```
**********************************************************
   50351 Wed Jun 19 14:34:34 2013
new/usr/src/uts/i86pc/os/mp_startup.c
3832 AMD E721 workaround panics on KVM
Reviewed by: Marcel Telka <marcel@telka.sk>
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
 */
/*
 * Copyright (c) 2010, Intel Corporation.
 * All rights reserved.
 */
/*
 * Copyright (c) 2012, Joyent, Inc.  All rights reserved.
 * Copyright 2013 Nexenta Systems, Inc.  All rights reserved.
 */

#include <sys/types.h>
#include <sys/thread.h>
#include <sys/cpuvar.h>
#include <sys/cpu.h>
#include <sys/t_lock.h>
#include <sys/param.h>
#include <sys/proc.h>
#include <sys/disp.h>
#include <sys/class.h>
#include <sys/cmn_err.h>
#include <sys/debug.h>
#include <sys/note.h>
#include <sys/asm_linkage.h>
#include <sys/x_call.h>
#include <sys/systm.h>
#include <sys/var.h>
#include <sys/vtrace.h>
#include <vm/hat.h>
#include <vm/as.h>
#include <vm/seg_kmem.h>
#include <vm/seg_kp.h>
#include <sys/segments.h>
#include <sys/kmem.h>
#include <sys/stack.h>
#include <sys/smp_impldefs.h>
#include <sys/x86_archext.h>
#include <sys/machsystm.h>
```



```
#include <sys/traptrace.h>
#include <sys/clock.h>
#include <sys/cpc_impl.h>
#include <sys/pg.h>
#include <sys/cmt.h>
#include <sys/dtrace.h>
#include <sys/archsystm.h>
#include <sys/fp.h>
#include <sys/reboot.h>
#include <sys/kdi_machimpl.h>
#include <vm/hat_i86.h>
#include <vm/vm_dep.h>
#include <sys/memnode.h>
#include <sys/pci_cfgspace.h>
#include <sys/mach_mmu.h>
#include <sys/sysmacros.h>
#if defined(__xpv)
#include <sys/hypervisor.h>
#endif
#include <sys/cpu_module.h>
#include <sys/ontrap.h>

struct cpu      cpus[1];                        /* CPU data */
struct cpu      *cpu[NCPU] = {&cpus[0]};        /* pointers to all CPUs */
struct cpu      *cpu_free_list;                 /* list for released CPUs */
cpu_core_t      cpu_core[NCPU];                 /* cpu_core structures */

#define cpu_next_free   cpu_prev

/*
 * Useful for disabling MP bring-up on a MP capable system.
 */
int use_mp = 1;

/*
 * to be set by a PSM to indicate what cpus
 * are sitting around on the system.
 */
cpuset_t mp_cpus;

/*
 * This variable is used by the hat layer to decide whether or not
 * critical sections are needed to prevent race conditions.  For sun4m,
 * this variable is set once enough MP initialization has been done in
 * order to allow cross calls.
 */
int flushes_require_xcalls;

cpuset_t cpu_ready_set;         /* initialized in startup() */

static void mp_startup_boot(void);
static void mp_startup_hotplug(void);

static void cpu_sep_enable(void);
static void cpu_sep_disable(void);
static void cpu_asysc_enable(void);
static void cpu_asysc_disable(void);

/*
 * Init CPU info - get CPU type info for processor_info system call.
 */
void
init_cpu_info(struct cpu *cp)
{
        processor_info_t *pi = &cp->cpu_type_info;
```

```
127         /*
128          * Get clock-frequency property for the CPU.
129          */
130         pi->pi_clock = cpu_freq;

132         /*
133          * Current frequency in Hz.
134          */
135         cp->cpu_curr_clock = cpu_freq_hz;

137         /*
138          * Supported frequencies.
139          */
140         if (cp->cpu_supp_freqs == NULL) {
141                 cpu_set_supp_freqs(cp, NULL);
142         }

144         (void) strcpy(pi->pi_processor_type, "i386");
145         if (fpu_exists)
146                 (void) strcpy(pi->pi_fputypes, "i387 compatible");

148         cp->cpu_idstr = kmem_zalloc(CPU_IDSTRLEN, KM_SLEEP);
149         cp->cpu_brandstr = kmem_zalloc(CPU_IDSTRLEN, KM_SLEEP);

151         /*
152          * If called for the BSP, cp is equal to current CPU.
