

[PATCH 03/13] [RFC] ipath copy routines

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-12/msg05435.html>

- *From:* Roland Dreier <rolandd@xxxxxxxxxx>
 - *Date:* Fri, 16 Dec 2005 15:48:54 -0800
-

Copy routines for ipath driver

```
drivers/infiniband/hw/ipath/ipath_copy.c | 666 ++++++
drivers/infiniband/hw/ipath/ipath_dwordcpy.S | 62 ++
2 files changed, 728 insertions(+), 0 deletions(-)
create mode 100644 drivers/infiniband/hw/ipath/ipath_copy.c
create mode 100644 drivers/infiniband/hw/ipath/ipath_dwordcpy.S
```

```
99f636a78e0d759ab663a7abb29e6a71b32a552d
diff --git a/drivers/infiniband/hw/ipath/ipath_copy.c b/drivers/infiniband/hw/ipath/ipath_copy.c
new file mode 100644
index 0000000..26211ad
--- /dev/null
+++ b/drivers/infiniband/hw/ipath/ipath_copy.c
@@ -0,0 +1,666 @@
+/*
+ * Copyright (c) 2003, 2004, 2005. PathScale, Inc. All rights reserved.
+ *
+ * This software is available to you under a choice of one of two
+ * licenses. You may choose to be licensed under the terms of the GNU
+ * General Public License (GPL) Version 2, available from the file
+ * COPYING in the main directory of this source tree, or the
+ * OpenIB.org BSD license below:
+ *
+ * Redistribution and use in source and binary forms, with or
+ * without modification, are permitted provided that the following
+ * conditions are met:
+ *
+ * - Redistributions of source code must retain the above
+ * copyright notice, this list of conditions and the following
+ * disclaimer.
+ *
+ * - Redistributions in binary form must reproduce the above
+ * copyright notice, this list of conditions and the following
+ * disclaimer in the documentation and/or other materials
+ * provided with the distribution.
+ */
```

[PATCH 03/13] [RFC] ipath copy routines

```
+ * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,  
+ * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF  
+ * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND  
+ * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS  
+ * BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN  
+ * ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN  
+ * CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE  
+ * SOFTWARE.  
+ *  
+ * Patent licenses, if any, provided herein do not apply to  
+ * combinations of this program with other software, or any other  
+ * product whatsoever.  
+ *  
+ * $Id: ipath_copy.c 4365 2005-12-10 00:04:16Z rjwalsh $  
+ */  
+  
+/*  
+ * This file provides support for doing sk_buff buffer swapping between  
+ * the low level driver eager buffers, and the network layer. It's part  
+ * of the core driver, rather than the ether driver, because it relies  
+ * on variables and functions in the core driver. It exports a single  
+ * entry point for use in the ipath_ether module.  
+ */  
+  
+#include <linux/kernel.h>  
+#include <linux/errno.h>  
+#include <linux/types.h>  
+#include <asm/io.h>  
+#include <asm/byteorder.h>  
+#include <asm/bitops.h>  
+#include <linux/skbuff.h>  
+#include <linux/netdevice.h>  
+  
+#include <linux/crc32.h> /* we can generate our own crc's for testing */  
+  
+#include "ipath_kernel.h"  
+#include "ips_common.h"  
+#include "ipath_layer.h"  
+  
+#define TRUE 1  
+#define FALSE 0  
+  
+/*  
+ * Allocate a PIO send buffer, initialize the header and copy it out.  
+ */  
+static int layer_send_getpiobuf(struct copy_data_s *cdp)  
+{  
+ int whichpb;  
+ uint32_t device = cdp->device;  
+ uint32_t extra_bytes;  
+ uint32_t len, nwords;
```

