第38卷 第17期
Vol.38 No.17

计 算 机 工 程
Computer Engineering

2012年9月
September 2012

·开发研究与设计技术· 文章编号：1000—3428(2012)17—0276—04 文献标识码：A 中图分类号：TP391

基于C6000的数据存储处理编程优化方法

苑玮琦，王 斌

(沈阳工业大学视觉检测技术研究所，沈阳 110870)

摘 要： 针对TI TMS320C6000系列数字信号处理器容易忽视的CPU指令并行、软件流水特点和编译器内联函数、线性汇编及汇编语言的高速运行特点，给出常见的指令存储器相关性问题、循环冗余问题、嵌套循环的流水性能问题和程序存储块冲突问题的分析与优化解决方法。以定点点积算法进行参照实验，结果证明代码运行速度在进行相应C程序编程优化、线性汇编编程优化和手工汇编编程优化后最高分别可以提高85.9%、86.4%和93.1%。

关键词： 数字信号处理器；编程优化；指令并行；软件流水；线性汇编；数据存储

Programming Optimization Method in Data Storage Processing Based on C6000

YUAN Wei-qi, WANG Bin

(Institute of Vision Detection Technology, Shenyang University of Technology, Shenyang 110870, China)

【Abstract】 Using traditional programming methods on the TMS320C6000 Digital Signal Processor(DSP) may omit the advantages of parallel instructions, software pipelining and the high-speed performance of intrinsics, linear assembly language and assembly language, so the analysis and solving methods of common problems of instruction storage correlation, circulation abundance, the performance of nested loop and interleaved memory conflicting are provided. The reference experience with fixed-point dot product programs shows that the code running speed can be separately increased by 85.9%, 86.4% and 93.1% after optimized by optimization methods of C programming, linear assembly programming and hand assembly programming.

【Key words】 Digital Signal Processor(DSP); programming optimization; instruction parallel; software pipelining; linear assembly; data storage

DOI: 10.3969/j.issn.1000-3428.2012.17.074

1 概述

当前，常用的DSP编程实现方法是以高级语言C/C++为编程语言，通过编译器将程序编译成汇编代码，进而通过汇编器将代码汇编成机器指令，然后经过链接过程生成可执行文件。但由于通用的高级语言编程方式难以触及硬件底层，该方法实际上远未充分利用CPU内部高效的硬件特性。实际上，通常情况下编译器可以生成和手工汇编一样优秀的代码，前提是高级编程语言必须按照特定的方式来写，而这种特定的编程方式也即编程优化方法的提出与应用需要对CPU的运行有深入的理解[1]。TMS320C6000高性能DSP有2个并行的数据通路A和B，每个通路有4个功能单元(.L、.S、.M和.D)和一个包含16个(C64x为32个)32位寄存器的寄存器组。A、B 2个通路的寄存器通过交叉数据通路可以被不同通路的功能单元所访问。其具有能够实现指令流水、指令并行等特点，与TMS320C5000数字信号处理器相比，可以进行软件流水，在一个时钟周期内最多可并行对8条指令同时进行指令流水[2]。为了充分提高DSP程序代码的运行速度，本文从解决DSP编程过程中常见的数据存储与处理过程中影响代码性能的角度对传统的编程方法进行优化方面的研究补充，同时根据实际情况利用C/C++编译器内联函数、线性汇编语言等对程序进行部分编程优化。

2 数据处理和存储过程中的编程优化方法

依据编程语言不同可以将编程优化方法分为3层，分别是C/C++编程优化、线性汇编语言编程优化和汇编语言编程优化。除了使用长型指令处理短型数据、进行部分代码手工汇编优化之外，以下介绍几种C/C++和线性汇编编程中常见的数据处理和存储相关的问题和解决方法。

2.1 指令存储器相关性问题的优化解决

编译器为了提高代码效率，在编译时将尽可能安排指令并行执行，然而由于只有前后互相独立的指令才能并行执行，如果编译器未能确定2条指令是否相关，那么编译

基金项目： 国家自然科学基金资助项目(60972123)；沈阳市科技计划基金资助项目(F10-213-1-00)

作者简介： 苑玮琦(1960－)，男，教授、博士后、博士生导师，主研方向：应用机器视觉，嵌入式图像采集与处理；王 斌，硕士研究生

收稿日期： 2011-10-17 **修回日期：** 2011-12-20 **E-mail：** yuan60@126.com

器将认定其相关，并安排他们串行执行[3]。即，设指令 1 执行周期范围 $T1$，操作存储空间 $M1$，指令 2 执行周期范围 $T2$，操作存储空间 $M2$，针对程序内容，依照编译器对指令并行的处理机制，有：