153          * For non-BSPs, cpuid info of cp is not ready yet, so use cpuid info
154          * of current CPU as default values for cpu_idstr and cpu_brandstr.
155          * They will be corrected in mp_startup_common() after cpuid_pass1()
156          * has been invoked on target CPU.
157          */
158         (void) cpuid_getidstr(CPU, cp->cpu_idstr, CPU_IDSTRLEN);
159         (void) cpuid_getbrandstr(CPU, cp->cpu_brandstr, CPU_IDSTRLEN);
160 }
_____unchanged_portion_omitted_

780 uint_t
781 workaround_errata(struct cpu *cpu)
782 {
783         uint_t missing = 0;

785         ASSERT(cpu == CPU);

787         /*LINTED*/
788         if (cpuid_opteron_erratum(cpu, 88) > 0) {
789                 /*
790                  * SWAPGS May Fail To Read Correct GS Base
791                  */
792 #if defined(OPTERON_ERRATUM_88)
793                 /*
794                  * The workaround is an mfence in the relevant assembler code
795                  */
796                 opteron_erratum_88++;
797 #else
798                 workaround_warning(cpu, 88);
799                 missing++;
800 #endif
801         }

803         if (cpuid_opteron_erratum(cpu, 91) > 0) {
804                 /*
805                  * Software Prefetches May Report A Page Fault
806                  */
807 #if defined(OPTERON_ERRATUM_91)
808                 /*
809                  * fix is in trap.c
```



```
810                  */
811                 opteron_erratum_91++;
812 #else
813                 workaround_warning(cpu, 91);
814                 missing++;
815 #endif
816         }

818         if (cpuid_opteron_erratum(cpu, 93) > 0) {
819                 /*
820                  * RSM Auto-Halt Restart Returns to Incorrect RIP
821                  */
822 #if defined(OPTERON_ERRATUM_93)
823                 /*
824                  * fix is in trap.c
825                  */
826                 opteron_erratum_93++;
827 #else
828                 workaround_warning(cpu, 93);
829                 missing++;
830 #endif
831         }

833         /*LINTED*/
834         if (cpuid_opteron_erratum(cpu, 95) > 0) {
835                 /*
836                  * RET Instruction May Return to Incorrect EIP
837                  */
838 #if defined(OPTERON_ERRATUM_95)
839 #if defined(_LP64)
840                 /*
841                  * Workaround this by ensuring that 32-bit user code and
842                  * 64-bit kernel code never occupy the same address
843                  * range mod 4G.
844                  */
845                 if (_userlimit32 > 0xc0000000ul)
846                         *(uintptr_t *)&_userlimit32 = 0xc0000000ul;

848                 /*LINTED*/
849                 ASSERT((uint32_t)COREHEAP_BASE == 0xc0000000u);
850                 opteron_erratum_95++;
851 #endif  /* _LP64 */
852 #else
853                 workaround_warning(cpu, 95);
854                 missing++;
855 #endif
856         }

858         if (cpuid_opteron_erratum(cpu, 100) > 0) {
859                 /*
860                  * Compatibility Mode Branches Transfer to Illegal Address
861                  */
862 #if defined(OPTERON_ERRATUM_100)
863                 /*
864                  * fix is in trap.c
865                  */
866                 opteron_erratum_100++;
867 #else
868                 workaround_warning(cpu, 100);
869                 missing++;
870 #endif
871         }

873         /*LINTED*/
874         if (cpuid_opteron_erratum(cpu, 108) > 0) {
875                 /*
```

```
876                  * CPUID Instruction May Return Incorrect Model Number In
877                  * Some Processors
878                  */
879 #if defined(OPTERON_ERRATUM_108)
880                 /*
881                  * (Our cpuid-handling code corrects the model number on
882                  * those processors)
883                  */
884 #else
885                 workaround_warning(cpu, 108);
886                 missing++;
887 #endif
888         }

890         /*LINTED*/
891         if (cpuid_opteron_erratum(cpu, 109) > 0) do {
892                 /*
893                  * Certain Reverse REP MOVS May Produce Unpredictable Behavior
894                  */
895 #if defined(OPTERON_ERRATUM_109)
896                 /*
897                  * The "workaround" is to print a warning to upgrade the BIOS
898                  */
899                 uint64_t value;
900                 const uint_t msr = MSR_AMD_PATCHLEVEL;
901                 int err;

