

[patch 06/16] Add support for IA64 platforms to KGDB

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-08/7296.html>

From: Tom Rini (trini_at_kernel.crashing.org)

Date: 08/29/05

Date: Mon, 29 Aug 2005 09:10:04 -0700

To: akpm@osdl.org

This is support of the IA64 arch for KGDB. This is primarily the work of Robert Picco. There are also some IA64 changes in the serial patch so that IA64 can pass in IRQ and iomembase, and all of this is documented in the DocBook files.

```

---
linux-2.6.13-trini/arch/ia64/kernel/Makefile      |    1
linux-2.6.13-trini/arch/ia64/kernel/entry.S      |    4
linux-2.6.13-trini/arch/ia64/kernel/ivt.S        |   15
linux-2.6.13-trini/arch/ia64/kernel/kgdb-jmp.S   |  238 ++++
linux-2.6.13-trini/arch/ia64/kernel/kgdb.c       | 1038 +++++
linux-2.6.13-trini/arch/ia64/kernel/mca.c        |   10
linux-2.6.13-trini/arch/ia64/kernel/process.c    |    6
linux-2.6.13-trini/arch/ia64/kernel/smp.c        |   17
linux-2.6.13-trini/arch/ia64/kernel/traps.c      |   36
linux-2.6.13-trini/arch/ia64/kernel/unwind.c     |   87 +
linux-2.6.13-trini/arch/ia64/mm/extable.c        |    6
linux-2.6.13-trini/arch/ia64/mm/fault.c          |    5
linux-2.6.13-trini/include/asm-ia64/kgdb.h       |   37
linux-2.6.13-trini/lib/Kconfig.debug             |    2
14 files changed, 1497 insertions(+), 5 deletions(-)
diff -puN arch/ia64/kernel/entry.S~ia64-lite arch/ia64/kernel/entry.S
--- linux-2.6.13/arch/ia64/kernel/entry.S~ia64-lite      2005-08-08 12:18:23.000000000 -0700
+++ linux-2.6.13-trini/arch/ia64/kernel/entry.S 2005-08-08 12:18:23.000000000 -0700
@@ -929,9 +929,9 @@ GLOBAL_ENTRY(ia64_leave_kernel)
     shr.u r18=r19,16          // get byte size of existing "dirty" partition
     ;;
     mov r16=ar.bsp           // get existing backing store pointer
-   addl r17=THIS_CPU(ia64_phys_stacked_size_p8),r0
+(pUStk)   addl r17=THIS_CPU(ia64_phys_stacked_size_p8),r0
     ;;
-   ld4 r17=[r17]             // r17 = cpu_data->phys_stacked_size_p8
+(pUStk)   ld4 r17=[r17]             // r17 = cpu_data->phys_stacked_size_p8
(pKStk)   br.cond.dpnt skip_rbs_switch

/*
diff -puN arch/ia64/kernel/ivt.S~ia64-lite arch/ia64/kernel/ivt.S
--- linux-2.6.13/arch/ia64/kernel/ivt.S~ia64-lite      2005-08-08 12:18:23.000000000 -0700
+++ linux-2.6.13-trini/arch/ia64/kernel/ivt.S 2005-08-08 12:18:23.000000000 -0700
@@ -69,6 +69,13 @@
 # define DBG_FAULT(i)
 #endif

```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```
+ #ifdef CONFIG_KGDB
+ #define KGDB_ENABLE_PSR_DB mov r31=psr;; movl r30=IA64_PSR_DB;; or r31=r31,r30;; \
+ mov psr.l=r31;; srlz.i;;
+ #else
+ #define KGDB_ENABLE_PSR_DB
+ #endif
+
+ #define MINSTATE_VIRT /* needed by minstate.h */
+ #include "minstate.h"

@@ -479,6 +486,7 @@ ENTRY(page_fault)
    movl r14=ia64_leave_kernel
    ;;
    SAVE_REST
+   KGDB_ENABLE_PSR_DB
    mov rp=r14
    ;;
    adds out2=16,r12 // out2 = pointer to pt_regs
@@ -820,6 +828,7 @@ ENTRY(interrupt)
    srlz.i // ensure everybody knows psr.ic is back on
    ;;
    SAVE_REST
+   KGDB_ENABLE_PSR_DB
    ;;
    alloc r14=ar.pfs,0,0,2,0 // must be first in an insn group
    mov out0=cr.ivr // pass cr.ivr as first arg
@@ -1066,6 +1075,7 @@ ENTRY(non_syscall)
    movl r15=ia64_leave_kernel
    ;;
    SAVE_REST
+   KGDB_ENABLE_PSR_DB
    mov rp=r15
    ;;
    br.call.sptk.many b6=ia64_bad_break // avoid WAW on CFM and ignore return addr
@@ -1099,6 +1109,7 @@ ENTRY(dispatch_unaligned_handler)
    adds r3=8,r2 // set up second base pointer
    ;;
    SAVE_REST
+   KGDB_ENABLE_PSR_DB
    movl r14=ia64_leave_kernel
    ;;
    mov rp=r14
@@ -1141,6 +1152,10 @@ ENTRY(dispatch_to_fault_handler)
    adds r3=8,r2 // set up second base pointer for SAVE_REST
    ;;
    SAVE_REST
+   cmp.eq p6,p0=29,out0
+ (p6) br.cond.spnt 1f;; // debug_vector
+   KGDB_ENABLE_PSR_DB
+1:
    movl r14=ia64_leave_kernel
    ;;
    mov rp=r14
diff -puN /dev/null arch/ia64/kernel/kgdb.c
--- /dev/null 2005-08-08 08:07:04.272443000 -0700
+++ linux-2.6.13-trini/arch/ia64/kernel/kgdb.c 2005-08-08 12:18:23.000000000 -0700
@@ -0,0 +1,1038 @@
+/*
+ *
+ * This program is free software; you can redistribute it and/or modify it
+ * under the terms of the GNU General Public License as published by the
```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```
+ * Free Software Foundation; either version 2, or (at your option) any
+ * later version.
+ *
+ * This program is distributed in the hope that it will be useful, but
+ * WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
+ * General Public License for more details.
+ *
+ */
+
+ * Copyright (C) 2000-2001 VERITAS Software Corporation.
+ */
+/*
+ * Contributor:      Lake Stevens Instrument Division$
+ * Written by:      Glenn Engel $
+ * Updated by:      Amit Kale<akale@veritas.com>
+ * Modified for 386 by Jim Kingdon, Cygnus Support.
+ * Original kgdb, compatibility with 2.1.xx kernel by David Grothe <dave@gcom.com>
+ */
+
+#include <linux/string.h>
+#include <linux/kernel.h>
+#include <linux/sched.h>
+#include <linux/smp.h>
+#include <linux/spinlock.h>
+#include <linux/delay.h>
+#include <asm/system.h>
+#include <asm/ptrace.h>          /* for linux pt_regs struct */
+#include <asm/unwind.h>
+#include <asm/rse.h>
+#include <linux/kgdb.h>
+#include <linux/init.h>
+#include <asm/cacheflush.h>
+
+#define NUM_REGS 590
+#define REGISTER_BYTES (NUM_REGS*8+128*8)
+#define REGISTER_BYTE(N) (((N) * 8) \
+ + ((N) <= IA64_FR0_REGNUM ? 0 : 8 * ((N) > IA64_FR127_REGNUM) ? 128 : (N) - IA64_FR0_REGNUM))
+#define REGISTER_SIZE(N) (((N) >= IA64_FR0_REGNUM && (N) <= IA64_FR127_REGNUM) ? 16 : 8)
+#define IA64_GRO_REGNUM      0
+#define IA64_FR0_REGNUM      128
+#define IA64_FR127_REGNUM    (IA64_FR0_REGNUM+127)
+#define IA64_PRO_REGNUM      256
+#define IA64_BR0_REGNUM      320
+#define IA64_VFP_REGNUM      328
+#define IA64_PR_REGNUM       330
+#define IA64_IP_REGNUM       331
+#define IA64_PSR_REGNUM      332
+#define IA64_CFM_REGNUM      333
+#define IA64_AR0_REGNUM      334
+#define IA64_NAT0_REGNUM     462
+#define IA64_NAT31_REGNUM    (IA64_NAT0_REGNUM+31)
+#define IA64_NAT32_REGNUM    (IA64_NAT0_REGNUM+32)
+#define IA64_RSC_REGNUM      (IA64_AR0_REGNUM+16)
+#define IA64_BSP_REGNUM      (IA64_AR0_REGNUM+17)
+#define IA64_BSPSTORE_REGNUM (IA64_AR0_REGNUM+18)
+#define IA64_RNAT_REGNUM     (IA64_AR0_REGNUM+19)
+#define IA64_FCR_REGNUM      (IA64_AR0_REGNUM+21)
+#define IA64_EFLAG_REGNUM    (IA64_AR0_REGNUM+24)
+#define IA64_CSD_REGNUM      (IA64_AR0_REGNUM+25)
+#define IA64_SSD_REGNUM      (IA64_AR0_REGNUM+26)
```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```

