(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 103473057 A
(43)申请公布日 2013.12.25

(21)申请号 201310408259.5

(22)申请日 2013.09.10

(71)申请人 江苏中科梦兰电子科技有限公司
地址 215500 江苏省苏州市常熟市虞山镇梦兰村

(72)发明人 张福新 陈杰 王锐 吴少刚 张斌 晏华

(74)专利代理机构 南京苏高专利商标事务所（普通合伙）32204
代理人 柏尚春

(51)Int.Cl.
G06F 9/44(2006.01)

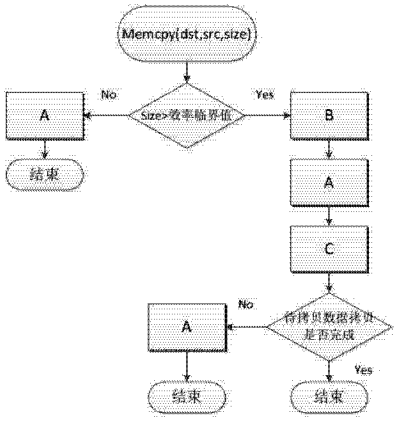
权利要求书1页 说明书3页 附图1页

(54)发明名称

一种 memcpy 函数的优化方法

(57)摘要

本发明公开了一种 memcpy 函数的优化方法，包括以下步骤：1）通过单字节拷贝指令拷贝 len 长度的待拷贝数据，使源地址/目的地址中至少一个满足 N 字节对齐，所述 N 为系统中位宽最高的指令一次能处理的字节个数，即拷贝后源地址/目的地址满足（x,N）或者（N,x）对齐，x 为 1，2，...，N；2）对于（x,N）对齐或者（N,x）对齐，分别以 x 字节指令或 N 字节指令从源地址读取数据存入寄存器，再由 N 字节指令或者 x 字节指令从寄存器读取数据存入目的地址，剩余待拷贝数据不足 N 字节时由单字节指令进行拷贝。该方案使用尽可能高的位宽指令进行拷贝，提升了拷贝效率。



CN 103473057 A

1. 一种 memcpy 函数的优化方法，其特征在于，包括以下步骤：

1）通过单字节拷贝指令拷贝 len 长度的待拷贝数据，使源地址 / 目的地址中至少一个满足 N 字节对齐，所述 N 为系统中位宽最高的指令一次能处理的字节个数，即拷贝后源地址 / 目的地址满足 (x,N) 或者 (N,x) 对齐，x 为 1,2,...,N；

2）对于 (x, N) 对齐，以 x 字节指令从源地址读取数据存入寄存器，再由 N 字节指令从寄存器读取数据存入目的地址，剩余待拷贝数据不足 N 字节时由单字节指令进行拷贝；对于 (N,x) 对齐，以 N 字节指令从源地址读取数据存入寄存器，再由 x 字节指令从寄存器读取数据存入目的地址，剩余待拷贝数据不足 N 字节时由单字节指令进行拷贝。

2. 根据权利要求 1 所述的 memcpy 函数的优化方法，其特征在于：所述 len 为源地址 / 目的地址除于 N 所得较大余数与 N 之间的差值。

3. 根据权利要求 1 所述的 memcpy 函数的优化方法，其特征在于：所述步骤 1）之前判断待拷贝数据长度是否大于效率临界值，待拷贝数据长度大于效率临界值时，进入步骤 1）；待拷贝数据长度小于等于效率临界值时，由单字节拷贝指令完成拷贝。

一种 memcpy 函数的优化方法

技术领域

[0001] 本发明涉及一种拷贝函数的优化方法，特别是涉及一种 memcpy 函数的优化方法。

背景技术

[0002] Memcpy 函数是 C 语言的标准函数，也是经常用的函数，其作用将一份内存中的数据复制到另一内存位置。

[0003] 现有优化技术中，通过在 memcpy 函数的实现中使用高位宽的指令，例如使用一次复制 8 字节的指令替代一次复制 4 字节的指令，从而提高效率。

[0004] 然而，高位宽的指令对地址的对齐有较高的要求。通常位宽为 N 的指令，对内存地址的对齐要求也是 N。当 memcpy 函数的源地址 / 目的地址不满足对齐要求时，在不同的硬件平台上，或是不能使用这些高位宽的指令、或是指令的效能出现大幅度下降。