903                 if ((err = checked_rdmsr(msr, &value)) != 0) {
904                         msr_warning(cpu, "rd", msr, err);
905                         workaround_warning(cpu, 109);
906                         missing++;
907                 }
908                 if (value == 0)
909                         opteron_erratum_109++;
910 #else
911                 workaround_warning(cpu, 109);
912                 missing++;
913 #endif
914         /*CONSTANTCONDITION*/
915         } while (0);

917         /*LINTED*/
918         if (cpuid_opteron_erratum(cpu, 121) > 0) {
919                 /*
920                  * Sequential Execution Across Non_Canonical Boundary Caused
921                  * Processor Hang
922                  */
923 #if defined(OPTERON_ERRATUM_121)
924 #if defined(_LP64)
925                 /*
926                  * Erratum 121 is only present in long (64 bit) mode.
927                  * Workaround is to include the page immediately before the
928                  * va hole to eliminate the possibility of system hangs due to
929                  * sequential execution across the va hole boundary.
930                  */
931                 if (opteron_erratum_121)
932                         opteron_erratum_121++;
933                 else {
934                         if (hole_start) {
935                                 hole_start -= PAGESIZE;
936                         } else {
937                                 /*
938                                  * hole_start not yet initialized by
939                                  * mmu_init. Initialize hole_start
940                                  * with value to be subtracted.
941                                  */
```



```
942                                 hole_start = PAGESIZE;
943                         }
944                         opteron_erratum_121++;
945                 }
946 #endif  /* _LP64 */
947 #else
948                 workaround_warning(cpu, 121);
949                 missing++;
950 #endif
951         }

953         /*LINTED*/
954         if (cpuid_opteron_erratum(cpu, 122) > 0) do {
955                 /*
956                  * TLB Flush Filter May Cause Coherency Problem in
957                  * Multiprocessor Systems
958                  */
959 #if defined(OPTERON_ERRATUM_122)
960                 uint64_t value;
961                 const uint_t msr = MSR_AMD_HWCR;
962                 int error;

964                 /*
965                  * Erratum 122 is only present in MP configurations (multi-core
966                  * or multi-processor).
967                  */
968 #if defined(__xpv)
969                 if (!DOMAIN_IS_INITDOMAIN(xen_info))
970                         break;
971                 if (!opteron_erratum_122 && xpv_nr_phys_cpus() == 1)
972                         break;
973 #else
974                 if (!opteron_erratum_122 && opteron_get_nnodes() == 1 &&
975                     cpuid_get_ncpu_per_chip(cpu) == 1)
976                         break;
977 #endif
978                 /* disable TLB Flush Filter */

980                 if ((error = checked_rdmsr(msr, &value)) != 0) {
981                         msr_warning(cpu, "rd", msr, error);
982                         workaround_warning(cpu, 122);
983                         missing++;
984                 } else {
985                         value |= (uint64_t)AMD_HWCR_FFDIS;
986                         if ((error = checked_wrmsr(msr, value)) != 0) {
987                                 msr_warning(cpu, "wr", msr, error);
988                                 workaround_warning(cpu, 122);
989                                 missing++;
990                         }
991                 }
992                 opteron_erratum_122++;
993 #else
994                 workaround_warning(cpu, 122);
995                 missing++;
996 #endif
997         /*CONSTANTCONDITION*/
998         } while (0);

1000        /*LINTED*/
1001        if (cpuid_opteron_erratum(cpu, 123) > 0) do {
1002                /*
1003                 * Bypassed Reads May Cause Data Corruption of System Hang in
1004                 * Dual Core Processors
1005                 */
1006 #if defined(OPTERON_ERRATUM_123)
1007                 uint64_t value;
```

```
1008                const uint_t msr = MSR_AMD_PATCHLEVEL;
1009                int err;
1010
1011                /*
1012                 * Erratum 123 applies only to multi-core cpus.