#define IA64_CFLG_REGNUM      (IA64_AR0_REGNUM+27)
#define IA64_FSR_REGNUM      (IA64_AR0_REGNUM+28)
#define IA64_FIR_REGNUM      (IA64_AR0_REGNUM+29)
#define IA64_FDR_REGNUM      (IA64_AR0_REGNUM+30)
#define IA64_CCV_REGNUM      (IA64_AR0_REGNUM+32)
#define IA64_UNAT_REGNUM     (IA64_AR0_REGNUM+36)
#define IA64_FPSR_REGNUM     (IA64_AR0_REGNUM+40)
#define IA64_ITC_REGNUM      (IA64_AR0_REGNUM+44)
#define IA64_PFS_REGNUM      (IA64_AR0_REGNUM+64)
#define IA64_LC_REGNUM       (IA64_AR0_REGNUM+65)
#define IA64_EC_REGNUM       (IA64_AR0_REGNUM+66)
+
#define REGISTER_INDEX(N)    (REGISTER_BYTE(N) / sizeof (unsigned long))
+
#define ptoff(V)              ((unsigned int) &((struct pt_regs *)0x0)->V)
+struct reg_to_ptreg_index {
+    unsigned int reg;
+    unsigned int ptregoff;
+};
+struct reg_to_ptreg_index gr_reg_to_ptreg_index[] = {
+    {IA64_GRO_REGNUM + 8, ptoff(r8)},
+    {IA64_GRO_REGNUM + 9, ptoff(r9)},
+    {IA64_GRO_REGNUM + 10, ptoff(r10)},
+    {IA64_GRO_REGNUM + 11, ptoff(r11)},
+    {IA64_GRO_REGNUM + 1, ptoff(r1)},
+    {IA64_GRO_REGNUM + 12, ptoff(r12)},
+    {IA64_GRO_REGNUM + 13, ptoff(r13)},
+    {IA64_GRO_REGNUM + 14, ptoff(r14)},
+    {IA64_GRO_REGNUM + 15, ptoff(r15)},
+};
+
+struct reg_to_ptreg_index br_reg_to_ptreg_index[] = {
+    {IA64_BR0_REGNUM, ptoff(b0)},
+    {IA64_BR0_REGNUM + 6, ptoff(b6)},
+    {IA64_BR0_REGNUM + 7, ptoff(b7)},
+};
+
+extern atomic_t cpu_doing_single_step;
+
+void kgdb_get_reg(char *outbuffer, int regnum, struct unw_frame_info *info,
+    struct pt_regs *ptregs)
+{
+    unsigned long reg, size = 0, *mem = &reg;
+    char nat;
+    struct ia64_fpreg freg;
+    int i;
+
+    if ((regnum >= IA64_GRO_REGNUM && regnum <= (IA64_GRO_REGNUM + 1)) ||
+        (regnum >= (IA64_GRO_REGNUM + 4) && regnum <= (IA64_GRO_REGNUM + 7))
+        || (regnum >= (IA64_GRO_REGNUM + 16)
+            && regnum <= (IA64_GRO_REGNUM + 31))) {
+        unw_access_gr(info, regnum - IA64_GRO_REGNUM, &reg, &nat, 0);
+        size = sizeof(reg);
+    } else
+        if ((regnum >= (IA64_GRO_REGNUM + 2)
+            && regnum <= (IA64_GRO_REGNUM + 3))
+            || (regnum >= (IA64_GRO_REGNUM + 8)
+                && regnum <= (IA64_GRO_REGNUM + 15))) {
+            if (ptregs) {
+                for (i = 0; i < (sizeof(gr_reg_to_ptreg_index) /
+                    sizeof(gr_reg_to_ptreg_index[0])); i++)
+                    if (gr_reg_to_ptreg_index[0].reg == regnum) {

```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```

+         reg = *((unsigned long *)
+             ((void *)ptregs) +
+             gr_reg_to_ptreg_index[i].ptregoff));
+         break;
+     }
+ } else
+     unw_access_gr(info, regnum - IA64_GRO_REGNUM, &reg,
+                 &nat, 0);
+     size = sizeof(reg);
+ } else if (regnum >= IA64_BR0_REGNUM && regnum <= (IA64_BR0_REGNUM + 7))
+     switch (regnum) {
+     case IA64_BR0_REGNUM:
+     case IA64_BR0_REGNUM + 6:
+     case IA64_BR0_REGNUM + 7:
+         if (ptregs) {
+             for (i = 0; i < (sizeof(br_reg_to_ptreg_index) /
+                 sizeof(br_reg_to_ptreg_index
+                     [0])); i++)
+                 if (br_reg_to_ptreg_index[i].reg ==
+                     regnum) {
+                     reg = *((unsigned long *)
+                         ((void *)ptregs) +
+                         br_reg_to_ptreg_index
+                             [i].ptregoff));
+                     break;
+                 }
+         } else
+             unw_access_br(info, regnum - IA64_BR0_REGNUM,
+                 &reg, 0);
+         size = sizeof(reg);
+         break;
+     case IA64_BR0_REGNUM + 1:
+     case IA64_BR0_REGNUM + 2:
+     case IA64_BR0_REGNUM + 3:
+     case IA64_BR0_REGNUM + 4:
+     case IA64_BR0_REGNUM + 5:
+         unw_access_br(info, regnum - IA64_BR0_REGNUM, &reg, 0);
+         size = sizeof(reg);
+         break;
+ } else if (regnum >= IA64_FR0_REGNUM
+     && regnum <= (IA64_FR0_REGNUM + 127))
+     switch (regnum) {
+     case IA64_FR0_REGNUM + 6:
+     case IA64_FR0_REGNUM + 7:
+     case IA64_FR0_REGNUM + 8:
+     case IA64_FR0_REGNUM + 9:
+     case IA64_FR0_REGNUM + 10:
+     case IA64_FR0_REGNUM + 11:
+     case IA64_FR0_REGNUM + 12:
+         if (!ptregs)
+             unw_access_fr(info, regnum - IA64_FR0_REGNUM,
+                 &freg, 0);
+         else {
+             freg =
+                 *(&ptregs->f6 +
+                     (regnum - (IA64_FR0_REGNUM + 6)));
+         }
+         size = sizeof(freg);
+         mem = (unsigned long *)&freg;
+         break;
+     default:
+         unw_access_fr(info, regnum - IA64_FR0_REGNUM, &freg, 0);

```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```
+         break;
+     } else if (regnum == IA64_IP_REGNUM) {
+         if (!ptregs)
+             unw_get_ip(info, &reg);
+         else
+             reg = ptregs->cr_iip;
+         size = sizeof(reg);
+     } else if (regnum == IA64_CFM_REGNUM) {
+         if (!ptregs)
+             unw_get_cfm(info, &reg);
+         else
+             reg = ptregs->cr_ifs;
+         size = sizeof(reg);
+     } else if (regnum == IA64_PSR_REGNUM) {
+         if (!ptregs && kgdb_usethread)
+             ptregs = (struct pt_regs *)
+                 ((unsigned long)kgdb_usethread + IA64_STK_OFFSET) -
+                 1;
+         if (ptregs)
+             reg = ptregs->cr_ipsr;
+         size = sizeof(reg);
+     } else if (regnum == IA64_PR_REGNUM) {
+         if (ptregs)
+             reg = ptregs->pr;
+         else
+             unw_access_pr(info, &reg, 0);
+         size = sizeof(reg);
+     } else if (regnum == IA64_BSP_REGNUM) {
+         unw_get_bsp(info, &reg);
+         size = sizeof(reg);
+     } else if (regnum >= IA64_AR0_REGNUM && regnum <= IA64_EC_REGNUM)
+         switch (regnum) {
+             case IA64_CSD_REGNUM:
+                 if (ptregs)
+                     reg = ptregs->ar_csd;
+                 else
+                     unw_access_ar(info, UNW_AR_CSD, &reg, 0);
+                 size = sizeof(reg);
+                 break;
+             case IA64_SSD_REGNUM:
+                 if (ptregs)
+                     reg = ptregs->ar_ssd;
+                 else
+                     unw_access_ar(info, UNW_AR_SSD, &reg, 0);
+                 size = sizeof(reg);
+                 break;
+             case IA64_UNAT_REGNUM:
+                 if (ptregs)
+                     reg = ptregs->ar_unat;
+                 else
+                     unw_access_ar(info, UNW_AR_UNAT, &reg, 0);
+                 size = sizeof(reg);
+                 break;
+             case IA64_RNAT_REGNUM:
+                 unw_access_ar(info, UNW_AR_RNAT, &reg, 0);
+                 size = sizeof(reg);
+                 break;
+             case IA64_BSPSTORE_REGNUM:
+                 unw_access_ar(info, UNW_AR_BSPSTORE, &reg, 0);
+                 size = sizeof(reg);
+                 break;
+             case IA64_PFS_REGNUM:
```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```

+         unw_access_ar(info, UNW_AR_PFS, &reg, 0);
+         size = sizeof(reg);
+         break;
+     case IA64_LC_REGNUM:
+         unw_access_ar(info, UNW_AR_LC, &reg, 0);
+         size = sizeof(reg);
+         break;
+     case IA64_EC_REGNUM:
+         unw_access_ar(info, UNW_AR_EC, &reg, 0);
+         size = sizeof(reg);
+         break;
+     case IA64_FPSR_REGNUM:
+         if (ptregs)
+             reg = ptregs->ar_fpsr;
+         else
+             unw_access_ar(info, UNW_AR_FPSR, &reg, 0);
+         size = sizeof(reg);
+         break;
+     case IA64_RSC_REGNUM:
+         if (ptregs)
+             reg = ptregs->ar_rsc;
+         else
+             unw_access_ar(info, UNW_AR_RNAT, &reg, 0);
+         size = sizeof(reg);
+         break;
+     case IA64_CCV_REGNUM:
+         unw_access_ar(info, UNW_AR_CCV, &reg, 0);
+         size = sizeof(reg);
+         break;
+     }
+
+     if (size) {
+         kgdb_mem2hex((char *)mem, outbuffer, size);
+         outbuffer[size * 2] = 0;
+     } else
+         strcpy(outbuffer, "E0");
+
+     return;
+ }
+
+void kgdb_put_reg(char *inbuffer, char *outbuffer, int regnum,
+                 struct unw_frame_info *info, struct pt_regs *ptregs)
+{
+     unsigned long reg;
+     char nat = 0;
+     struct ia64_fpreg freg;
+     int i;
+     char *ptr;
+
+     ptr = inbuffer;
+     kgdb_hex2long(&ptr, &reg);
+     strcpy(outbuffer, "OK");
+
+     if ((regnum >= IA64_GR0_REGNUM && regnum <= (IA64_GR0_REGNUM + 1)) ||
+         (regnum >= (IA64_GR0_REGNUM + 4) && regnum <= (IA64_GR0_REGNUM + 7))
+         || (regnum >= (IA64_GR0_REGNUM + 16)
+             && regnum <= (IA64_GR0_REGNUM + 31))) {
+         unw_access_gr(info, regnum - IA64_GR0_REGNUM, &reg, &nat, 1);
+     } else
+         if ((regnum >= (IA64_GR0_REGNUM + 2)
+             && regnum <= (IA64_GR0_REGNUM + 3))
+             || (regnum >= (IA64_GR0_REGNUM + 8)

