基于 SSE 指令的大内存快速拷贝*

钱昌松, 刘志刚, 刘代志
(第二炮兵工程学院, 陕西 西安 710025)

摘 要: 在深入研究单指令多数据流扩展指令集(Streaming SIMD Extensions, SSE)数据传输指令操作特点的基础上,充分考虑了数据预取、数据对齐、CPU 缓存和新的 128 位寄存器等因素,在 Visual C++平台上用嵌入汇编开发了内存拷贝函数。通过实验分析了各内存拷贝函数拷贝速度与拷贝内存量之间的对应关系。

关键词: 单指令多数据流扩展指令集; 内存拷贝; MMX; 代码优化
中图法分类号: TP319 文献标识码: A 文章编号: 1001-3695(2005)02-0113-02

Large Memory Fast Copy Technology Based on SSE Instructions

QIAN Chang-song, LIU Zhi-gang, LIU Dai-zhi
(Second Artillery Institute of Engineering, Xi an Shanxi 710025, China)

Abstract: A new memory copy function is developed after researching on the operating characters of Streaming SIMD Extensions (SSE) data transfer instructions and taking facts into account such as prefech, alignment, cache and new 128 bits registers. In the function, assemble language embedded in Visual C++ 6.0 is employed. An experiment is conducted to analyze the relations between the velocity of every memory copy function and the size of memory to be copied.

Key words: Streaming SIMD Extensions(SSE); Memory Copy; MMX; Code Optimization

1 引言

由于内存的速度比 CPU 的速度慢得多,在一些频繁进行大块内存拷贝的程序中,内存拷贝会消耗大量的时间,从而严重影响程序的性能。通常采用两种方法在现有 CPU 和内存的条件下解决这个问题:一是采用后台程序在 CPU 空闲时间进行内存拷贝;二是采用数据缓存提高数据存取速度。这些方法在一定程度上可以提高程序的性能,但是没有提高内存拷贝操作本身的速度。

自 Intel 公司推出 MMX 技术和单指令多数据流扩展指令集(Streaming SIMD Extensions, SSE)[1]技术以来,已经有人利用 MMX 开发内存拷贝函数 MMXMemoryCopy()[2],也有人利用 SSE 技术开发内存拷贝函数 memcpySSE()[3],其目的是为了提高内存拷贝操作本身的速度。实验结果表明,MMXMemoryCopy()在拷贝较大内存时速度并不比 Windows 提供的 memcpy()的速度快多少,而 memcpySSE()在数据没有对齐时拷贝速度却比 Windows 提供的函数的速度还小。

2 基于 SSE 的内存拷贝函数代码优化方法

SSE 即单指令多数据扩展指令集。由于在支持 SSE 指令的 CPU 中可以用一个 SSE 指令同时执行四个 32 位单精度浮点数的传输与运算操作,所以可用 SSE 指令优化内存拷贝函数代码以提高拷贝速度。用 SSE 指令优化函数代码时要注意以下几点:

(1) 数据对齐。Pentium II 和 Pentium III 的内存单元以 16 字节为边界,跨越边界的访问就是非对齐的,如果数据在运算之前不进行对齐,就会使指令运算产生大量的延时。例如对于已经存在于一级缓存中的数据来讲,一次非对齐访问将产生一次访问请求,这将消耗 6 ~9 个时钟周期,而对于二级缓存的非对齐访问则将消耗更多时钟周期,所以数据对齐在执行大量数据存取的代码中是非常关键的[4]。但是由于存在源数据地址和目的地址不可能同时对齐的情况,实际情况中只能对齐一个地址。实验表明:源数据地址和目的地址两者中,前者没有对齐对拷贝速度的影响很小,所以在程序设计中只考虑目的地址的对齐。由于要克服数据对齐方式对拷贝速度的影响,所以在选择数据传输指令时就不能选择那些仅传输对齐数据的指令如 MOVAPS。

(2) CPU 缓存和数据预取。为了避免内存访问速度成为 CPU 速度的瓶颈,现在的 CPU 都有缓存(Cache)的设计,甚至还有多层的缓存。例如 Intel 的 Pentium III 500MHz CPU 就设计有两级缓存。

