WC
pci proc !PCIIOC_WRITE_COMBINE	--	--	UC-
pci proc PCIIOC_WRITE_COMBINE	--	--	WC
/dev/mem read-write	--	WB/WC/UC-	WB/WC/UC-
/dev/mem mmap SYNC flag	--	UC-	UC-
/dev/mem mmap !SYNC flag and any alias to this area	--	WB/WC/UC- (from existing alias)	WB/WC/UC- (from existing alias)
/dev/mem mmap !SYNC flag no alias to this area and MTRR says WB	--	WB	WB
/dev/mem mmap !SYNC flag no alias to this area and MTRR says !WB	--	--	UC-

Advanced APIs for drivers

A. Exporting pages to users with remap_pfn_range, io_remap_pfn_range, vmf_insert_pfn.

Drivers wanting to export some pages to userspace do it by using mmap interface and a combination of:

1. pgprot_noncached()
2. io_remap_pfn_range() or remap_pfn_range() or vmf_insert_pfn()

With PAT support, a new API pgprot_writecombine is being added. So, drivers can continue to use the above sequence, with either pgprot_noncached() or pgprot_writecombine() in step 1, followed by step 2.

In addition, step 2 internally tracks the region as UC or WC in memtype list in order to ensure no conflicting mapping.

Note that this set of APIs only works with IO (non RAM) regions. If driver wants to export a RAM region, it has to do set_memory_uc() or set_memory_wc() as step 0 above and also track the usage of those pages and use set_memory_wb() before the page is freed to free pool.

MTRR effects on PAT / non-PAT setups

MTRR effects on PAT / non-PAT systems

The following table provides the effects of using write-combining MTRRs when using ioremap*() calls on x86 for both non-PAT and PAT systems. Ideally mtrr_add() usage will be phased out in favor of arch_phys_wc_add() which will be a no-op on PAT enabled systems. The region over which a arch_phys_wc_add() is made, should already have been ioremapped with WC attributes or PAT entries, this can be done by using ioremap_wc() / set_memory_wc(). Devices which combine areas of IO memory desired to remain uncacheable with areas where write-combining is desirable should consider use of ioremap_uc() followed by set_memory_wc() to white-list effective write-combined areas. Such use is nevertheless discouraged as the effective memory type is considered implementation defined, yet this strategy can be used as last resort on devices with size-constrained regions where otherwise MTRR write-combining would otherwise not be effective.

```
====  =======  ===  =========================  =====================
MTRR  Non-PAT  PAT  Linux ioremap value        Effective memory type
====  =======  ===  =========================  =====================
      PAT                                        Non-PAT |  PAT
      |PCD                                               |
      ||PWT                                              |
      |||                                                |
WC    000      WB   _PAGE_CACHE_MODE_WB             WC   |   WC
WC    001      WC   _PAGE_CACHE_MODE_WC             WC*  |   WC
WC    010      UC-  _PAGE_CACHE_MODE_UC_MINUS       WC*  |   UC
WC    011      UC   _PAGE_CACHE_MODE_UC             UC   |   UC
====  =======  ===  =========================  =====================

(*) denotes implementation defined and is discouraged
```

Note

-- in the above table mean "Not suggested usage for the API". Some of the --'s are strictly enforced by the kernel. Some others are not really enforced today, but may be enforced in future.

For ioremap and pci access through /sys or /proc - The actual type returned can be more restrictive, in case of any existing aliasing for that address. For example: If there is an existing uncached mapping, a new ioremap_wc can return uncached mapping in place of write-combine requested.

set_memory_[uc|wc|wt] and set_memory_wb should be used in pairs, where driver will first make a region uc, wc or wt and switch it back to wb after use.

Over time writes to /proc/mtrr will be deprecated in favor of using PAT based interfaces. Users writing to /proc/mtrr are suggested to use above interfaces.

Drivers should use ioremap_[uc|wc] to access PCI BARs with [uc|wc] access types.

Drivers should use set_memory_[uc|wc|wt] to set access type for RAM ranges.

PAT debugging

With CONFIG_DEBUG_FS enabled, PAT memtype list can be examined by:

```
# mount -t debugfs debugfs /sys/kernel/debug
# cat /sys/kernel/debug/x86/pat_memtype_list
PAT memtype list:
uncached-minus @ 0x7fadf000-0x7fae0000
uncached-minus @ 0x7fb19000-0x7fb1a000
uncached-minus @ 0x7fb1a000-0x7fb1b000
uncached-minus @ 0x7fb1b000-0x7fb1c000
uncached-minus @ 0x7fb1c000-0x7fb1d000
uncached-minus @ 0x7fb1d000-0x7fb1e000
uncached-minus @ 0x7fb1e000-0x7fb25000
uncached-minus @ 0x7fb25000-0x7fb26000
uncached-minus @ 0x7fb26000-0x7fb27000
uncached-minus @ 0x7fb27000-0x7fb28000
uncached-minus @ 0x7fb28000-0x7fb2e000
uncached-minus @ 0x7fb2e000-0x7fb2f000
uncached-minus @ 0x7fb2f000-0x7fb30000
uncached-minus @ 0x7fb31000-0x7fb32000
uncached-minus @ 0x80000000-0x90000000
```

This list shows physical address ranges and various PAT settings used to access those physical address ranges.

Another, more verbose way of getting PAT related debug messages is with "debugpat" boot parameter. With this parameter, various debug messages are printed to dmesg log.

PAT Initialization

The following table describes how PAT is initialized under various configurations. The PAT MSR must be updated by Linux in order to support WC and WT attributes. Otherwise, the PAT MSR has the value programmed in it by the firmware. Note, Xen enables WC attribute in the PAT MSR for guests.

MTRR	PAT	Call Sequence	PAT State	PAT MSR
E	E	MTRR -> PAT init	Enabled	OS
E	D	MTRR -> PAT init	Disabled	•
D	E	MTRR -> PAT disable	Disabled	BIOS
D	D	MTRR -> PAT disable	Disabled	•
•	np/E	PAT -> PAT disable	Disabled	BIOS
•	np/D	PAT -> PAT disable	Disabled	•
E	!P/E	MTRR -> PAT init	Disabled	BIOS
D	!P/E	MTRR -> PAT disable	Disabled	BIOS
!M	!P/E	MTRR stub -> PAT disable	Disabled	BIOS

Legend

E	Feature enabled in CPU
D	Feature disabled/unsupported in CPU
np	"nopat" boot option specified
!P	CONFIG_X86_PAT option unset
!M	CONFIG_MTRR option unset
Enabled	PAT state set to enabled
Disabled	PAT state set to disabled
OS	PAT initializes PAT MSR with OS setting
BIOS	PAT keeps PAT MSR with BIOS setting