```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```

+         && regnum <= (IA64_GR0_REGNUM + 15))) {
+     for (i = 0;
+         i <
+         (sizeof(gr_reg_to_ptreg_index) /
+          sizeof(gr_reg_to_ptreg_index[0])); i++)
+         if (gr_reg_to_ptreg_index[0].reg == regnum) {
+             *((unsigned long *)
+              ((void *)ptregs) +
+              gr_reg_to_ptreg_index[i].ptregoff) = reg;
+             break;
+         }
+ } else if (regnum >= IA64_BR0_REGNUM && regnum <= (IA64_BR0_REGNUM + 7))
+     switch (regnum) {
+     case IA64_BR0_REGNUM:
+     case IA64_BR0_REGNUM + 6:
+     case IA64_BR0_REGNUM + 7:
+         for (i = 0; i < (sizeof(br_reg_to_ptreg_index) /
+                          sizeof(br_reg_to_ptreg_index[0])); i++)
+             if (br_reg_to_ptreg_index[i].reg == regnum) {
+                 *((unsigned long *)
+                  ((void *)ptregs) +
+                  br_reg_to_ptreg_index[i].ptregoff) =
+                 reg;
+                 break;
+             }
+         break;
+     case IA64_BR0_REGNUM + 1:
+     case IA64_BR0_REGNUM + 2:
+     case IA64_BR0_REGNUM + 3:
+     case IA64_BR0_REGNUM + 4:
+     case IA64_BR0_REGNUM + 5:
+         unw_access_br(info, regnum - IA64_BR0_REGNUM, &reg, 1);
+         break;
+ } else if (regnum >= IA64_FR0_REGNUM
+           && regnum <= (IA64_FR0_REGNUM + 127))
+     switch (regnum) {
+     case IA64_FR0_REGNUM + 6:
+     case IA64_FR0_REGNUM + 7:
+     case IA64_FR0_REGNUM + 8:
+     case IA64_FR0_REGNUM + 9:
+     case IA64_FR0_REGNUM + 10:
+     case IA64_FR0_REGNUM + 11:
+     case IA64_FR0_REGNUM + 12:
+         freg.u.bits[0] = reg;
+         kgdb_hex2long(&ptr, &freg.u.bits[1]);
+         *(&ptregs->f6 + (regnum - (IA64_FR0_REGNUM + 6))) =
+         freg;
+         break;
+     default:
+         break;
+ } else if (regnum == IA64_IP_REGNUM)
+     ptregs->cr_iip = reg;
+ else if (regnum == IA64_CFM_REGNUM)
+     ptregs->cr_ifs = reg;
+ else if (regnum == IA64_PSR_REGNUM)
+     ptregs->cr_ipsr = reg;
+ else if (regnum == IA64_PR_REGNUM)
+     ptregs->pr = reg;
+ else if (regnum >= IA64_AR0_REGNUM && regnum <= IA64_EC_REGNUM)
+     switch (regnum) {
+     case IA64_CSD_REGNUM:
+         ptregs->ar_csd = reg;

```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```
+         break;
+     case IA64_SSD_REGNUM:
+         ptregs->ar_ssd = reg;
+     case IA64_UNAT_REGNUM:
+         ptregs->ar_unat = reg;
+     case IA64_RNAT_REGNUM:
+         unw_access_ar(info, UNW_AR_RNAT, &reg, 1);
+         break;
+     case IA64_BSPSTORE_REGNUM:
+         unw_access_ar(info, UNW_AR_BSPSTORE, &reg, 1);
+         break;
+     case IA64_PFS_REGNUM:
+         unw_access_ar(info, UNW_AR_PFS, &reg, 1);
+         break;
+     case IA64_LC_REGNUM:
+         unw_access_ar(info, UNW_AR_LC, &reg, 1);
+         break;
+     case IA64_EC_REGNUM:
+         unw_access_ar(info, UNW_AR_EC, &reg, 1);
+         break;
+     case IA64_FPSR_REGNUM:
+         ptregs->ar_fpsr = reg;
+         break;
+     case IA64_RSC_REGNUM:
+         ptregs->ar_rsc = reg;
+         break;
+     case IA64_CCV_REGNUM:
+         unw_access_ar(info, UNW_AR_CCV, &reg, 1);
+         break;
+     } else
+         strcpy(outbuffer, "E01");
+
+     return;
+}
+
+void regs_to_gdb_regs(unsigned long *gdb_regs, struct pt_regs *regs)
+{
+}
+
+void sleeping_thread_to_gdb_regs(unsigned long *gdb_regs, struct task_struct *p)
+{
+}
+
+void gdb_regs_to_regs(unsigned long *gdb_regs, struct pt_regs *regs)
+{
+}
+
+#define      MAX_HW_BREAKPOINT      (20)
+long hw_break_total_dbr, hw_break_total_ibr;
+#define      HW_BREAKPOINT      (hw_break_total_dbr + hw_break_total_ibr)
+#define      WATCH_INSTRUCTION      0x0
+#define      WATCH_WRITE      0x1
+#define      WATCH_READ      0x2
+#define      WATCH_ACCESS      0x3
+
+#define      HWCAP_DBR      ((1 << WATCH_WRITE) | (1 << WATCH_READ))
+#define      HWCAP_IBR      (1 << WATCH_INSTRUCTION)
+struct hw_breakpoint {
+    unsigned enabled;
+    unsigned long capable;
+    unsigned long type;
+};
```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```

+     unsigned long mask;
+     unsigned long addr;
+} *breakinfo;
+
+static struct hw_breakpoint hwbreaks[MAX_HW_BREAKPOINT];
+
+enum instruction_type { A, I, M, F, B, L, X, u };
+
+static enum instruction_type bundle_encoding[32][3] = {
+     {M, I, I},          /* 00 */
+     {M, I, I},          /* 01 */
+     {M, I, I},          /* 02 */
+     {M, I, I},          /* 03 */
+     {M, L, X},          /* 04 */
+     {M, L, X},          /* 05 */
+     {u, u, u},          /* 06 */
+     {u, u, u},          /* 07 */
+     {M, M, I},          /* 08 */
+     {M, M, I},          /* 09 */
+     {M, M, I},          /* 0A */
+     {M, M, I},          /* 0B */
+     {M, F, I},          /* 0C */
+     {M, F, I},          /* 0D */
+     {M, M, F},          /* 0E */
+     {M, M, F},          /* 0F */
+     {M, I, B},          /* 10 */
+     {M, I, B},          /* 11 */
+     {M, B, B},          /* 12 */
+     {M, B, B},          /* 13 */
+     {u, u, u},          /* 14 */
+     {u, u, u},          /* 15 */
+     {B, B, B},          /* 16 */
+     {B, B, B},          /* 17 */
+     {M, M, B},          /* 18 */
+     {M, M, B},          /* 19 */
+     {u, u, u},          /* 1A */
+     {u, u, u},          /* 1B */
+     {M, F, B},          /* 1C */
+     {M, F, B},          /* 1D */
+     {u, u, u},          /* 1E */
+     {u, u, u},          /* 1F */
+};
+
+static int kgdb_arch_set_breakpoint(unsigned long addr, char *saved_instr)
+{
+     extern unsigned long _start[];
+     unsigned long slot = addr & 0xf, bundle_addr;
+     unsigned long template;
+     struct bundle {
+         struct {
+             unsigned long long template:5;
+             unsigned long long slot0:41;
+             unsigned long long slot1_p0:64 - 46;
+         } quad0;
+         struct {
+             unsigned long long slot1_p1:41 - (64 - 46);
+             unsigned long long slot2:41;
+         } quad1;
+     } bundle;
+     int ret;
+
+     bundle_addr = addr & ~0xFULL;

```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```
+
+   if (bundle_addr == (unsigned long)_start)
+       return 0;
+
+   ret = kgdb_get_mem((char *)bundle_addr, (char *)&bundle,
+                     BREAK_INSTR_SIZE);
+   if (ret < 0)
+       return ret;
+
+   if (slot > 2)
+       slot = 0;
+
+   memcpy(saved_instr, &bundle, BREAK_INSTR_SIZE);
+   template = bundle.quad0.template;
+
+   if (slot == 1 && bundle_encoding[template][1] == L)
+       slot = 2;
+
+   switch (slot) {
+   case 0:
+       bundle.quad0.slot0 = BREAKNUM;
+       break;
+   case 1:
+       bundle.quad0.slot1_p0 = BREAKNUM;
+       bundle.quad1.slot1_p1 = (BREAKNUM >> (64 - 46));
+       break;
+   case 2:
+       bundle.quad1.slot2 = BREAKNUM;
+       break;
+   }
+
+   return kgdb_set_mem((char *)bundle_addr, (char *)&bundle,
+                      BREAK_INSTR_SIZE);
+}
+
+static int kgdb_arch_remove_breakpoint(unsigned long addr, char *bundle)
+{
+   extern unsigned long _start[];
+
+   addr = addr & ~0xFULL;
+   if (addr == (unsigned long)_start)
+       return 0;
+   return kgdb_set_mem((char *)addr, (char *)bundle, BREAK_INSTR_SIZE);
+}
+
+static int hw_breakpoint_init;
+
+void do_init_hw_break(void)
+{
+   s64 status;
+   int i;
+
+   hw_breakpoint_init = 1;
+
+   #ifdef CONFIG_IA64_HP_SIM
+   hw_break_total_ibr = 8;
+   hw_break_total_dbr = 8;
+   status = 0;
+   #else
+   status = ia64_pal_debug_info(&hw_break_total_ibr, &hw_break_total_dbr);
+   #endif
+}
+
```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```