1013                 */
1014                if (cpuid_get_ncpu_per_chip(cpu) < 2)
1015                        break;
1016 #if defined(__xpv)
1017                if (!DOMAIN_IS_INITDOMAIN(xen_info))
1018                        break;
1019 #endif
1020                /*
1021                 * The "workaround" is to print a warning to upgrade the BIOS
1022                 */
1023                if ((err = checked_rdmsr(msr, &value)) != 0) {
1024                        msr_warning(cpu, "rd", msr, err);
1025                        workaround_warning(cpu, 123);
1026                        missing++;
1027                }
1028                if (value == 0)
1029                        opteron_erratum_123++;
1030 #else
1031                workaround_warning(cpu, 123);
1032                missing++;
1033
1034 #endif
1035        /*CONSTANTCONDITION*/
1036        } while (0);
1037
1038        /*LINTED*/
1039        if (cpuid_opteron_erratum(cpu, 131) > 0) do {
1040                /*
1041                 * Multiprocessor Systems with Four or More Cores May Deadlock
1042                 * Waiting for a Probe Response
1043                 */
1044 #if defined(OPTERON_ERRATUM_131)
1045                uint64_t nbcfg;
1046                const uint_t msr = MSR_AMD_NB_CFG;
1047                const uint64_t wabits =
1048                    AMD_NB_CFG_SRQ_HEARTBEAT | AMD_NB_CFG_SRQ_SPR;
1049                int error;
1050
1051                /*
1052                 * Erratum 131 applies to any system with four or more cores.
1053                 */
1054                if (opteron_erratum_131)
1055                        break;
1056 #if defined(__xpv)
1057                if (!DOMAIN_IS_INITDOMAIN(xen_info))
1058                        break;
1059                if (xpv_nr_phys_cpus() < 4)
1060                        break;
1061 #else
1062                if (opteron_get_nnodes() * cpuid_get_ncpu_per_chip(cpu) < 4)
1063                        break;
1064 #endif
1065                /*
1066                 * Print a warning if neither of the workarounds for
1067                 * erratum 131 is present.
1068                 */
1069                if ((error = checked_rdmsr(msr, &nbcfg)) != 0) {
1070                        msr_warning(cpu, "rd", msr, error);
1071                        workaround_warning(cpu, 131);
1072                        missing++;
1073                } else if ((nbcfg & wabits) == 0) {
```



```
1074                        opteron_erratum_131++;
1075                } else {
1076                        /* cannot have both workarounds set */
1077                        ASSERT((nbcfg & wabits) != wabits);
1078                }
1079 #else
1080                workaround_warning(cpu, 131);
1081                missing++;
1082 #endif
1083        /*CONSTANTCONDITION*/
1084        } while (0);
1085
1086        /*
1087         * This isn't really an erratum, but for convenience the
1088         * detection/workaround code lives here and in cpuid_opteron_erratum.
1089         */
1090        if (cpuid_opteron_erratum(cpu, 6336786) > 0) {
1091 #if defined(OPTERON_WORKAROUND_6336786)
1092                /*
1093                 * Disable C1-Clock ramping on multi-core/multi-processor
1094                 * K8 platforms to guard against TSC drift.
1095                 */
1096                if (opteron_workaround_6336786) {
1097                        opteron_workaround_6336786++;
1098 #if defined(__xpv)
1099                } else if ((DOMAIN_IS_INITDOMAIN(xen_info) &&
1100                    xpv_nr_phys_cpus() > 1) ||
1101                    opteron_workaround_6336786_UP) {
1102                        /*
1103                         * XXPV Hmm.  We can't walk the Northbridges on
1104                         *      the hypervisor; so just complain and drive
1105                         *      on.  This probably needs to be fixed in
1106                         *      the hypervisor itself.