发明内容

[0005] 针对上述现有技术的不足，本发明的目的是提供一种 memcpy 函数的优化方法，尽可能源地址 / 目的地址达到高位宽对齐，从而使用高位宽读写指令完成拷贝任务，提升 memcpy 函数效率。

[0006] 本发明的技术方案是这样的：一种 memcpy 函数的优化方法，其特征在于，包括以下步骤：

[0007] 1）通过单字节拷贝指令拷贝 len 长度的待拷贝数据，使源地址 / 目的地址中至少一个满足 N 字节对齐，所述 N 为系统中位宽最高的指令一次能处理的字节个数，即拷贝后源地址 / 目的地址满足 (x,N) 或者 (N,x) 对齐，x 为 1,2,...,N；

[0008] 2)对于 (x,N) 对齐，以 x 字节指令从源地址读取数据存入寄存器，再由 N 字节指令从寄存器读取数据存入目的地址，剩余待拷贝数据不足 N 字节时由单字节指令进行拷贝；对于 (N,x) 对齐，以 N 字节指令从源地址读取数据存入寄存器，再由 x 字节指令从寄存器读取数据存入目的地址，剩余待拷贝数据不足 N 字节时由单字节指令进行拷贝。

[0009] 在本发明的一个具体实施例中，所述 len 为源地址 / 目的地址除于 N 所得较大余数与 N 之间的差值。

[0010] 在本发明的另一个具体实施例中，所述步骤 1)之前判断待拷贝数据长度是否大于效率临界值，待拷贝数据长度大于效率临界值时，进入步骤 1)；待拷贝数据长度小于等于效率临界值时，由单字节拷贝指令完成拷贝。

[0011] 本发明所提供的技术方案，针对源地址 / 目的地址较差的对齐情况，通过对源地址 / 目的地址对齐情况的“升级”，使用尽可能高的位宽指令进行读写来完成拷贝，相比原对齐状况下使用受限指令的拷贝提升了拷贝效率，减少了拷贝延时，增强系统性能。通过判断待拷贝数据长度是否超过效率临界值，确定由优化指令拷贝还是由单字节拷贝指令，避免由于待拷贝数据长度过短，造成优化指令本身的效率损失无法通过整个拷贝过程效率的提升所弥补而使得优化失败，提升优化指令的适用性。

附图说明

[0012] 图 1 为本发明 memcpy 函数优化后数据处理流程示意图。

具体实施方式

[0013] 下面结合实施例对本发明作进一步说明，但不作为对本发明的限定。

[0014] 请参见图 1，首先对 memcpy 函数的参数说明：memcpy(dst, src, size)，从内存地址 src（源地址）开始，复制 size 个字节到内存地址 dst（目标地址）。其中，如果 src/dst 整除 4，则称为源地址 / 目标地址为 4 字节对齐，如果 src/dst 整除 8，则称为源地址 / 目标地址为 8 字节对齐，依次类推。

[0015] 现假设目前系统中位宽最高的指令一次能处理 N 个字节，如果 src/dst 均是 N 字节对齐的，则效能最高。本发明的优化主要针对 src/dst 不满足 N 字节对齐的情况，先以单字节拷贝指令进行少量字节拷贝，使得拷贝后，src/dst 的对齐状况尽可能地“升级”。例如 N=16，在 src=15、dst=31 情况下，两地址均为非 16 字节对齐，进行单字节拷贝效率较低。在这里，我们先完成一个字节的拷贝，使得 src=16，dst=32，此时两者都是 16 字节对齐，可以使用 16 字节位宽指令来进行操作，以此提高效率。