+     if (status) {
+         printk(KERN_INFO "do_init_hw_break: pal call failed %d\n",
+                 (int)status);
+         return;
+     }
+
+     if (HW_BREAKPOINT > MAX_HW_BREAKPOINT) {
+         printk(KERN_INFO "do_init_hw_break: %d exceeds max %d\n",
+                 (int)HW_BREAKPOINT, (int)MAX_HW_BREAKPOINT);
+
+         while ((HW_BREAKPOINT > MAX_HW_BREAKPOINT)
+                && hw_break_total_ibr != 1)
+             hw_break_total_ibr--;
+         while (HW_BREAKPOINT > MAX_HW_BREAKPOINT)
+             hw_break_total_dbr--;
+     }
+
+     breakinfo = hwbreaks;
+
+     memset(breakinfo, 0, HW_BREAKPOINT * sizeof(struct hw_breakpoint));
+
+     for (i = 0; i < hw_break_total_dbr; i++)
+         breakinfo[i].capable = HWCAP_DBR;
+
+     for (; i < HW_BREAKPOINT; i++)
+         breakinfo[i].capable = HWCAP_IBR;
+
+     return;
+ }
+
+void kgdb_correct_hw_break(void)
+{
+     int breakno;
+
+     if (!breakinfo)
+         return;
+
+     for (breakno = 0; breakno < HW_BREAKPOINT; breakno++) {
+         if (breakinfo[breakno].enabled) {
+             if (breakinfo[breakno].capable & HWCAP_IBR) {
+                 int ibreakno = breakno - hw_break_total_dbr;
+                 ia64_set_ibr(ibreakno << 1,
+                             breakinfo[breakno].addr);
+                 ia64_set_ibr((ibreakno << 1) + 1,
+                             (~breakinfo[breakno].mask &
+                              ((1UL << 56UL) - 1)) |
+                              (1UL << 56UL) | (1UL << 63UL));
+             } else {
+                 ia64_set_dbr(breakno << 1,
+                             breakinfo[breakno].addr);
+                 ia64_set_dbr((breakno << 1) + 1,
+                             (~breakinfo[breakno].mask &
+                              ((1UL << 56UL) - 1)) |
+                              (1UL << 56UL) |
+                              (breakinfo[breakno].type << 62UL));
+             }
+         } else {
+             if (breakinfo[breakno].capable & HWCAP_IBR)
+                 ia64_set_ibr(((breakno -
+                                hw_break_total_dbr) << 1) + 1,
+                               0);
+             else

```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```
+             ia64_set_dbr((breakno << 1) + 1, 0);
+         }
+     }
+     return;
+ }
+
+int hardware_breakpoint(unsigned long addr, int length, int type, int action)
+{
+     int breakno, found, watch;
+     unsigned long mask;
+     extern unsigned long _start[];
+
+     if (!hw_breakpoint_init)
+         do_init_hw_break();
+
+     if (!breakinfo)
+         return 0;
+     else if (addr == (unsigned long)_start)
+         return 1;
+
+     if (type == WATCH_ACCESS)
+         mask = HWCAP_DBR;
+     else
+         mask = 1UL << type;
+
+     for (watch = 0, found = 0, breakno = 0; breakno < HW_BREAKPOINT;
+         breakno++) {
+         if (action) {
+             if (breakinfo[breakno].enabled
+                 || !(breakinfo[breakno].capable & mask))
+                 continue;
+             breakinfo[breakno].enabled = 1;
+             breakinfo[breakno].type = type;
+             breakinfo[breakno].mask = length - 1;
+             breakinfo[breakno].addr = addr;
+             watch = breakno;
+         } else if (breakinfo[breakno].enabled &&
+                 ((length < 0 && breakinfo[breakno].addr == addr) ||
+                 ((breakinfo[breakno].capable & mask) &&
+                 (breakinfo[breakno].mask == (length - 1)) &&
+                 (breakinfo[breakno].addr == addr)))) {
+             breakinfo[breakno].enabled = 0;
+             breakinfo[breakno].type = 0UL;
+         } else
+             continue;
+         found++;
+         if (type != WATCH_ACCESS)
+             break;
+         else if (found == 2)
+             break;
+         else
+             mask = HWCAP_IBR;
+     }
+
+     if (type == WATCH_ACCESS && found == 1) {
+         breakinfo[watch].enabled = 0;
+         found = 0;
+     }
+
+     mb();
+     return found;
+ }
```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```
+}
+
+int kgdb_arch_set_hw_breakpoint(unsigned long addr, int len,
+                               enum kgdb_bptype type)
+{
+    return hardware_breakpoint(addr, len, type - '1', 1);
+}
+
+int kgdb_arch_remove_hw_breakpoint(unsigned long addr, int len,
+                                   enum kgdb_bptype type)
+{
+    return hardware_breakpoint(addr, len, type - '1', 0);
+}
+
+int kgdb_remove_hw_break(unsigned long addr)
+{
+    return hardware_breakpoint(addr, 8, WATCH_INSTRUCTION, 0);
+}
+
+void kgdb_remove_all_hw_break(void)
+{
+    int i;
+
+    for (i = 0; i < HW_BREAKPOINT; i++)
+        memset(&breakinfo[i], 0, sizeof(struct hw_breakpoint));
+}
+
+int kgdb_set_hw_break(unsigned long addr)
+{
+    return hardware_breakpoint(addr, 8, WATCH_INSTRUCTION, 1);
+}
+
+void kgdb_disable_hw_debug(struct pt_regs *regs)
+{
+    unsigned long hw_breakpoint_status;
+
+    hw_breakpoint_status = ia64_getreg(_IA64_REG_PSR);
+    if (hw_breakpoint_status & IA64_PSR_DB)
+        ia64_setreg(_IA64_REG_PSR_L,
+                    hw_breakpoint_status ^ IA64_PSR_DB);
+}
+
+volatile static struct smp_unw {
+    struct unw_frame_info *unw;
+    struct task_struct *task;
+} smp_unw[NR_CPUS];
+
+static int inline kgdb_get_blocked_state(struct task_struct *p,
+                                         struct unw_frame_info *unw)
+{
+    unsigned long ip;
+    int count = 0;
+
+    unw_init_from_blocked_task(unw, p);
+    ip = 0UL;
+    do {
+        if (unw_unwind(unw) < 0)
+            return -1;
+        unw_get_ip(unw, &ip);
+        if (!in_sched_functions(ip))
+            break;
+    }
```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```
+     } while (count++ < 16);
+
+     if (!ip)
+         return -1;
+     else
+         return 0;
+ }
+
+static void inline kgdb_wait(struct pt_regs *regs)
+{
+     unsigned long hw_breakpoint_status = ia64_getreg(_IA64_REG_PSR);
+     if (hw_breakpoint_status & IA64_PSR_DB)
+         ia64_setreg(_IA64_REG_PSR_L,
+                     hw_breakpoint_status ^ IA64_PSR_DB);
+     kgdb_nmihook(smp_processor_id(), regs);
+     if (hw_breakpoint_status & IA64_PSR_DB)
+         ia64_setreg(_IA64_REG_PSR_L, hw_breakpoint_status);
+     return;
+ }
+
+static void inline normalize(struct unw_frame_info *running,
+                             struct pt_regs *regs)
+{
+     unsigned long sp;
+
+     do {
+         unw_get_sp(running, &sp);
+         if ((sp + 0x10) >= (unsigned long)regs)
+             break;
+     } while (unw_unwind(running) >= 0);
+
+     return;
+ }
+
+static void kgdb_init_running(struct unw_frame_info *unw, void *data)
+{
+     struct pt_regs *regs;
+
+     regs = data;
+     normalize(unw, regs);
+     smp_unw[smp_processor_id()].unw = unw;
+     kgdb_wait(regs);
+ }
+
+void kgdb_wait_ipi(struct pt_regs *regs)
+{
+     struct unw_frame_info unw;
+
+     smp_unw[smp_processor_id()].task = current;
+
+     if (user_mode(regs)) {
+         smp_unw[smp_processor_id()].unw = (struct unw_frame_info *)1;
+         kgdb_wait(regs);
+     } else {
+         if (current->state == TASK_RUNNING)
+             unw_init_running(kgdb_init_running, regs);
+         else {
+             if (kgdb_get_blocked_state(current, &unw))
+                 smp_unw[smp_processor_id()].unw =
+                     (struct unw_frame_info *)1;
+             else
+                 smp_unw[smp_processor_id()].unw = &unw;
+         }
+     }
+ }
```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```