[PATCH 03/13] [RFC] ipath copy routines

```
+ uint32_t *piobuf;
+
+ whichpb = ipath_getpiobuf(device);
+ if (whichpb < 0) {
+ cdp->error = whichpb;
+ return whichpb;
+ }
+
+ /*
+ * Compute the max amount of data that can fit into a PIO buffer.
+ * buffer size - header size - trigger qword length & flags - CRC
+ */
+ len = devdata[device].ipath_ibmaxlen -
+ sizeof(ether_header_typ) - 8 - (SIZE_OF_CRC << 2);
+ if (len > (cdp->len + cdp->extra))
+ len = (cdp->len + cdp->extra);
+ /* Compute word alignment (i.e., (len & 3) ? 4 - (len & 3) : 0) */
+ extra_bytes = (4 - len) & 3;
+ nwords = (sizeof(ether_header_typ) + len + extra_bytes) >> 2;
+ cdp->hdr->lrh[2] = htons(nwords + SIZE_OF_CRC);
+ cdp->hdr->bth[0] = htonl((OPCODE_ITH4X << 24) + (extra_bytes << 20) +
+ IPS_DEFAULT_P_KEY);
+ cdp->hdr->sub_opcode = OPCODE_ENCAP;
+
+ cdp->hdr->bth[2] = 0;
+ /* Generate an interrupt on the receive side for the last fragment. */
+ cdp->hdr->iph.pkt_flags = ((cdp->len+cdp->extra) == len) ? INFINIPATH_KPF_INTR : 0;
+ cdp->hdr->iph.chksum =
+ (uint16_t) IPS_LRH_BTH +
+ (uint16_t) (nwords + SIZE_OF_CRC) -
+ (uint16_t) ((cdp->hdr->iph.ver_port_tid_offset >> 16) & 0xFFFF) -
+ (uint16_t) (cdp->hdr->iph.ver_port_tid_offset & 0xFFFF) -
+ (uint16_t) cdp->hdr->iph.pkt_flags;
+
+ piobuf = (uint32_t *) (((char *) (devdata[device].ipath_kregbase)) +
+ devdata[device].ipath_piobufbase +
+ whichpb * devdata[device].ipath_palign);
+ _IPATH_VDBG("send %d (%x %x %x %x %x %x %x %x)\n", nwords,
+ cdp->hdr->lrh[0],
+ cdp->hdr->lrh[1],
+ cdp->hdr->lrh[2],
+ cdp->hdr->lrh[3],
+ cdp->hdr->bth[0], cdp->hdr->bth[1], cdp->hdr->bth[2]);
+ /*
+ * Write len to control qword, no flags.
+ * +1 is for the qword padding of pbc.
+ */
+ *((uint64_t *) piobuf) = (uint64_t) (nwords + 1);
+ piobuf += 2;
+ ipath_dwordcpy(piobuf, (uint32_t *) cdp->hdr,
+ sizeof(ether_header_typ) >> 2);
```

[PATCH 03/13] [RFC] ipath copy routines

```
+ cdp->csum_pio = &((ether_header_typ *) piobuf)->csum;
+ cdp->to = piobuf + (sizeof(ether_header_typ) >> 2);
+ cdp->flen = nwords - (sizeof(ether_header_typ) >> 2);
+ cdp->hdr->frag_num++;
+ return 0;
+}
+
+/*
+ * Copy data out of one or a chain of sk_buffs, into the PIO buffer.
+ * Fragment an sk_buff into multiple IB packets if the amount of data is
+ * more than a single eager send.
+ * Offset and len are in bytes.
+ * Note that this function is recursive!
+ */
+static void copy_bits(const struct sk_buff *skb, unsigned int offset,
+ unsigned int len, struct copy_data_s *cdp)
+{
+ unsigned int start = skb_headlen(skb);
+ unsigned int i, copy;
+ uint32_t n;
+ u8 *p;
+
+ /* Copy header. */
+ if ((int)(copy = start - offset) > 0) {
+ if (copy > len)
+ copy = len;
+ p = skb->data + offset;
+ offset += copy;
+ len -= copy;
+ /*
+ * If the alignment buffer is not empty, fill it and write
+ * it out.
+ */
+ if (cdp->extra) {
+ if (cdp->extra == 4)
+ goto extra_copy_bits_done;
+
+ while (copy != 0) {
+ cdp->u.buf[cdp->extra] = *p++;
+ copy--;
+ cdp->offset++;
+ cdp->len--;
+
+ if (++cdp->extra == 4) {
+ extra_copy_bits_done:
+ if (cdp->flen == 0)
+ && layer_send_getpiobuf(cdp) < 0)
+ return;
+ *cdp->to++ = cdp->u.w;
+ cdp->extra = 0;
+ cdp->flen -= 1;