[0016] 具体地，在本实施例中，src/dst 的对齐“升级”首先判断 src/dst 除于 N 所得余数中较大者，由此较大余数与 N 之间的差值确定单字节拷贝长度；采用单字节拷贝命令对待拷贝数据进行拷贝，使源地址 / 目标地址逐一变化。“升级”后的 src/dst 的对齐状态必然为 (x, N) 或者 (N, x)；x 取值为 1，2，4，...，N，表示该地址为 x 字节对齐，即“升级后”，src/dst 中至少有一个地址满足 N 对齐，另一个地址则依据具体情况变为 1 字节对齐或者 2 字节对齐或者 4 字节对齐，依次类推，出现的最佳状况为“升级”后，源地址 / 目的地址均为 N 字节对齐。此处 x 的由 src/dst 除于 N 所得的余数差确定，余数差中所有 2 因子的乘积为 x，余数差为 0 时，x 取 N，例如余数差为 1 时，具有 0 个 2 因子，$x=2^0=1$，余数差为 10 时，具有 1 个 2 和一个 5 因子，$x=2^1=2$。升级后的 src/dst，对于 (x, N) 对齐，以 x 字节指令从源地址读取数据存入寄存器，再由 N 字节指令从寄存器读取数据存入目的地址，剩余待拷贝数据不足 N 字节时由单字节指令进行拷贝；对于 (N, x) 对齐，以 N 字节指令从源地址读取数据存入寄存器，再由 x 字节指令从寄存器读取数据存入目的地址，剩余待拷贝数据不足 N 字节时由单字节指令进行拷贝。

[0017] 一个更为完善，效率更高的 memcpy 函数优化后的处理方式是这样的：请结合图 1，memcpy 中含有三个模块：

[0018] A. 任意对齐状况拷贝模块，该模块能够处理任意对齐状况，但效率较差；

[0019] B. 升级参数的计算模块，计算“升级”后的对齐状况是 (x, N) 还是 (N, x) 并确定 x 取值；计算“升级”所需拷贝的长度；

[0020] C. 固定对齐状况拷贝模块，含有“$2*\log_2N+1$”个代码块对应处理 (1, N)、(2, N)、(4, N)，...(N, N) 以及 (N, N/2)、...(N, 2)、(N, 1) 这“$2*\log_2N+1$”种对齐状况；代码块中尽可能使用高位宽指令。

[0021] 开始拷贝处理时，首先判断 memcpy 参数中，size 大小是否足够，即待拷贝数据长度是否够长。因为后续的优化处理需要消耗一定的计算时间，size 太小，优化处理本身所

耗时间无法从优化所获得的时间中得到弥补，则优化效果不明显甚至可能更慢。所以此处可以设定一个常数，该常数为一个效率临界值，当 size 小于该常数时，拷贝任务直接由 A 模块完成，当 size 大于该常数时，进入 B 模块处理，B 模块内容以伪代码表示为：

[0022] a=src%N 取源地址除于 N 后的余数

[0023] b=dst%N 取目的地址除于 N 后的余数

[0024] len=N-MAX(a,b) “升级”过程需拷贝的数据长度

[0025] 对于 $2*\log_2N+1$ 种对齐状况，可以用 Y 位（二进制位）数来表示。

[0026] Y=int(log2(2*log2N+1))+1

[0027] 升级后状况取的是 (N,x) 形式，还是 (x,N) 形式，并将对齐情况编码成 index

[0028] code1=cmp(a,b) a>=b 时，cmp 返回 1，否则返回 0

[0029] code2 表示 (N,x)/(x,N) 中 x 的值

[0030] diff=a－b 确定余数差

[0031] 计算余数差中 2 因子的个数。实际上就是二进制表达下，低位连续的 0 的个数。这里用 ctz 函数（count trailing zeros）来表达这个功能。

[0032] set(diff,$\log_2N+1$) 设置第 $\log_2N+1$ 位，处理 diff 为 0 的特殊情况

[0033] code2=ctz(diff)

[0034] 有的 CPU 直接支持等同功能指令，例如 intel 上 bsr 指令（从后往前寻找第一个被设置位的索引）则

[0035] code2=bsr(diff)

[0036] 有的 CPU 支持近似指令，需要作转换。例如 MIPS 上的 clz 指令（Count Leading Zeros）则通过 diff 和其相反数（计算机的补码表达）按位进行与操作，从而获得形如“000010000”的数。然后用寄存器总长 word_len，减去高位连续 0 的个数以及中间的 1，获得低位连续的 0 的个数