若 $M1=M2, T1=T2$，则 $T1 \neq T2$(指令非并行执行)；

若 $M1 \neq M2, T1 \neq T2$，则 $T1=T2$(指令并行执行)。

在 C/C++编程过程中，通过使用 Restrict 关键字直接在程序中对指针或数组对象进行独立性限定，可以从程序上强制告知编译器指令访问的指针或数组对象所关联的存储空间不会发生重叠。或者通过编译器配置选项-mt 告知编译器可以执行指令并行。此外，通过编译器配置选项-pm 和-O3 进行程序级优化，编译器可以通过查看程序整体情况来主动判断是否有影响指令并行执行的指令相关存在。

2.2　循环问题的优化解决

通过执行 DSP 的软件流水，可以使复杂数字信号运算中大量循环的多次迭代并行执行，加速程序代码运行。在软件流水过程中，循环中的迭代部分连续以固定的时间间隔启动，而无需等待上一次迭代结束[4]。当使用了-O2 和-O3 编译器选项命令之后，在满足条件时，编译器对循环采用软件流水。

图 1 是一个软件流水循环示意图，其中循环内的各条指令分别为 A、B 和 C，并且该循环一个周期最多可以并行执行 3 条指令。完全并行执行指令的部分称作循环核部分，前面的部分称作循环填充部分，后面的部分称作循环释放部分。

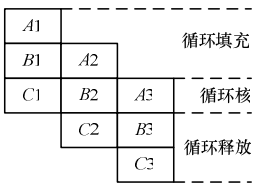

图 1　软件流水循环示意图

2.2.1　冗余循环的避免

由于进行软件流水需要至少得到一次循环核，因此存在一个软件流水过程的最小安全循环次数 N。当已知的最小循环次数 n 小于该最小安全循环次数 N 时，便需要增加一定冗余循环来保证整个软件流水过程的安全性。但编译器由于缺乏对循环次数信息的了解，往往在软件流水实现过程中增加了一定不必要的冗余循环。显然，增加冗余循环既增大了代码尺寸，又影响了程序性能。

在编译器未能判别循环次数是否安全时，将对 2 种情况分别处理：

当 $n<N$ 时，不执行软件流水；

当 $n \geqslant N$ 时，执行软件流水。

针对循环冗余问题，可使用编译器配置选项-ms0 或-ms1 控制代码尺寸，此时只有在编译器断定安全的情况下方采用软件流水；使用-pm 和-O3 选项，使编译器从整体上了解程序，以了解循环次数的确切信息；使用 MUST_ITERATE 伪指令来强制确定循环次数，可以保证循环执行一定的次数，不至于使编译器认为循环迭代次数可能为 0 而产生冗余循环。

2.2.2　循环的展开

由于编译器只对内部循环进行软件流水，因此在内层循环操作没有充分利用 DSP 结构资源的情况下，对周期较少的内循环进行展开将获得一个比较大的内循环，可以增加并行执行的指令数。

展开的方式主要有：编译器自动执行循环展开；在程序中使用 UNROLL 伪指令建议编译器循环展开；用户自己在 C/C++代码中手动展开。由于展开循环同样增大了代码尺寸，因此只有在软件流水性能受该循环影响极大的情况下才考虑对循环进行展开。

2.3　编译器高效内联函数的 C/C++优化编程

TMS320C6000 C/C++编译器中的内联函数直接调用了某些汇编语句，当使用内联函数时，编译器将对原函数进行检查以确保调用正确且返回值被正确使用。然后将函数调用替换为函数体，因而消除了参数压栈和执行调用等的开销[5]。这些汇编语句在 C/C++中实现起来很繁杂，除节省了调用开销外，也节省了程序代码本身运行时间。

但内联函数扩展会扩大代码尺寸，尤其是在一个被多次调用的函数内联扩展后，因此，函数内联最好只用在被调用次数少和函数尺寸不大的情况下。

2.4　线性汇编编程问题的优化解决

2.4.1　线性汇编程序

如果经过 C/C++编程优化后程序运行仍旧达不到系统要求，可以通过使用 CCS Profiler 代码剖析工具对 C/C++程序进行剖析，找出严重影响程序性能的函数和程序段，并对其进行线性汇编编程，以达到提高程序性能的目的。之所以称作线性汇编，就是因为其输入汇编优化器的程序按照正常逻辑顺序编排，不存在并行运行等特点[6]。