CPU 在从内存中读取对齐数据时,先查询一级缓存,如果命中,则读取数据,操作没有额外的延时;如果没有命中,则继续查询二级缓存。如果在二级缓存中命中,则读取数据,操作大约会延时 10 个指令周期;如果没有命中,则在主存储器中读取数据(假设没有三级缓存),延时大约是 50 个指令周期。所以在内存拷贝时,如果在 CPU 读取数据之前将数据预先读取到一级缓存和二级缓存中,将可以大大提高命中率,减小读取数据造成的延时。另外 SSE 提供了 CPU 缓存数据预取指令,如 Prefetchnta 和 Prefetcht0 等,这些指令可以告诉 CPU 将稍后要处理的数据先读到缓存中,而且开销非常小,如执行 Prefetchnta 指令只需一个时钟周期,且没有延时。所以如果在

收稿日期: 2004-03-06; 修返日期: 2004-05-04
基金项目: 国家自然科学基金资助项目(40274044)

CPU 读取内存数据之前先用预取指令将数据预取到缓存中,将大大提高数据读取的速度。

(3) 128 位寄存器并行存取。SSE 定义了八个全新的 128 位寄存器[1],并提供了一套结合这 8 个 128 位寄存器的数据传输指令,这些指令的耗时特征[4]如表 1 所示。

表 1 指令的耗时特性

指令	操作	指令周期		延时	吞吐量
		P1	P2		
MOVQ	r64, m32/64	1			1/1
MOVQ	m32/64, r64		1		1/1
MOVNTQ	m64, r64		1		1/30 - 1/1
MOVAPS	r128, m128	2		2	1/2
MOVAPS	m128, r128		2	3	1/2
MOVUPS	r128, m128	4		2	1/4
MOVUPS	m128, r128		4	3	1/4
Prefechnta	m	1			
MOVNTPS	m128, r128		2		1/15 - 1/2

为了与 MMX 指令的耗时特征相比较,表 1 中也列出了对应的 MMX 64 位数据传输指令: MOVQ 和 MOVNTQ。其中 P1 和 P2 分别表示取数据和存数据时指令在执行端口消耗的指令周期, r 表示寄存器, m 表示寄存器, 32, 64 和 128 表示寄存器或内存空间的位数;这里的延时是指在依赖链中消耗的指令周期;吞吐量,以 1/4 为例,表示四个时钟周期后可以执行相同的指令。表中的值是最小值,当拷贝的数据不在缓存中或者没有对齐都会增加。memcpySSE( ) 和 MMXMemoryCopy( ) 选择 MOVQ 和 MOVNTQ 这一对 MMX 数据传输指令实现内存拷贝。以拷贝 16 个字节为例,函数部分代码如下:

```
movq mm1, 0[ESI]  //取数据
movq mm2, 8[ESI]
movntq 0[EDI], mm1 //存数据
movntq 8[EDI], mm2
```

如果代码在执行之前已经将数据拷贝到 CPU 缓存中,由于指令的吞吐量至少是 1/1,所以前两条指令不可能同时执行,后两条指令也不可能同时执行。在没有发生意外时,拷贝的数据对齐,则代码需要消耗的周期为 4 个,数据没有对齐则消耗 62 个。

前面已经论述了程序设计中不能选择 MOVAPS 指令,所以采用 MOVUPS 和 MOVNPTS 指令对。也以拷贝 16 字节为例,函数主体部分代码如下:

```
movups xmm1, 0[ESI]
movntps 0[EDI], xmm1
```

如果代码在执行之前已经将数据拷贝到 CPU 缓存中,在没有发生意外时,拷贝的数据对齐,则代码需要消耗的周期为 9 个,数据没有对齐则消耗 24 个。

比较分析的结果,SSE 指令对拷贝非对齐数据效率比 MMX 指令高。而且由于拷贝大块内存时需要调用循环语句实现,用 SSE 指令可以将循环数目减少一半,从而将减少循环语句所消耗的时钟周期。