```

```

+ break;
+ }
+ }
+ }
+ while (copy >= 4) {
+ if (cdp->flen == 0 && layer_send_getpiobuf(cdp) < 0)
+ return;
+ n = copy >> 2;
+ if (n > cdp->flen)
+ n = cdp->flen;
+ ipath_dwordcpy(cdp->to, (uint32_t *) p, n);
+ cdp->to += n;
+ cdp->flen -= n;
+ n <<= 2;
+ p += n;
+ cdp->offset += n;
+ cdp->len -= n;
+ copy -= n;
+ }
+ /*
+ * Either cdp->extra is zero or copy is zero which means that
+ * the loop here can't cause the alignment buffer to fill up.
+ */
+ while (copy != 0) {
+ cdp->u.buf[cdp->extra++] = *p++;
+ copy--;
+ cdp->offset++;
+ cdp->len--;
+
+ }
+ if (len == 0)
+ return;
+ }
+
+ for (i = 0; i < skb_shinfo(skb)->nr_frags; i++) {
+ skb_frag_t *frag = &skb_shinfo(skb)->frags[i];
+ unsigned int end;
+
+ end = start + frag->size;
+ if ((int)(copy = end - offset) > 0) {
+ u8 *vaddr;
+
+ if (copy > len)
+ copy = len;
+ vaddr = kmap_skb_frag(frag);
+ p = vaddr + frag->page_offset + offset - start;
+ offset += copy;
+ len -= copy;
+ }
+ /*
+ * If the alignment buffer is not empty, fill
+ * it and write it out.

```

```

+ */
+ if (cdp->extra) {
+ if (cdp->extra == 4)
+ goto extra1_copy_bits_done;
+
+ while (copy != 0) {
+ cdp->u.buf[cdp->extra] = *p++;
+ copy--;
+ cdp->offset++;
+ cdp->len--;
+
+ if (++cdp->extra == 4) {
+extra1_copy_bits_done:
+ if (cdp->flen == 0
+ && layer_send_getpiobuf(cdp)
+ < 0)
+ return;
+ *cdp->to++ = cdp->u.w;
+ cdp->extra = 0;
+ cdp->flen -= 1;
+ break;
+ }
+ }
+ }
+ while (copy >= 4) {
+ if (cdp->flen == 0
+ && layer_send_getpiobuf(cdp) < 0)
+ return;
+ n = copy >> 2;
+ if (n > cdp->flen)
+ n = cdp->flen;
+ ipath_dwordcpy(cdp->to, (uint32_t *) p, n);
+ cdp->to += n;
+ cdp->flen -= n;
+ n <<= 2;
+ p += n;
+ cdp->offset += n;
+ cdp->len -= n;
+ copy -= n;
+ }
+ /*
+ * Either cdp->extra is zero or copy is zero
+ * which means that the loop here can't cause
+ * the alignment buffer to fill up.
+ */
+ while (copy != 0) {
+ cdp->u.buf[cdp->extra++] = *p++;
+ copy--;
+ cdp->offset++;
+ cdp->len--;
+ }