以定点点积程序为例，主循环的线性汇编程序如下：

```
loop    LDH  *x++,a
;a 指向数列 x
        LDH  *y++,b
;将 y 的值赋给 b
        MPY  a,b,prod
;prod=x*y
        ADD  prod,sum,sum
;乘积之和赋给 sum
        SUB        count,1,count
;递减计数器
[count] B          loop
;如果 count 不为 0 则重复循环
```

2.4.2　线性汇编程序存储块冲突问题的解决

多数 TMS320C6000 系列的 DSP 内部存储器采用交叉式存储结构(C6x1x 除外)，其中定点 DSP 一般内存的交叉存储使用 4 个存储块(Bank)，浮点 DSP 一般内存的交叉存储使用 8 个存储块。该种交叉存储结构设计可增大有效存储带宽，以便在单位时间内进行更多存储字的读取[7]。在

DSP 内存储器中，每个存储块都在每个指令周期只能有一次访问，如果针对同一存储块在一个周期有 2 次操作，则会出现存储块冲突问题，影响了指令的并行执行。

图 2 表示了定点 DSP 采用的 4 存储块(Banks)的交叉式存储器结构，其中的数字表示字节地址。一条字节加载指令(LDB)的加载地址如果是 0，则从 Bank1 中加载字节 0 中的内容。

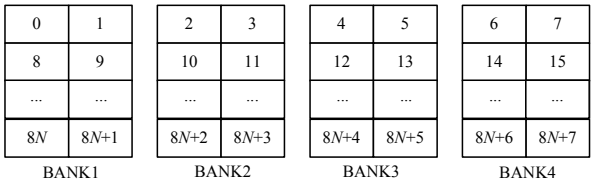

图 2 定点 DSP 采用的 4 存储块的交叉式存储器

欲对存储块冲突情况加以分析，需要了解存储对象的基址、偏移量和步长。存储对象的基址、偏移量决定其地址，在存储对象位置变化步长一般相同的情况下，由于整个存储结构为偶数个存储块，因此当 2 个指令操作的存储对象的地址之差为奇数时，不易发生存储块冲突，反之地址之差为偶数时，发生存储块冲突的概率则大幅增加[8]。根据这种特点，在编写线性汇编程序时，令 2 个指令的操作对象地址之差为奇数，可避免存储块冲突。即按图 3 中安排指令操作的存储对象地址 A、B，并且有 $A \geqslant 1$，$B=A+(2m+1)(m \geqslant 0)$。

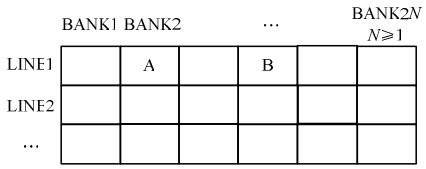

图 3 交叉式存储结构中非冲突存储对象安排

为在线性汇编程序中避免导致存储块冲突，使用.mptr 伪指令将数据对象的基址、偏移量等信息包含进一次处理数据的操作中。此时表示让汇编优化器自动检查 2 个存储器操作是否有块冲突。如果有块冲突，则汇编优化器就不会让这 2 次指令操作并行，从而避免了存储器操作和流水线操作的延迟现象的发生，反之则可以进行多个数据的并行指令处理操作。

2.4.3 线性汇编程序指令存储器相关性问题的解决

与 C/C++程序中可能出现的存储对象访问冲突问题相同，线性汇编程序中也可能出现 2 条指令存取相同存储器位置的情况，也即 2 条指令间可能存在存储器相关性。该情况的可能出现限制了线性汇编指令的并行编排，同时影响了软件流水编排。与存储器块冲突问题不同，存储对象地址访问是否冲突的不确定性除了降低指令编排的性能之外，还可能导致汇编优化器的误判，严重影响程序的正确性，因此，有必要对存储对象访问冲突情况进行明确。

针对线性汇编程序中存在的指令存储器相关性问题，在程序中可以使用.no_mdep 伪指令声明在该函数中不存在任何存储器相关，或者使用.mdep 伪指令可以通过给存储器引用添加注释名的方式来设置存储器相关，以使汇编优化器依照明确的存储器相关信息决定是否采用指令并行优化。

2.5 DSP 系统片外数据存取与处理的优化

对于 TMS320C6000 DSP 系统整体而言，由于 DSP 片上存储器的空间局限，片上存储器的数据往往来自于直接内存存取(DMA)预取自片外存储器的内容，因此，对片内外存储的调度将成为限制程序高效运行的关键。对 DMA 操作的合理调度不仅可以提高 CPU 存取数据的速率，同时也能够通过有效控制以避免预取数据量过大导致的系统运行速度过慢、假死等现象的发生。