1107                         */
1108                        opteron_workaround_6336786++;
1109                        workaround_warning(cpu, 6336786);
1110 #else   /* __xpv */
1111                } else if ((opteron_get_nnodes() *
1112                    cpuid_get_ncpu_per_chip(cpu) > 1) ||
1113                    opteron_workaround_6336786_UP) {
1114
1115                        uint_t  node, nnodes;
1116                        uint8_t data;
1117
1118                        nnodes = opteron_get_nnodes();
1119                        for (node = 0; node < nnodes; node++) {
1120                                /*
1121                                 * Clear PMM7[1:0] (function 3, offset 0x87)
1122                                 * Northbridge device is the node id + 24.
1123                                 */
1124                                data = pci_getb_func(0, node + 24, 3, 0x87);
1125                                data &= 0xFC;
1126                                pci_putb_func(0, node + 24, 3, 0x87, data);
1127                        }
1128                        opteron_workaround_6336786++;
1129 #endif  /* __xpv */
1130                }
1131 #else
1132                workaround_warning(cpu, 6336786);
1133                missing++;
1134 #endif
1135        }
1136
1137        /*LINTED*/
1138        /*
1139         * Mutex primitives don't work as expected.
```

```
1140          */
1141         if (cpuid_opteron_erratum(cpu, 6323525) > 0) {
1142 #if defined(OPTERON_WORKAROUND_6323525)
1143                 /*
1144                  * This problem only occurs with 2 or more cores. If bit in
1145                  * MSR_AMD_BU_CFG set, then not applicable. The workaround
1146                  * is to patch the semaphone routines with the lfence
1147                  * instruction to provide necessary load memory barrier with
1148                  * possible subsequent read-modify-write ops.
1149                  *
1150                  * It is too early in boot to call the patch routine so
1151                  * set erratum variable to be done in startup_end().
1152                  */
1153                 if (opteron_workaround_6323525) {
1154                         opteron_workaround_6323525++;
1155 #if defined(__xpv)
1156                 } else if (is_x86_feature(x86_featureset, X86FSET_SSE2)) {
1157                         if (DOMAIN_IS_INITDOMAIN(xen_info)) {
1158                                 /*
1159                                  * XXPV Use dom0_msr here when extended
1160                                  *      operations are supported?
1161                                  */
1162                                 if (xpv_nr_phys_cpus() > 1)
1163                                         opteron_workaround_6323525++;
1164                         } else {
1165                                 /*
1166                                  * We have no way to tell how many physical
1167                                  * cpus there are, or even if this processor
1168                                  * has the problem, so enable the workaround
1169                                  * unconditionally (at some performance cost).
1170                                  */
1171                                 opteron_workaround_6323525++;
1172                         }
1173 #else   /* __xpv */
1174                 } else if (is_x86_feature(x86_featureset, X86FSET_SSE2) &&
1175                     ((opteron_get_nnodes() *
1176                     cpuid_get_ncpu_per_chip(cpu)) > 1)) {
1177                         if ((xrdmsr(MSR_AMD_BU_CFG) & (UINT64_C(1) << 33)) == 0)
1178                                 opteron_workaround_6323525++;
1179 #endif  /* __xpv */
1180                 }
1181 #else
1182                 workaround_warning(cpu, 6323525);
1183                 missing++;
1184 #endif
1185         }

1187         missing += do_erratum_298(cpu);

1189         if (cpuid_opteron_erratum(cpu, 721) > 0) {
1190 #if defined(OPTERON_ERRATUM_721)
1191                 on_trap_data_t otd;

1193                 if (!on_trap(&otd, OT_DATA_ACCESS))
1194                         wrmsr(MSR_AMD_DE_CFG,
1195                             rdmsr(MSR_AMD_DE_CFG) | AMD_DE_CFG_E721);
1196                 no_trap();

1189                 wrmsr(MSR_AMD_DE_CFG, rdmsr(MSR_AMD_DE_CFG) | AMD_DE_CFG_E721);
1198                 opteron_erratum_721++;
1199 #else
1200                 workaround_warning(cpu, 721);
1201                 missing++;
1202 #endif
1203         }
```



```
1205 #ifdef __xpv
1206         return (0);
1207 #else
1208         return (missing);
1209 #endif
1210 }
______unchanged_portion_omitted_
```