```

[PATCH 03/13] [RFC] ipath copy routines

```
+ kunmap_skb_frag(vaddr);
+
+ if (len == 0)
+ return;
+ }
+ start = end;
+ }
+
+ if (skb_shinfo(skb)->frag_list) {
+ struct sk_buff *list = skb_shinfo(skb)->frag_list;
+
+ for (; list; list = list->next) {
+ unsigned int end;
+
+ end = start + list->len;
+ if ((int)(copy = end - offset) > 0) {
+ if (copy > len)
+ copy = len;
+ copy_bits(list, offset - start, copy, cdp);
+ if (cdp->error || (len -= copy) == 0)
+ return;
+ }
+ start = end;
+ }
+ }
+ if (len)
+ cdp->error = -EFAULT;
+}
+
+/*
+ * Copy data out of one or a chain of sk_buffs, into the PIO buffer, generating
+ * the checksum as we go.
+ * Fragment an sk_buff into multiple IB packets if the amount of data is
+ * more than a single eager send.
+ * Offset and len are in bytes.
+ * Note that this function is recursive!
+ */
+static void copy_and_csum_bits(const struct sk_buff *skb, unsigned int offset,
+ unsigned int len, struct copy_data_s *cdp)
+{
+ unsigned int start = skb_headlen(skb);
+ unsigned int i, copy;
+ unsigned int csum2;
+ uint32_t n;
+ u8 *p;
+
+ /* Copy header. */
+ if ((int)(copy = start - offset) > 0) {
+ if (copy > len)
+ copy = len;
+ p = skb->data + offset;
```

```

+ offset += copy;
+ len -= copy;
+ if (!cdp->checksum_calc) {
+ cdp->checksum_calc = TRUE;
+
+ csum2 = csum_partial(p, copy, 0);
+ cdp->csum = csum_block_add(cdp->csum, csum2, cdp->pos);
+ cdp->pos += copy;
+ }
+ /*
+ * If the alignment buffer is not empty, fill it and
+ * write it out.
+ */
+ if (cdp->extra) {
+ if (cdp->extra == 4)
+ goto extra_copy_and_csum_bits_done;
+
+ while (copy != 0) {
+ cdp->u.buf[cdp->extra] = *p++;
+ copy--;
+ cdp->offset++;
+ cdp->len--;
+ if (++cdp->extra == 4) {
+extra_copy_and_csum_bits_done:
+ if (cdp->flen == 0
+ && layer_send_getpiobuf(cdp) < 0)
+ return;
+ /*
+ * write the checksum before
+ * the last PIO write.
+ */
+ if (cdp->flen == 1) {
+ *cdp->csum_pio =
+ csum_fold(cdp->csum);
+ mb();
+ }
+ *cdp->to++ = cdp->u.w;
+ cdp->extra = 0;
+ cdp->flen -= 1;
+ break;
+ }
+ }
+ }
+
+ while (copy >= 4) {
+ if (cdp->flen == 0 && layer_send_getpiobuf(cdp) < 0)
+ return;
+
+ n = copy >> 2;
+ if (n > cdp->flen)
+ n = cdp->flen;