+             kgdb_wait(regs);
+         }
+     }
+
+     smp_unw[smp_processor_id()].unw = NULL;
+     return;
+ }
+
+ void kgdb_roundup_cpus(unsigned long flags)
+ {
+     if (num_online_cpus() > 1)
+         smp_send_nmi_allbutself();
+ }
+
+ static void do_kgdb_handle_exception(struct unw_frame_info *, void *data);
+
+ struct kgdb_state {
+     int e_vector;
+     int signo;
+     unsigned long err_code;
+     struct pt_regs *regs;
+     struct unw_frame_info *unw;
+     char *inbuf;
+     char *outbuf;
+     int unwind;
+     int ret;
+ };
+
+ static void inline kgdb_pc(struct pt_regs *regs, unsigned long pc)
+ {
+     regs->cr_iip = pc & ~0xf;
+     ia64_psr(regs)->ri = pc & 0x3;
+     return;
+ }
+
+ volatile int kgdb_hwbreak_sstep[NR_CPUS];
+
+ int kgdb_arch_handle_exception(int e_vector, int signo,
+                               int err_code, char *remcom_in_buffer,
+                               char *remcom_out_buffer,
+                               struct pt_regs *linux_regs)
+ {
+     struct kgdb_state info;
+
+     info.e_vector = e_vector;
+     info.signo = signo;
+     info.err_code = err_code;
+     info.unw = (void *)0;
+     info.inbuf = remcom_in_buffer;
+     info.outbuf = remcom_out_buffer;
+     info.unwind = 0;
+     info.ret = -1;
+
+     if (remcom_in_buffer[0] == 'c' || remcom_in_buffer[0] == 's') {
+         info.regs = linux_regs;
+         do_kgdb_handle_exception(NULL, &info);
+     } else if (kgdb_usethread == current) {
+         info.regs = linux_regs;
+         /*
+          * If we were invoked by init handler, then
+          * the unwind must be handled as an interruption.
+          */
+     }
+ }

```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```

+         if (e_vector == -1) {
+             struct unw_frame_info unw_info;
+
+             unw_init_from_interruption(&unw_info, current,
+                                       linux_regs,
+                                       (struct switch_stack *)
+                                       (((char *)linux_regs) -
+                                       sizeof(struct
+                                           switch_stack)));
+             do_kgdb_handle_exception(&unw_info, &info);
+         } else {
+             info.unwind = 1;
+             unw_init_running(do_kgdb_handle_exception, &info);
+         }
+     } else if (kgdb_uthread->state != TASK_RUNNING) {
+         struct unw_frame_info unw_info;
+
+         if (kgdb_get_blocked_state(kgdb_uthread, &unw_info)) {
+             info.ret = 1;
+             goto bad;
+         }
+         info.regs = NULL;
+         do_kgdb_handle_exception(&unw_info, &info);
+     } else {
+         int i;
+
+         for (i = 0; i < NR_CPUS; i++)
+             if (smp_unw[i].task == kgdb_uthread && smp_unw[i].unw
+                 && smp_unw[i].unw != (struct unw_frame_info *)1) {
+                 info.regs = NULL;
+                 do_kgdb_handle_exception(smp_unw[i].unw, &info);
+                 break;
+             } else {
+                 info.ret = 1;
+                 goto bad;
+             }
+     }
+
+bad:
+     if (info.ret != -1 && remcom_in_buffer[0] == 'p') {
+         unsigned long bad = 0xbad4badbadbadbadUL;
+
+         printk("kgdb_arch_handle_exception: p packet bad (%s)\n",
+               remcom_in_buffer);
+         kgdb_mem2hex((char *)&bad, remcom_out_buffer, sizeof(bad));
+         remcom_out_buffer[sizeof(bad) * 2] = 0;
+         info.ret = -1;
+     }
+     return info.ret;
+}
+
+static void do_kgdb_handle_exception(struct unw_frame_info *unw_info,
+                                    void *data)
+{
+     long addr;
+     char *ptr;
+     unsigned long newPC;
+     int e_vector, signo;
+     unsigned long err_code;
+     struct pt_regs *linux_regs;
+     struct kgdb_state *info;
+     char *remcom_in_buffer, *remcom_out_buffer;

```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```

+
+   info = data;
+   info->unw = unw_info;
+   e_vector = info->e_vector;
+   signo = info->signo;
+   err_code = info->err_code;
+   remcom_in_buffer = info->inbuf;
+   remcom_out_buffer = info->outbuf;
+   linux_regs = info->regs;
+
+   if (info->unwind)
+       normalize(unw_info, linux_regs);
+
+   switch (remcom_in_buffer[0]) {
+   case 'p':
+       {
+           int regnum;
+
+           kgdb_hex2mem(&remcom_in_buffer[1], (char *)&regnum,
+                       sizeof(regnum));
+           if (regnum >= NUM_REGS) {
+               remcom_out_buffer[0] = 'E';
+               remcom_out_buffer[1] = 0;
+           } else
+               kgdb_get_reg(remcom_out_buffer, regnum,
+                           unw_info, linux_regs);
+
+           break;
+       }
+   case 'P':
+       {
+           int regno;
+           long v;
+           char *ptr;
+
+           ptr = &remcom_in_buffer[1];
+           if ((!kgdb_usethread || kgdb_usethread == current) &&
+               kgdb_hex2long(&ptr, &v) &&
+               *ptr++ == '=' && (v >= 0)) {
+               regno = (int)v;
+               regno = (regno >= NUM_REGS ? 0 : regno);
+               kgdb_put_reg(ptr, remcom_out_buffer, regno,
+                           unw_info, linux_regs);
+           } else
+               strcpy(remcom_out_buffer, "E01");
+
+           break;
+       }
+   case 'c':
+   case 's':
+       if (e_vector == 11 && err_code == KGDBBREAKNUM) {
+           if (ia64_psr(linux_regs)->ri < 2)
+               kgdb_pc(linux_regs, linux_regs->cr_iip +
+                       ia64_psr(linux_regs)->ri + 1);
+           else
+               kgdb_pc(linux_regs, linux_regs->cr_iip + 16);
+       }
+
+       /* try to read optional parameter, pc unchanged if no parm */
+       ptr = &remcom_in_buffer[1];
+       if (kgdb_hex2long(&ptr, &addr)) {
+           linux_regs->cr_iip = addr;
+       }
+       newPC = linux_regs->cr_iip;

```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```
+
+      /* clear the trace bit */
+      linux_regs->cr_ipsr &= ~IA64_PSR_SS;
+
+      atomic_set(&cpu_doing_single_step, -1);
+
+      /* set the trace bit if we're stepping or took a hardware break */
+      if (remcom_in_buffer[0] == 's' || e_vector == 29) {
+          linux_regs->cr_ipsr |= IA64_PSR_SS;
+          debugger_step = 1;
+          if (kgdb_contthread)
+              atomic_set(&cpu_doing_single_step,
+                          smp_processor_id());
+      }
+
+      kgdb_correct_hw_break();
+
+      /* if not hardware breakpoint, then reenale them */
+      if (e_vector != 29)
+          linux_regs->cr_ipsr |= IA64_PSR_DB;
+      else {
+          kgdb_hwbreak_sstep[smp_processor_id()] = 1;
+          linux_regs->cr_ipsr &= ~IA64_PSR_DB;
+      }
+
+      info->ret = 0;
+      break;
+  default:
+      break;
+  }
+
+  return;
+}
+
+struct kgdb_arch arch_kgdb_ops = {
+    .set_breakpoint = kgdb_arch_set_breakpoint,
+    .remove_breakpoint = kgdb_arch_remove_breakpoint,
+    .set_hw_breakpoint = kgdb_arch_set_hw_breakpoint,
+    .remove_hw_breakpoint = kgdb_arch_remove_hw_breakpoint,
+    .gdb_bpt_instr = {0xcc},
+    .flags = KGDB_HW_BREAKPOINT,
+};
diff -puN /dev/null arch/ia64/kernel/kgdb-jmp.S
--- /dev/null      2005-08-08 08:07:04.272443000 -0700
+++ linux-2.6.13-trini/arch/ia64/kernel/kgdb-jmp.S      2005-08-08 12:18:23.000000000 -0700
@@ -0,0 +1,238 @@
+/* setjmp() and longjmp() assembler support for kdb on ia64.
+
+   This code was copied from glibc CVS as of 2001-06-27 and modified where
+   necessary to fit the kernel.
+   Keith Owens <kaos@melbourne.sgi.com> 2001-06-27
+ */
+
+/* Copyright (C) 1999, 2000, 2001 Free Software Foundation, Inc.
+   Contributed by David Mosberger-Tang <davidm@hpl.hp.com>.
+
+   The GNU C Library is free software; you can redistribute it and/or
+   modify it under the terms of the GNU Library General Public License as
+   published by the Free Software Foundation; either version 2 of the
+   License, or (at your option) any later version.
+
+   The GNU C Library is distributed in the hope that it will be useful,
```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```
+ but WITHOUT ANY WARRANTY; without even the implied warranty of
+ MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
+ Library General Public License for more details.
+
+ You should have received a copy of the GNU Library General Public
+ License along with the GNU C Library; see the file COPYING.LIB. If
+ not, write to the Free Software Foundation, Inc.,
+ 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
+*/
+
+#include <asm/asmmacro.h>
+GLOBAL_ENTRY(kgdb_fault_setjmp)
+    .prologue ASM_UNW_PRLG_RP|ASM_UNW_PRLG_PFS, ASM_UNW_PRLG_GRSAVE(2)
+    alloc loc1=ar.pfs,2,2,2,0
+    mov r16=ar.unat
+    ;;
+    mov r17=ar.fpsr
+    mov r2=in0
+    add r3=8,in0
+    ;;
+.mem.offset 0,0;
+    st8.spill.nta [r2]=sp,16          // r12 (sp)
+.mem.offset 8,0;
+    st8.spill.nta [r3]=gp,16          // r1 (gp)
+    ;;
+    st8.nta [r2]=r16,16              // save caller's unat
+    st8.nta [r3]=r17,16              // save fpsr
+    add r8=0xa0,in0
+    ;;
+.mem.offset 160,0;
+    st8.spill.nta [r2]=r4,16          // r4
+.mem.offset 168,0;
+    st8.spill.nta [r3]=r5,16          // r5
+    add r9=0xb0,in0
+    ;;
+    stf.spill.nta [r8]=f2,32
+    stf.spill.nta [r9]=f3,32
+    mov loc0=rp
+    .body
+    ;;
+    stf.spill.nta [r8]=f4,32
+    stf.spill.nta [r9]=f5,32
+    mov r17=b1
+    ;;
+    stf.spill.nta [r8]=f16,32
+    stf.spill.nta [r9]=f17,32
+    mov r18=b2
+    ;;
+    stf.spill.nta [r8]=f18,32
+    stf.spill.nta [r9]=f19,32
+    mov r19=b3
+    ;;
+    stf.spill.nta [r8]=f20,32
+    stf.spill.nta [r9]=f21,32
+    mov r20=b4
+    ;;
+    stf.spill.nta [r8]=f22,32
+    stf.spill.nta [r9]=f23,32
+    mov r21=b5
+    ;;
+    stf.spill.nta [r8]=f24,32
+    stf.spill.nta [r9]=f25,32
```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```

+     mov r22=ar.lc
+     ;;
+     stf.spill.nta [r8]=f26,32
+     stf.spill.nta [r9]=f27,32
+     mov r24=pr
+     ;;
+     stf.spill.nta [r8]=f28,32
+     stf.spill.nta [r9]=f29,32
+     ;;
+     stf.spill.nta [r8]=f30
+     stf.spill.nta [r9]=f31
+
+.mem.offset 0,0;
+     st8.spill.nta [r2]=r6,16           // r6
+.mem.offset 8,0;
+     st8.spill.nta [r3]=r7,16           // r7
+     ;;
+     mov r23=ar.bsp
+     mov r25=ar.unat
+     st8.nta [r2]=loc0,16              // b0
+     st8.nta [r3]=r17,16              // b1
+     ;;
+     st8.nta [r2]=r18,16              // b2
+     st8.nta [r3]=r19,16              // b3
+     ;;
+     st8.nta [r2]=r20,16              // b4
+     st8.nta [r3]=r21,16              // b5
+     ;;
+     st8.nta [r2]=loc1,16             // ar.pfs
+     st8.nta [r3]=r22,16             // ar.lc
+     ;;
+     st8.nta [r2]=r24,16              // pr
+     st8.nta [r3]=r23,16             // ar.bsp
+     ;;
+     st8.nta [r2]=r25                 // ar.unat
+     st8.nta [r3]=in0                 // &__jmp_buf
+     mov r8=0
+     mov rp=loc0
+     mov ar.pfs=loc1
+     br.ret.sptk.few rp
+END(kdba_setjmp)
+#define      pPos      p6      /* is rotate count positive? */
+#define      pNeg      p7      /* is rotate count negative? */
+GLOBAL _ENTRY(kgdb_fault_longjmp)
+     alloc r8=ar.pfs,2,1,0,0
+     mov r27=ar.rsc
+     add r2=0x98,in0                  // r2 <- &jmpbuf.orig_jmp_buf_addr
+     ;;
+     ld8 r8=[r2],-16                  // r8 <- orig_jmp_buf_addr
+     mov r10=ar.bsp
+     and r11=~0x3,r27                 // clear ar.rsc.mode
+     ;;
+     flushrs                          // flush dirty regs to backing store (must be first in insn grp)
+     ld8 r23=[r2],8                   // r23 <- jmpbuf.ar_bsp
+     sub r8=r8,in0                    // r8 <- &orig_jmpbuf - &jmpbuf
+     ;;
+     ld8 r25=[r2]                     // r25 <- jmpbuf.ar_unat
+     extr.u r8=r8,3,6                 // r8 <- (&orig_jmpbuf - &jmpbuf)/8 & 0x3f
+     ;;
+     cmp.lt pNeg,pPos=r8,r0
+     mov r2=in0
+     ;;

