

[PATCH] [5/34] i386: Enable support for fixed-range IORRs to keep RdMem & WrMem in sync

[PATCH] [5/34] i386: Enable support for fixed-range IORRs to keep RdMem & WrMem in sync

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-04/msg12091.html>

- *From:* Andi Kleen <ak@xxxxxxx>
 - *Date:* Mon, 30 Apr 2007 17:49:36 +0200 (CEST)
-

From: Bernhard Kaindl <bk@xxxxxxx>

If our copy of the MTRRs of the BSP has RdMem or WrMem set, and we are running on an AMD64/K8 system, the boot CPU must have had MtrrFixDramEn and MtrrFixDramModEn set (otherwise our RDMSR would have copied these bits cleared), so we set them on this CPU as well.

This allows us to keep the AMD64/K8 RdMem and WrMem bits in sync across the CPUs of SMP systems in order to fulfill the duty of system software to "initialize and maintain MTRR consistency across all processors." as written in the AMD and Intel manuals.

If an WRMSR instruction fails because MtrrFixDramModEn is not set, I expect that also the Intel-style MTRR bits are not updated.

AK: minor cleanup, moved MSR defines around

Signed-off-by: Bernhard Kaindl <bk@xxxxxxx>
Signed-off-by: Andi Kleen <ak@xxxxxxx>
Cc: Andrew Morton <akpm@xxxxxxxxxxxxxxxxxxxxxx>
Cc: Andi Kleen <ak@xxxxxxx>
Cc: Dave Jones <davej@xxxxxxxxxxxxxxxxxxxxxx>

arch/i386/kernel/cpu/mtrr/generic.c | 85 ++++++-----
include/asm-i386/msr-index.h | 5 ++
2 files changed, 65 insertions(+), 25 deletions(-)

Index: linux/arch/i386/kernel/cpu/mtrr/generic.c

```
=====  
--- linux.orig/arch/i386/kernel/cpu/mtrr/generic.c  
+++ linux/arch/i386/kernel/cpu/mtrr/generic.c  
@@ -20,6 +20,18 @@ struct mtrr_state {  
mtrr_type def_type;  
};
```

[PATCH] [5/34] i386: Enable support for fixed-range IORRs to keep RdMem & WrMem in sync

```
+struct fixed_range_block {
+ int base_msr; /* start address of an MTRR block */
+ int ranges; /* number of MTRRs in this block */
+};
+
+static struct fixed_range_block fixed_range_blocks[] = {
+ { MTRRfix64K_00000_MSR, 1 }, /* one 64k MTRR */
+ { MTRRfix16K_80000_MSR, 2 }, /* two 16k MTRRs */
+ { MTRRfix4K_C0000_MSR, 8 }, /* eight 4k MTRRs */
+ { }
+};
+
+static unsigned long smp_changes_mask;
+static struct mtrr_state mtrr_state = { };

@@ -152,6 +164,44 @@ void mtrr_wrmsr(unsigned msr, unsigned a
smp_processor_id(), msr, a, b);
}

+/**
+ * Enable and allow read/write of extended fixed-range MTRR bits on K8 CPUs
+ * see AMD publication no. 24593, chapter 3.2.1 for more information
+ */
+static inline void k8_enable_fixed_iorrs(void)
+{
+ unsigned lo, hi;
+
+ rdmsr(MSR_K8_SYSCFG, lo, hi);
+ mtrr_wrmsr(MSR_K8_SYSCFG, lo
+ | K8_MTRRFIXRANGE_DRAM_ENABLE
+ | K8_MTRRFIXRANGE_DRAM_MODIFY, hi);
+}
+
+/**
+ * Checks and updates an fixed-range MTRR if it differs from the value it
+ * should have. If K8 extensions are wanted, update the K8 SYSCFG MSR also.
+ * see AMD publication no. 24593, chapter 7.8.1, page 233 for more information
+ * \param msr MSR address of the MTRR which should be checked and updated
+ * \param changed pointer which indicates whether the MTRR needed to be changed
+ * \param msrwords pointer to the MSR values which the MSR should have
+ */
+static void set_fixed_range(int msr, int * changed, unsigned int * msrwords)
+{
+ unsigned lo, hi;
+
+ rdmsr(msr, lo, hi);
+
+ if (lo != msrwords[0] || hi != msrwords[1]) {
+ if (boot_cpu_data.x86_vendor == X86_VENDOR_AMD &&
+ boot_cpu_data.x86 == 15 &&
```