```

[PATCH 03/13] [RFC] ipath copy routines

```
+ /* write the checksum before the last PIO write. */
+ if (cdp->flen == n) {
+ *cdp->csum_pio = csum_fold(cdp->csum);
+ mb();
+ }
+ ipath_dwordcopy(cdp->to, (uint32_t *) p, n);
+ cdp->to += n;
+ cdp->flen -= n;
+ n <<= 2;
+ p += n;
+ cdp->offset += n;
+ cdp->len -= n;
+ copy -= n;
+ }
+ /*
+ * Either cdp->extra is zero or copy is zero which means that
+ * the loop here can't cause the alignment buffer to fill up.
+ */
+ while (copy != 0) {
+ cdp->u.buf[cdp->extra++] = *p++;
+ copy--;
+ cdp->offset++;
+ cdp->len--;
+ }
+
+ cdp->checksum_calc = FALSE;
+
+ if (len == 0)
+ return;
+ }
+
+ for (i = 0; i < skb_shinfo(skb)->nr_frags; i++) {
+ skb_frag_t *frag = &skb_shinfo(skb)->frags[i];
+ unsigned int end;
+
+ end = start + frag->size;
+ if ((int)(copy = end - offset) > 0) {
+ u8 *vaddr;
+
+ if (copy > len)
+ copy = len;
+ vaddr = kmap_skb_frag(frag);
+ p = vaddr + frag->page_offset + offset - start;
+ offset += copy;
+ len -= copy;
+
+ if (!cdp->checksum_calc) {
+ cdp->checksum_calc = TRUE;
+
+ csum2 = csum_partial(p, copy, 0);
+ cdp->csum = csum_block_add(cdp->csum, csum2,
```

```

+ cdp->pos);
+ cdp->pos += copy;
+ }
+ /*
+ * If the alignment buffer is not empty, fill
+ * it and write it out.
+ */
+ if (cdp->extra) {
+ if (cdp->extra == 4)
+ goto extra1_copy_and_csum_bits_done;
+ while (copy != 0) {
+ cdp->u.buf[cdp->extra] = *p++;
+ copy--;
+ cdp->offset++;
+ cdp->len--;
+
+ if (++cdp->extra == 4) {
+extra1_copy_and_csum_bits_done:
+ if (cdp->flen == 0
+ && layer_send_getpiobuf(cdp)
+ < 0) {
+ kunmap_skb_frag(vaddr);
+ return;
+ }
+ /*
+ * write the checksum
+ * before the last PIO
+ * write.
+ */
+ if (cdp->flen == 1) {
+ *cdp->csum_pio =
+ csum_fold(cdp->
+ csum);
+ mb();
+ }
+ *cdp->to++ = cdp->u.w;
+ cdp->extra = 0;
+ cdp->flen -= 1;
+ break;
+ }
+ }
+ }
+ while (copy >= 4) {
+ if (cdp->flen == 0
+ && layer_send_getpiobuf(cdp) < 0) {
+ kunmap_skb_frag(vaddr);
+ return;
+ }
+ n = copy >> 2;
+ if (n > cdp->flen)
+ n = cdp->flen;

```

[PATCH 03/13] [RFC] ipath copy routines

```
+ /*
+ * write the checksum before the last
+ * PIO write.
+ */
+ if (cdp->flen == n) {
+ *cdp->csum_pio = csum_fold(cdp->csum);
+ mb();
+ }
+ ipath_dwordcpy(cdp->to, (uint32_t *) p, n);
+ cdp->to += n;
+ cdp->flen -= n;
+ n <<= 2;
+ p += n;
+ cdp->offset += n;
+ cdp->len -= n;
+ copy -= n;
+ }
+ /*
+ * Either cdp->extra is zero or copy is zero
+ * which means that the loop here can't cause
+ * the alignment buffer to fill up.
+ */
+ while (copy != 0) {
+ cdp->u.buf[cdp->extra++] = *p++;
+ copy--;
+ cdp->offset++;
+ cdp->len--;
+ }
+ kunmap_skb_frag(vaddr);
+
+ cdp->checksum_calc = FALSE;
+
+ if (len == 0)
+ return;
+ }
+ start = end;
+ }
+
+ if (skb_shinfo(skb)->frag_list) {
+ struct sk_buff *list = skb_shinfo(skb)->frag_list;
+
+ for (; list; list = list->next) {
+ unsigned int end;
+
+ end = start + list->len;
+ if ((int)(copy = end - offset) > 0) {
+ if (copy > len)
+ copy = len;
+ copy_and_csum_bits(list, offset - start, copy, cdp);
+ if (cdp->error || (len -= copy) == 0)
+ return;