```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```

+(pPos) mov r16=r8
+(pNeg) add r16=64,r8
+(pPos) sub r17=64,r8
+(pNeg) sub r17=r0,r8
+
+    ;;
+    mov ar.rsc=r11          // put RSE in enforced lazy mode
+    shr.u r8=r25,r16
+    add r3=8,in0           // r3 <- &jmpbuf.r1
+    shl r9=r25,r17
+    ;;
+    or r25=r8,r9
+    ;;
+    mov r26=ar.rnat
+    mov ar.unat=r25        // setup ar.unat (NaT bits for r1, r4-r7, and r12)
+    ;;
+    ld8.fill.nta sp=[r2],16 // r12 (sp)
+    ld8.fill.nta gp=[r3],16 // r1 (gp)
+    dep r11=-1,r23,3,6     // r11 <- ia64_rse_rnat_addr(jmpbuf.ar_bsp)
+    ;;
+    ld8.nta r16=[r2],16    // caller's unat
+    ld8.nta r17=[r3],16    // fpsr
+    ;;
+    ld8.fill.nta r4=[r2],16 // r4
+    ld8.fill.nta r5=[r3],16 // r5 (gp)
+    cmp.geu p8,p0=r10,r11 // p8 <- (ar.bsp >= jmpbuf.ar_bsp)
+    ;;
+    ld8.fill.nta r6=[r2],16 // r6
+    ld8.fill.nta r7=[r3],16 // r7
+    ;;
+    mov ar.unat=r16        // restore caller's unat
+    mov ar.fpsr=r17        // restore fpsr
+    ;;
+    ld8.nta r16=[r2],16    // b0
+    ld8.nta r17=[r3],16    // b1
+    ;;
+(p8) ld8 r26=[r11]        // r26 <- *ia64_rse_rnat_addr(jmpbuf.ar_bsp)
+    mov ar.bspstore=r23   // restore ar.bspstore
+    ;;
+    ld8.nta r18=[r2],16    // b2
+    ld8.nta r19=[r3],16    // b3
+    ;;
+    ld8.nta r20=[r2],16    // b4
+    ld8.nta r21=[r3],16    // b5
+    ;;
+    ld8.nta r11=[r2],16    // ar.pfs
+    ld8.nta r22=[r3],56    // ar.lc
+    ;;
+    ld8.nta r24=[r2],32    // pr
+    mov b0=r16
+    ;;
+    ldf.fill.nta f2=[r2],32
+    ldf.fill.nta f3=[r3],32
+    mov b1=r17
+    ;;
+    ldf.fill.nta f4=[r2],32
+    ldf.fill.nta f5=[r3],32
+    mov b2=r18
+    ;;
+    ldf.fill.nta f16=[r2],32
+    ldf.fill.nta f17=[r3],32
+    mov b3=r19
+    ;;

```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```

+     ldf.fill.nta f18=[r2],32
+     ldf.fill.nta f19=[r3],32
+     mov b4=r20
+     ;;
+     ldf.fill.nta f20=[r2],32
+     ldf.fill.nta f21=[r3],32
+     mov b5=r21
+     ;;
+     ldf.fill.nta f22=[r2],32
+     ldf.fill.nta f23=[r3],32
+     mov ar.lc=r22
+     ;;
+     ldf.fill.nta f24=[r2],32
+     ldf.fill.nta f25=[r3],32
+     cmp.eq p8,p9=0,in1
+     ;;
+     ldf.fill.nta f26=[r2],32
+     ldf.fill.nta f27=[r3],32
+     mov ar.pfs=r11
+     ;;
+     ldf.fill.nta f28=[r2],32
+     ldf.fill.nta f29=[r3],32
+     ;;
+     ldf.fill.nta f30=[r2]
+     ldf.fill.nta f31=[r3]
+(p8)  mov r8=1
+
+     mov ar.rnat=r26           // restore ar.rnat
+     ;;
+     mov ar.rsc=r27           // restore ar.rsc
+(p9)  mov r8=in1
+
+     invala                   // virt. -> phys. regnum mapping may change
+     mov pr=r24,-1
+     br.ret.sptk.few rp
+END(kgdb_fault_longjmp)
diff -puN arch/ia64/kernel/Makefile~ia64-lite arch/ia64/kernel/Makefile
--- linux-2.6.13/arch/ia64/kernel/Makefile~ia64-lite      2005-08-08 12:18:23.000000000 -0700
+++ linux-2.6.13-trini/arch/ia64/kernel/Makefile          2005-08-08 12:18:23.000000000 -0700
@@ -24,6 +24,7 @@ obj-$(CONFIG_IA64_MCA_RECOVERY)      += mca_r
obj-$(CONFIG_KPROBES)      += kprobes.o jprobes.o
obj-$(CONFIG_IA64_UNCACHED_ALLOCATOR) += uncached.o
mca_recovery-y            += mca_drv.o mca_drv_asm.o
+obj-$(CONFIG_KGDB)       += kgdb.o kgdb-jmp.o

# The gate DSO image is built using a special linker script.
targets += gate.so gate-syms.o
diff -puN arch/ia64/kernel/mca.c~ia64-lite arch/ia64/kernel/mca.c
--- linux-2.6.13/arch/ia64/kernel/mca.c~ia64-lite        2005-08-08 12:18:23.000000000 -0700
+++ linux-2.6.13-trini/arch/ia64/kernel/mca.c           2005-08-08 12:18:23.000000000 -0700
@@ -472,9 +472,18 @@ static void
init_handler_platform (pal_min_state_area_t *ms,
                      struct pt_regs *pt, struct switch_stack *sw)
{
+ifndef CONFIG_KGDB
+     struct unw_frame_info info;
+endif

+     /* if a kernel debugger is available call it here else just dump the registers */
+ifdef CONFIG_KGDB
+     fetch_min_state(ms, pt, sw);
+endif
+     /*