3 基于 SSE 的内存拷贝函数设计

基于上面的分析,快速内存拷贝函数将采用 MOVUPS 和 MOVNTPS 这对 SSE 指令进行数据传输。该函数的程序流程图如图 1 所示。

具体步骤如下: ①程序首先读入源数据内存地址 Dst、目的内存地址 Src 和所读取数据的字节数 Nbytes; ②为了确定是否使用 SSE 指令,需要判断 Nbytes 是否大于 256,这样可以保证用 SSE 指令至少拷贝 128 个字节; ③判断 Dst 是否按照 32 字节对齐; ④如果没有对齐, 先从 Dst 拷贝偏移字节到 Src,同时更新 Dst 和 Src,更新后 Dst 按照 32 字节对齐; ⑤对齐后先计算缓存数据量和循环数(缓存数据量是 128 字节的整数倍,循环数≥1 而≤128),并进入缓冲循环,每个循环从 Src 取 128 个字节数据到缓存,直到取够缓存数据量; ⑥如果 Dst 已经对齐,则直接计算缓存数据量和循环数并进入缓冲循环; ⑦缓冲循环完后,重新读入缓冲循环数,进入读数据循环,每次用 SSE 指令从缓存取 128 个字节数据到 8 个 128 位寄存器中,再将这 128 个字节数据不经过缓存直接送到目的内存地址,直到缓冲循环数为 0; ⑧从 Src 读偏移字节到 Dst。

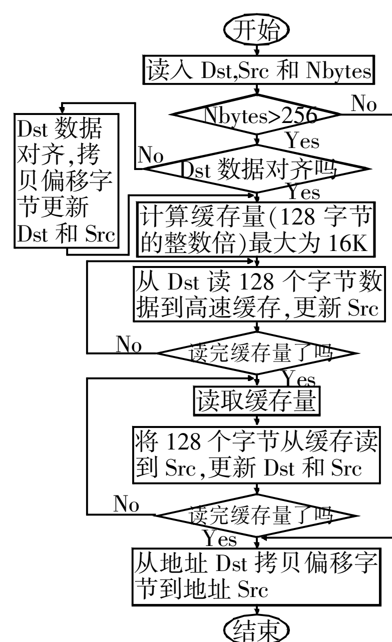

图 1 程序流程图

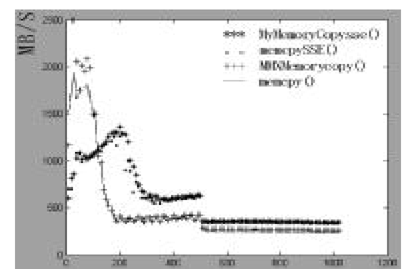

图 2 数据对齐情况

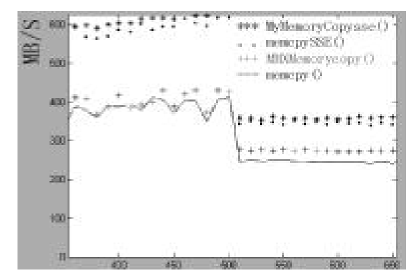

图 3 数据对齐的局部细节放大

4 实验结果与分析

下面设计的实验中,将 MyMemcopySSE( ) 与现存的拷贝函数的内存拷贝速度进行了比较,这些函数包括 MMXMemoryCopy( ), memcopySSE( ) 以及 Windows 的内存拷贝函数 memcpy( )。

实验数据如下: ①实验环境。Intel Pentium(r) IV Processor, 1781MHz CPU, 采用 Intel SSE™2 Technology; 2 ×128MB SDR 内存; 4kCPU 一级数据缓存, 16k 一级代码缓存, 256k 二级数据缓存; Windows Profesional 2000 操作系统。②实验方案。拷贝的内存大小从 10KB 开始,以 10KB 为步长,逐渐增加到 1 020KB;数据类型分别为对齐和没有对齐两种;使用编程语言 Visual C++中的嵌入式汇编。③实验得到的数据如图 2 ~图 5 所示,横坐标表示拷贝内存的大小(K 字节),纵坐标表示函数拷贝内存的速度(Mbps)。

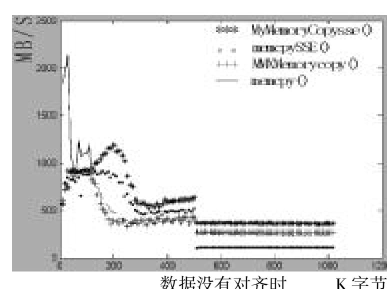

图 4 数据未对齐情况

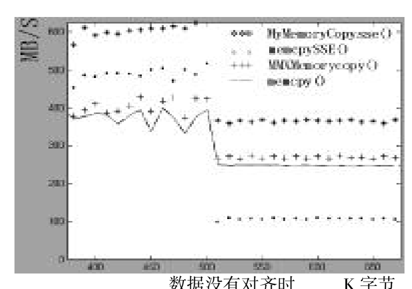

图 5 数据未对齐情况的局部细节放大

对实验得到的数据分析可知: ①当拷贝的内存量小于 500KB 时,所列内存拷贝函数的拷贝速度随拷贝 (下转第 120 页)