```

[PATCH 03/13] [RFC] ipath copy routines

```
+ offset += copy;
+ }
+ start = end;
+ }
+ }
+ if (len)
+ cdp->error = -EFAULT;
+ }
+
+ /*
+ * Note that the header should have the unchanging parts
+ * initialized but the rest of the header is computed as needed in
+ * order to break up skb data buffers larger than the hardware MTU.
+ * In other words, the Linux network stack MTU can be larger than the
+ * hardware MTU.
+ */
+int ipath_layer_send_skb(struct copy_data_s *cdata)
+{
+ int ret = 0;
+ uint16_t vlslnh;
+ int device = cdata->device;
+
+ if (device >= infinipath_max) {
+ _IPATH_INFO("Invalid unit %u, failing\n", device);
+ return -EINVAL;
+ }
+ if (!(devdata[device].ipath_flags & IPATH_RCVHDRSZ_SET)) {
+ _IPATH_INFO("send while not open\n");
+ ret = -EINVAL;
+ } else
+ if ((devdata[device].ipath_flags & (IPATH_LINKUNK | IPATH_LINKDOWN))
+ || devdata[device].ipath_lid == 0) {
+ /* lid check is for when sma hasn't yet configured */
+ ret = -ENETDOWN;
+ _IPATH_VDBG("send while not ready, mylid=%u, flags=0x%x\n",
+ devdata[device].ipath_lid,
+ devdata[device].ipath_flags);
+ }
+ vlslnh = *((uint16_t *) cdata->hdr);
+ if (vlslnh != htons(IPS_LRH_BTH)) {
+ _IPATH_DBG("Warning: lrh[0] wrong (%x, not %x); not sending\n",
+ vlslnh, htons(IPS_LRH_BTH));
+ ret = -EINVAL;
+ }
+ if (ret)
+ goto done;
+
+ cdata->error = 0; /* clear last calls error */
+
+ if (cdata->skb->ip_summed == CHECKSUM_HW) {
+ unsigned int csstart = cdata->skb->h.raw - cdata->skb->data;
```

```

+
+ /*
+ * Computing the checksum is a bit tricky since if we fragment
+ * the packet, the fragment that should contain the checksum
+ * will have already been sent. The solution is to
+ * store the checksum in the header of the last fragment
+ * just before we write the last data word which triggers
+ * the last fragment to be sent. The receiver will
+ * check the header "tag" field, see that there is a
+ * checksum, and store the checksum back into the packet.
+ *
+ * Save the offset of the two byte checksum.
+ * Note that we have to add 2 to account for the two
+ * bytes of the ethernet address we stripped from the
+ * packet and put in the header.
+ */
+ cdata->hdr->csum_offset = csstart + cdata->skb->csum + 2;
+
+ if (cdata->offset < csstart)
+ copy_bits(cdata->skb, cdata->offset,
+ csstart - cdata->offset, cdata);
+
+ if (cdata->error) {
+ return (cdata->error);
+ }
+
+ if (cdata->offset < cdata->skb->len)
+ copy_and_csum_bits(cdata->skb, cdata->offset,
+ cdata->skb->len - cdata->offset,
+ cdata);
+
+ if (cdata->error) {
+ return (cdata->error);
+ }
+
+ if (cdata->extra) {
+ while (cdata->extra < 4)
+ cdata->u.buf[cdata->extra++] = 0;
+ if (cdata->flen != 0
+ || layer_send_getpiobuf(cdata) >= 0) {
+ /*
+ * write the checksum before the last
+ * PIO write.
+ */
+ *cdata->csum_pio = csum_fold(cdata->csum);
+ mb();
+ *cdata->to = cdata->u.w;
+ }
+ }
+ } else {