```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```

+      * switch_stack is at ((char *) pt) - sizeof (struct switch_stack)
+      */
+      kgdb_handle_exception(-1, SIGTRAP, 0, pt);
+##else

+      /*
+       * Wait for a bit.  On some machines (e.g., HP's zx2000 and zx6000, INIT can be
@@ -510,6 +519,7 @@ init_handler_platform (pal_min_state_are
+      if (!tasklist_lock.write_lock)
+          read_unlock(&tasklist_lock);
+      #endif
+##endif

+      printk("\nINIT dump complete.  Please reboot now.\n");
+      while (1);          /* hang city if no debugger */
diff -puN arch/ia64/kernel/process.c~ia64-lite arch/ia64/kernel/process.c
--- linux-2.6.13/arch/ia64/kernel/process.c~ia64-lite    2005-08-08 12:18:23.000000000 -0700
+++ linux-2.6.13-trini/arch/ia64/kernel/process.c        2005-08-08 12:18:23.000000000 -0700
@@ -451,6 +451,9 @@ copy_thread (int nr, unsigned long clone
+      */
+      child_ptregs->cr_ipsr = ((child_ptregs->cr_ipsr | IA64_PSR_BITS_TO_SET)
+          & ~(IA64_PSR_BITS_TO_CLEAR | IA64_PSR_PP | IA64_PSR_UP));
+##ifdef CONFIG_KGDB
+      child_ptregs->cr_ipsr |= IA64_PSR_DB;
+##endif

+      /*
+       * NOTE: The calling convention considers all floating point
@@ -679,6 +682,9 @@ kernel_thread (int (*fn)(void *), void *
+      regs.pt.r11 = (unsigned long) arg;          /* 2nd argument */
+      /* Preserve PSR bits, except for bits 32-34 and 37-45, which we can't read.  */
+      regs.pt.cr_ipsr = ia64_getreg(_IA64_REG_PSR) | IA64_PSR_BN;
+##ifdef CONFIG_KGDB
+      regs.pt.cr_ipsr |= IA64_PSR_DB;
+##endif
+      regs.pt.cr_ifs = 1UL << 63;          /* mark as valid, empty frame */
+      regs.sw.ar_fpsr = regs.pt.ar_fpsr = ia64_getreg(_IA64_REG_AR_FPSR);
+      regs.sw.ar_bspstore = (unsigned long) current + IA64_RBS_OFFSET;
diff -puN arch/ia64/kernel/smp.c~ia64-lite arch/ia64/kernel/smp.c
--- linux-2.6.13/arch/ia64/kernel/smp.c~ia64-lite      2005-08-08 12:18:23.000000000 -0700
+++ linux-2.6.13-trini/arch/ia64/kernel/smp.c          2005-08-08 12:18:23.000000000 -0700
@@ -47,6 +47,7 @@
+      #include <asm/tlbflush.h>
+      #include <asm/unistd.h>
+      #include <asm/mca.h>
+##include <linux/kgdb.h>

+      /*
+       * Structure and data for smp_call_function().  This is designed to minimise static memory
@@ -66,6 +67,9 @@ static volatile struct call_data_struct
+
+      #define IPI_CALL_FUNC          0
+      #define IPI_CPU_STOP          1
+##ifdef CONFIG_KGDB
+      #define IPI_KGDB_INTERRUPT    2
+##endif

+      /* This needs to be cacheline aligned because it is written to by *other* CPUs.  */
+      static DEFINE_PER_CPU(u64, ipi_operation) ____cacheline_aligned;
@@ -155,6 +159,11 @@ handle_IPI (int irq, void *dev_id, struc
+          case IPI_CPU_STOP:
+              stop_this_cpu();

```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```
                                break;
+#ifdef CONFIG_KGDB
+                                case IPI_KGDB_INTERRUPT:
+                                    kgdb_wait_ipi(regs);
+                                    break;
+#endif

                                default:
                                    printk(KERN_CRIT "Unknown IPI on CPU %d: %lu\n", this_cpu, which)
@@ -305,6 +314,14 @@ smp_call_function_single (int cpuid, voi
    }
    EXPORT_SYMBOL(smp_call_function_single);

+#ifdef CONFIG_KGDB
+void
+smp_send_nmi_allbutself(void)
+{
+    send_IPI_allbutself(IPI_KGDB_INTERRUPT);
+}
+#endif
+
+/*
+ * this function sends a 'generic call function' IPI to all other CPUs
+ * in the system.
diff -puN arch/ia64/kernel/traps.c~ia64-lite arch/ia64/kernel/traps.c
--- linux-2.6.13/arch/ia64/kernel/traps.c~ia64-lite      2005-08-08 12:18:23.000000000 -0700
+++ linux-2.6.13-trini/arch/ia64/kernel/traps.c 2005-08-08 12:18:23.000000000 -0700
@@ -15,6 +15,7 @@
 #include <linux/vt_kern.h>                /* For unblank_screen() */
 #include <linux/module.h>                /* for EXPORT_SYMBOL */
 #include <linux/hardirq.h>
+#include <linux/kgdb.h>

 #include <asm/fpswa.h>
 #include <asm/ia32.h>
@@ -108,6 +109,10 @@ die (const char *str, struct pt_regs *re
    } else
        printk(KERN_ERR "Recursive die() failure, output suppressed\n");

+#ifdef CONFIG_KGDB
+    kgdb_handle_exception(1, SIGTRAP, err, regs);
+#endif
+
    bust_spinlocks(0);
    die.lock_owner = -1;
    spin_unlock_irq(&die.lock);
@@ -157,7 +162,9 @@ ia64_bad_break (unsigned long break_num,
                == NOTIFY_STOP) {
                    return;
                }
+#ifndef CONFIG_KGDB
    die_if_kernel("bugcheck!", regs, break_num);
+#endif

    sig = SIGILL; code = ILL_ILLOPC;
    break;

@@ -219,8 +226,10 @@ ia64_bad_break (unsigned long break_num,
    break;

    default:
+#ifndef CONFIG_KGDB
        if (break_num < 0x40000 || break_num > 0x100000)
```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```

        die_if_kernel("Bad break", regs, break_num);
+##endif

        if (break_num < 0x80000) {
            sig = SIGILL; code = __ILL_BREAK;
@@ -228,6 +237,15 @@ ia64_bad_break (unsigned long break_num,
            sig = SIGTRAP; code = TRAP_BRKPT;
        }
    }
+##ifdef CONFIG_KGDB
+    /*
+     * We don't want to trap simulator system calls.
+     */
+    if (break_num != 0x80001) {
+        kgdb_handle_exception(11, sig, break_num, regs);
+        return;
+    }
+##endif
    siginfo.si_signo = sig;
    siginfo.si_errno = 0;
    siginfo.si_code = code;
@@ -543,10 +561,21 @@ ia64_fault (unsigned long vector, unsign
    }
    sprintf(buf, "Unsupported data reference");
    break;
-
-    case 29: /* Debug */
-    case 35: /* Taken Branch Trap */
-    case 36: /* Single Step Trap */
+##ifdef CONFIG_KGDB
+    if (vector == 36 && !user_mode(&regs) &&
+        kgdb_hwbreak_sstep[smp_processor_id()]) {
+        kgdb_hwbreak_sstep[smp_processor_id()] = 0;
+        regs.cr_ipsr &= ~IA64_PSR_SS;
+        return;
+    } else if (!user_mode(&regs)) {
+        kgdb_handle_exception(vector, SIGTRAP, isr, &regs);
+        return;
+    }
+##endif
+    /* Fall */
+    case 35: /* Taken Branch Trap */
+        if (fsys_mode(current, &regs)) {
+            extern char __kernel_syscall_via_break[];
+            /*
@@ -664,6 +693,9 @@ ia64_fault (unsigned long vector, unsign
            sprintf(buf, "Fault %lu", vector);
            break;
        }
+##ifdef CONFIG_KGDB
+    kgdb_handle_exception(vector, SIGTRAP, isr, &regs);
+##endif
    die_if_kernel(buf, &regs, error);
    force_sig(SIGILL, current);
}
diff -puN arch/ia64/kernel/unwind.c~ia64-lite arch/ia64/kernel/unwind.c
--- linux-2.6.13/arch/ia64/kernel/unwind.c~ia64-lite      2005-08-08 12:18:23.000000000 -0700
+++ linux-2.6.13-trini/arch/ia64/kernel/unwind.c        2005-08-08 12:18:23.000000000 -0700
@@ -72,10 +72,68 @@
 # define STAT(x...)
 #endif

```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```

#ifdef CONFIG_KGDB
#define KGDB_EARLY_SIZE 100
+static struct unw_reg_state __initdata kgdb_reg_state[KGDB_EARLY_SIZE];
+static struct unw_labeled_state __initdata kgdb_labeled_state[KGDB_EARLY_SIZE];
+void __initdata *kgdb_reg_state_free, __initdata *kgdb_labeled_state_free;
+
+static void __init
+kgdb_malloc_init(void)
+{
+    int i;
+
+    kgdb_reg_state_free = kgdb_reg_state;
+    for (i = 1; i < KGDB_EARLY_SIZE; i++) {
+        *((unsigned long *) &kgdb_reg_state[i]) = (unsigned long) kgdb_reg_state_free;
+        kgdb_reg_state_free = &kgdb_reg_state[i];
+    }
+
+    kgdb_labeled_state_free = kgdb_labeled_state;
+    for (i = 1; i < KGDB_EARLY_SIZE; i++) {
+        *((unsigned long *) &kgdb_labeled_state[i]) =
+            (unsigned long) kgdb_labeled_state_free;
+        kgdb_labeled_state_free = &kgdb_labeled_state[i];
+    }
+}
+
+static void * __init
+kgdb_malloc(void **mem)
+{
+    void *p;
+
+    p = *mem;
+    *mem = *((void **) p);
+    return p;
+}
+
+static void __init
+kgdb_free(void **mem, void *p)
+{
+    *((void **)p) = *mem;
+    *mem = p;
+}
+
+#define alloc_reg_state() (!malloc_sizes[0].cs_cachep ? \
+    kgdb_malloc(&kgdb_reg_state_free) : \
+    kmalloc(sizeof(struct unw_reg_state), GFP_ATOMIC))
+#define free_reg_state(usr) (!malloc_sizes[0].cs_cachep ? \
+    kgdb_free(&kgdb_reg_state_free, usr) : \
+    kfree(usr))
+#define alloc_labeled_state() (!malloc_sizes[0].cs_cachep ? \
+    kgdb_malloc(&kgdb_labeled_state_free) : \
+    kmalloc(sizeof(struct unw_labeled_state), GFP_ATOMIC))
+#define free_labeled_state(usr) (!malloc_sizes[0].cs_cachep ? \
+    kgdb_free(&kgdb_labeled_state_free, usr) : \
+    kfree(usr))
+
+#else
#define alloc_reg_state() kmalloc(sizeof(struct unw_reg_state), GFP_ATOMIC)
#define free_reg_state(usr) kfree(usr)
#define alloc_labeled_state() kmalloc(sizeof(struct unw_labeled_state), GFP_ATOMIC)
#define free_labeled_state(usr) kfree(usr)
+#endif