[PATCH] [5/34] i386: Enable support for fixed-range IORRs to keep RdMem & WrMem in sync

```
+ ((msrwords[0] | msrwords[1]) & K8_MTRR_RDMEM_WRMEM_MASK))
+ k8_enable_fixed_iorrs();
+ mtrr_wrmsr(msr, msrwords[0], msrwords[1]);
+ *changed = TRUE;
+ }
+}
+
int generic_get_free_region(unsigned long base, unsigned long size, int replace_reg)
/* [SUMMARY] Get a free MTRR.
<base> The starting (base) address of the region.
@@ -201,36 +251,21 @@ static void generic_get_mtrr(unsigned in
*type = base_lo & 0xff;
}

+/**
+ * Checks and updates the fixed-range MTRRs if they differ from the saved set
+ * \param frs pointer to fixed-range MTRR values, saved by get_fixed_ranges()
+ */
static int set_fixed_ranges(mtrr_type * frs)
{
- unsigned int *p = (unsigned int *) frs;
+ unsigned long long *saved = (unsigned long long *) frs;
int changed = FALSE;
- int i;
- unsigned int lo, hi;
-
- rdmsr(MTRRfix64K_00000_MSR, lo, hi);
- if (p[0] != lo || p[1] != hi) {
- mtrr_wrmsr(MTRRfix64K_00000_MSR, p[0], p[1]);
- changed = TRUE;
- }
+ int block=-1, range;

- for (i = 0; i < 2; i++) {
- rdmsr(MTRRfix16K_80000_MSR + i, lo, hi);
- if (p[2 + i * 2] != lo || p[3 + i * 2] != hi) {
- mtrr_wrmsr(MTRRfix16K_80000_MSR + i, p[2 + i * 2],
- p[3 + i * 2]);
- changed = TRUE;
- }
- }
+ while (fixed_range_blocks[++block].ranges)
+ for (range=0; range < fixed_range_blocks[block].ranges; range++)
+ set_fixed_range(fixed_range_blocks[block].base_msr + range,
+ &changed, (unsigned int *) saved++);

- for (i = 0; i < 8; i++) {
- rdmsr(MTRRfix4K_C0000_MSR + i, lo, hi);
- if (p[6 + i * 2] != lo || p[7 + i * 2] != hi) {
- mtrr_wrmsr(MTRRfix4K_C0000_MSR + i, p[6 + i * 2],
- p[7 + i * 2]);
```

[PATCH] [5/34] i386: Enable support for fixed-range IORRs to keep RdMem & WrMem in sync 3

[PATCH] [5/34] i386: Enable support for fixed-range IORRs to keep RdMem & WrMem in sync

```
- changed = TRUE;
- }
- }
return changed;
}
```

Index: linux/include/asm-i386/msr-index.h

```
-----
--- linux.orig/include/asm-i386/msr-index.h
+++ linux/include/asm-i386/msr-index.h
@@ -87,6 +87,11 @@
#define MSR_K7_CLK_CTL 0xc001001b
#define MSR_K8_TOP_MEM2 0xc001001d
#define MSR_K8_SYSCFG 0xc0010010
+
+#define K8_MTRRFIXRANGE_DRAM_ENABLE 0x00040000 /* MtrrFixDramEn bit */
+#define K8_MTRRFIXRANGE_DRAM_MODIFY 0x00080000 /* MtrrFixDramModEn bit */
+#define K8_MTRR_RDMEM_WRMEM_MASK 0x18181818 /* Mask: RdMem|WrMem */
+
#define MSR_K7_HWCR 0xc0010015
#define MSR_K8_HWCR 0xc0010015
#define MSR_K7_FID_VID_CTL 0xc0010041
```

-
To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
the body of a message to majordomo@xxxxxxxxxxxxxxxxxxx
More majordomo info at <http://vger.kernel.org/majordomo-info.html>
Please read the FAQ at <http://www.tux.org/lkml/>