```

[PATCH 03/13] [RFC] ipath copy routines

```
+ copy_bits(cdata->skb, cdata->offset,
+ cdata->skb->len - cdata->offset, cdata);
+
+ if (cdata->error) {
+ return (cdata->error);
+ }
+
+ if (cdata->extra) {
+ while (cdata->extra < 4)
+ cdata->u.buf[cdata->extra++] = 0;
+ if (cdata->flen != 0
+ || layer_send_getpiobuf(cdata) >= 0)
+ *cdata->to = cdata->u.w;
+ }
+ }
+
+ if (cdata->error) {
+ ret = cdata->error;
+ if (cdata->error != -EBUSY)
+ /* just means no PIO buffers available */
+ _IPATH_UNIT_ERROR(device,
+ "layer_send copy_bits failed with error %d\n",
+ -ret);
+ }
+
+ ipath_stats.sps_ether_spkts++; /* another ether packet sent */
+
+done:
+ return ret;
+}
+
+EXPORT_SYMBOL(ipath_layer_send_skb);
diff --git a/drivers/infiniband/hw/ipath/ipath_dwordcpy.S b/drivers/infiniband/hw/ipath/ipath_dwordcpy.S
new file mode 100644
index 0000000..fdd8ec7
--- /dev/null
+++ b/drivers/infiniband/hw/ipath/ipath_dwordcpy.S
@@ -0,0 +1,62 @@
+/*
+ * Copyright (c) 2003, 2004, 2005. PathScale, Inc. All rights reserved.
+ *
+ * This software is available to you under a choice of one of two
+ * licenses. You may choose to be licensed under the terms of the GNU
+ * General Public License (GPL) Version 2, available from the file
+ * COPYING in the main directory of this source tree, or the
+ * OpenIB.org BSD license below:
+ *
+ * Redistribution and use in source and binary forms, with or
+ * without modification, are permitted provided that the following
+ * conditions are met:
+ *
+ */
```

[PATCH 03/13] [RFC] ipath copy routines

```
+ * – Redistributions of source code must retain the above
+ * copyright notice, this list of conditions and the following
+ * disclaimer.
+ *
+ * – Redistributions in binary form must reproduce the above
+ * copyright notice, this list of conditions and the following
+ * disclaimer in the documentation and/or other materials
+ * provided with the distribution.
+ *
+ * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
+ * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
+ * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
+ * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
+ * BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
+ * ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
+ * CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
+ * SOFTWARE.
+ *
+ * Patent licenses, if any, provided herein do not apply to
+ * combinations of this program with other software, or any other
+ * product whatsoever.
+ *
+ * $Id: ipath_dwordcpy.S 4365 2005-12-10 00:04:16Z rjwalsh $
+ */
+
+/*
+ * ipath_dwordcpy – Copy a memory block, primarily for writing to the
+ * InfiniPath PIO buffers, which only support dword multiple writes, and
+ * thus can not use memcpy(). For this reason, we use nothing smaller than
+ * dword writes.
+ * It is also used as a fast copy routine in some places that have been
+ * measured to win over memcpy, and the performance delta matters.
+ *
+ * Count is number of dwords; might not be a qword multiple.
+ */
+
+.globl ipath_dwordcpy
+/* rdi destination, rsi source, rdx count */
+ipath_dwordcpy:
+ movl %edx,%ecx
+ shrl $1,%ecx
+ andl $1,%edx
+ cld
+ rep
+ movsq
+ movl %edx,%ecx
+ rep
+ movsd
+ ret
+
+
+0.99.9n
```

—

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/>

- **Follow-Ups:**

- ◆ **[Re: \[PATCH 03/13\] \[RFC\] ipath copy routines](#)**
 - ◇ From: Andrew Morton
- ◆ **[Re: \[PATCH 03/13\] \[RFC\] ipath copy routines](#)**
 - ◇ From: Christoph Hellwig
- ◆ **[Re: \[PATCH 03/13\] \[RFC\] ipath copy routines](#)**
 - ◇ From: Pekka Enberg
- ◆ **[\[PATCH 04/13\] \[RFC\] ipath LLD core, part 1](#)**
 - ◇ From: Roland Dreier

- **References:**

- ◆ **[\[PATCH 02/13\] \[RFC\] ipath debug header](#)**
 - ◇ From: Roland Dreier

- Prev by Date: **[\[PATCH 04/13\] \[RFC\] ipath LLD core, part 1](#)**
- Next by Date: **[\[PATCH 08/13\] \[RFC\] ipath core last bit](#)**
- Previous by thread: **[\[PATCH 02/13\] \[RFC\] ipath debug header](#)**
- Next by thread: **[\[PATCH 04/13\] \[RFC\] ipath LLD core, part 1](#)**
- Index(es):
 - ◆ **[Date](#)**
 - ◆ **[Thread](#)**