```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```

typedef unsigned long unw_word;
typedef unsigned char unw_hash_index_t;
@@ -238,6 +296,24 @@ static struct {
    #endif
};

+#ifdef CONFIG_KGDB
+/*
+ * This makes it safe to call breakpoint() very early
+ * in setup_arch providing:
+ *     1) breakpoint isn't called between lines in cpu_init
+ *        where init_mm.mm_count is incremented and ia64_mmu_init
+ *        is called. Otherwise the test below is invalid.
+ *     2) the memory examined doesn't result in tlbmiss.
+ */
+static unsigned long inline kgdb_unimpl_va_mask(void)
+{
+    if (atomic_read(&init_mm.mm_count) > 1)
+        return local_cpu_data->unimpl_va_mask;
+    else
+        return 0UL;
+}
+#endif
+
+static inline int
+read_only (void *addr)
+{
@@ -1786,7 +1862,11 @@ run_script (struct unw_script *script, s

        case UNW_INSN_LOAD:

            #ifdef UNW_DEBUG
+#ifdef CONFIG_KGDB
+                if ((s[val] & (kgdb_unimpl_va_mask() | 0x7)) != 0
+#else
+                if ((s[val] & (local_cpu_data->unimpl_va_mask | 0x7)) != 0
+#endif
+                || s[val] < TASK_SIZE)
+                {
+                    UNW_DPRINT(0, "unwind.%s: rejecting bad psp=0x%lx\n",
@@ -1821,7 +1901,11 @@ find_save_locs (struct unw_frame_info *i
    struct unw_script *scr;
    unsigned long flags = 0;

+#ifdef CONFIG_KGDB
+    if ((info->ip & (kgdb_unimpl_va_mask() | 0xf)) || info->ip < TASK_SIZE) {
+#else
+    if ((info->ip & (local_cpu_data->unimpl_va_mask | 0xf)) || info->ip < TASK_SIZE) {
+#endif
+        /* don't let obviously bad addresses pollute the cache */
+        /* FIXME: should really be level 0 but it occurs too often. KAO */
+        UNW_DPRINT(1, "unwind.%s: rejecting bad ip=0x%lx\n", __FUNCTION__, info->ip);
@@ -2271,6 +2355,9 @@ unw_init (void)

    init_unwind_table(&unw.kernel_table, "kernel", KERNEL_START, (unsigned long) __gp,
                     __start_unwind, __end_unwind);
+#ifdef CONFIG_KGDB
+    kgdb_malloc_init();
+#endif
+}

+/*

```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```
diff -puN arch/ia64/mm/extable.c~ia64-lite arch/ia64/mm/extable.c
--- linux-2.6.13/arch/ia64/mm/extable.c~ia64-lite      2005-08-08 12:18:23.000000000 -0700
+++ linux-2.6.13-trini/arch/ia64/mm/extable.c      2005-08-08 12:18:23.000000000 -0700
@@ -7,6 +7,7 @@

#include <linux/config.h>
#include <linux/sort.h>
+#include <linux/kgdb.h>

#include <asm/uaccess.h>
#include <asm/module.h>
@@ -74,6 +75,11 @@ search_extable (const struct exception_t
        else
                last = mid - 1;
    }
+#ifdef CONFIG_KGDB
+    if (atomic_read(&debugger_active) && kgdb_may_fault)
+        kgdb_fault_longjmp(kgdb_fault_jmp_regs);
+        /* Not reached. */
+#endif
    return NULL;
}

diff -puN arch/ia64/mm/fault.c~ia64-lite arch/ia64/mm/fault.c
--- linux-2.6.13/arch/ia64/mm/fault.c~ia64-lite      2005-08-08 12:18:23.000000000 -0700
+++ linux-2.6.13-trini/arch/ia64/mm/fault.c      2005-08-08 12:18:23.000000000 -0700
@@ -9,6 +9,7 @@
#include <linux/mm.h>
#include <linux/smp_lock.h>
#include <linux/interrupt.h>
+#include <linux/kgdb.h>

#include <asm/pgtable.h>
#include <asm/processor.h>
@@ -248,6 +249,10 @@ ia64_do_page_fault (unsigned long address
    /*
        bust_spinlocks(1);

+#ifdef CONFIG_KGDB
+    kgdb_handle_exception(14, SIGSEGV, isr, regs);
+#endif
+
    if (address < PAGE_SIZE)
        printk(KERN_ALERT "Unable to handle kernel NULL pointer dereference (address %016

diff -puN /dev/null include/asm-ia64/kgdb.h
--- /dev/null      2005-08-08 08:07:04.272443000 -0700
+++ linux-2.6.13-trini/include/asm-ia64/kgdb.h      2005-08-08 12:18:23.000000000 -0700
@@ -0,0 +1,37 @@
+#ifdef __KERNEL__
+#ifndef _ASM_KGDB_H_
+#define _ASM_KGDB_H_
+
+
+/*
+ * Copyright (C) 2001-2004 Amit S. Kale
+ */
+
+#include <linux/threads.h>
+
+
+/* *****
+/* BUFMAX defines the maximum number of characters in inbound/outbound buffers*/
+/* at least NUMREGBYTES*2 are needed for register packets */
```

Linux-Kernel: [patch 06/16] Add support for IA64 platforms to KGDB

```

+/* Longer buffer is needed to list all threads */
+#define BUFMAX                1024
+
+/* Number of bytes of registers.  We set this to 0 so that certain GDB
+ * packets will fail, forcing the use of others, which are more friendly
+ * on ia64. */
+#define NUMREGBYTES           0
+
+#define NUMCRITREGBYTES       (70*8)
+#define JMP_REGS_ALIGNMENT    __attribute__((aligned(16)))
+
+#define BREAKNUM               0x00003333300LL
+#define KGDBBREAKNUM          0x6665UL
+#define BREAKPOINT()          asm volatile ("break.m 0x6665")
+#define BREAK_INSTR_SIZE      16
+#define CHECK_EXCEPTION_STACK() 1
+#define CACHE_FLUSH_IS_SAFE   1
+
+struct pt_regs;
+extern volatile int kgdb_hwbreak_sstep[NR_CPUS];
+extern void smp_send_nmi_allbutself(void);
+extern void kgdb_wait_ipi(struct pt_regs *);
+#endif /* _ASM_KGDB_H_ */
+#endif /* __KERNEL__ */
diff -puN lib/Kconfig.debug~ia64-lite lib/Kconfig.debug
--- linux-2.6.13/lib/Kconfig.debug~ia64-lite 2005-08-08 12:18:23.000000000 -0700
+++ linux-2.6.13-trini/lib/Kconfig.debug      2005-08-10 10:54:53.000000000 -0700
@@ -163,7 +163,7 @@ config FRAME_POINTER
 config KGDB
     bool "KGDB: kernel debugging with remote gdb"
     select WANT_EXTRA_DEBUG_INFORMATION
-    depends on DEBUG_KERNEL && (X86 || MIPS32 || (!SMP || BROKEN) && PPC32)
+    depends on DEBUG_KERNEL && (X86 || MIPS32 || IA64 || (!SMP || BROKEN) && PPC32)
     help
         If you say Y here, it will be possible to remotely debug the
         kernel using gdb. It is strongly suggested that you enable
-
-
To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
the body of a message to majordomo@vger.kernel.org
More majordomo info at http://vger.kernel.org/majordomo-info.html
Please read the FAQ at http://www.tux.org/lkml/
```