[0037] codec2=word_len-clz(diff&-diff)－1

[0038] 上述两种 CPU 下，index=(code1<<(Y-1))|code2。

[0039] B 模块完成后进入 A 模块，按照 B 模块获得的 len 进行相应长度的数据拷贝进行对齐“升级”，然后进入 C 模块。

[0040] C 模块中存在一个长度为 2^Y 的入口表（含有 $2*\log_2N+1$ 个有效项），表中每个有效项存储了 C 模块中处理固定对齐状况的代码块起始地址。对于 (x，N) 对齐，以 x 字节指令从源地址读取数据存入寄存器，再由 N 字节指令从寄存器读取数据存入目的地址；对于 (N,x) 对齐，以 N 字节指令从源地址读取数据存入寄存器，再由 x 字节指令从寄存器读取数据存入目的地址。用 B 模块获得的 index 来索引 C 模块入口表，便可得到 C 模块中需要用到的代码块起始地址，进行拷贝。

[0041] 而后判断待拷贝数据是否全部处理完成，如完成则整个函数结束，如没有完成则进入 A 模块，完成余下的不能用高位宽指令处理的数据拷贝至函数结束。

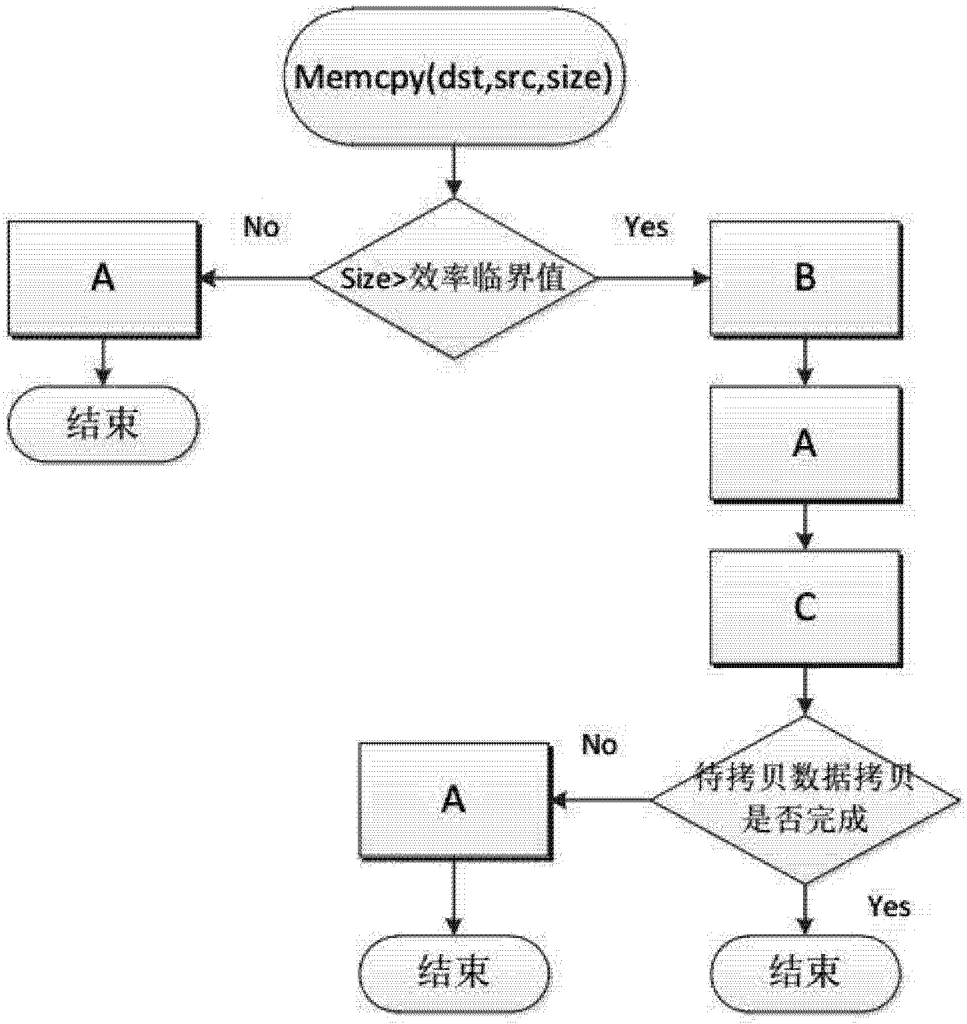

Memcpy(dst,src,size)
Size>效率临界值
No
Yes
A
B
结束
A
C
待拷贝数据拷贝是否完成
No
A
Yes
结束
结束

图 1