通过实现 DMA 片外存储器数据预取与 CPU 数据处理重叠进行，可以有效解决以上问题。此时，可以使 DMA 与 CPU 由串行改为并行工作，提高系统性能。同时也能够使数据预取量与数据处理量维持平衡，避免数据预取过多而造成 CPU 负荷过大，保证系统稳定。DMA 和 CPU 两者并行时序如图 4 所示。$T1$ 至 $T4$ 分别为每次并行运行时间段，在前一个时间段内使用 DMA 预取适量的数据至片内存储器后，下一个时间段在进行后续预取的同时，CPU 也在处理当前片内指定位置的已预取数据。

图 4 DMA 与 CPU 并行执行示意图

针对片外存储器一般为包含多个存储块的 DRAM 的情况，使存储器访问命令操作与数据传输并行流水执行，可以增加数据传输周期，充分利用片外存储器带宽。

3 实验结果与分析

在运行主频为 600 MHz 的 TMS320DM642 DSP 芯片上进行实验，对采用各类编程优化方法处理过的定点点积函数进行代码量和运算速度剖析，所得结果如表 1 所示。

表 1 编程优化方法的实验结果对比

代码特点	代码大小/Byte	运行时间(周期)	代码大小增减比例/(%)	运行速度提高比例/(%)
原始 C 函数	168	405	0.0	0.0
使用长指令访问短数据+内联函数	312	354	85.7	12.6
C 函数使用-pm，-o3 编译器选项	248	57	47.6	85.9
线性汇编(不使用-o3)	56	216	−66.7	46.7
线性汇编(使用-o3)	244	55	45.2	86.4
线性汇编(使用.mptr，不使用-o3)	56	216	−66.7	46.7
线性汇编(使用.mptr 及-o3)	184	57	9.5	85.9
基于线性汇编优化后的手工汇编	64	170	−61.9	58.0
并行软件流水优化的手工汇编	192	28	14.3	93.1

对以上数据进行分析可知，对该原始 C 程序利用 32 Byte 数据处理指令优化 16 Byte 数据的方法结合使用内联函数_mpy()和_mpyh()，可使代码大小增加 85.7%，运行速度提高 12.6%，表明使用编译器内联函数将大幅增加代码量，同时能够一定程度上提高代码运行速度。对原始 C 函数使用-pm，-o3 编译器选项，以实现把所有源文件合并成一个模块进行程序级优化，同时开启最高级编译器自动优化，可使代码大小增加 47.6%，运行速度提高 85.9%，表明使用程序级优化和开启最高级编译器自动优化能显著增加代码大小，同时大幅提高运行速度。

针对定点点积线性汇编程序，对汇编优化器不使用 -o3 编译器选项，经汇编优化器自动汇编优化后可以获得较原始 C 程序的 46.7%的速度提高，同时代码大小减少 66.7%，符合使用线性汇编语言编程产生代码量小、运行速度高的特点。针对定点点积线性汇编程序，对汇编优化器使用-o3 编译器选项，开启最高级自动优化功能，可以获得 86.4%的代码速度提高，但代码大小较原始 C 程序高出 45.2%，表明-o3 编译器选项对汇编优化器同样具有增加代码大小、加快代码运行速度的作用。

由于未使用-o3 编译器选项的线性汇编代码尚未开启软件流水和循环展开等，因此此时使用.mptr 伪指令对原程序无影响，而在使用-o3 编译器选项的同时使用.mptr 伪指令标明存在的存储块冲突，代码大小较原始 C 程序增加 9.5%，运行速度仅提高 85.9%，表明使用.mptr 伪指令标明存储块冲突后减少了编译过程中的不确定性操作，在对代码运行速度影响不大的同时降低了代码量。

基于线性汇编的基本手工汇编函数代码大小较原始 C 程序减少 61.9%，代码运行速度提高 58.0%，体现了汇编语言编程比线性汇编语言编程更具有代码量小和速度快的优势。对基本的手工汇编函数进行并行软件流水优化处理，可以获得 93.1%的代码运行速度提高，而代码大小仅增加 14.3%，表明经过软件流水优化实现的汇编语言程序代码具有极高的运行速度优势和可接受的代码量大小。

实验测得不同方法下代码大小与运行时间及其变化情况如图 5、图 6 所示。

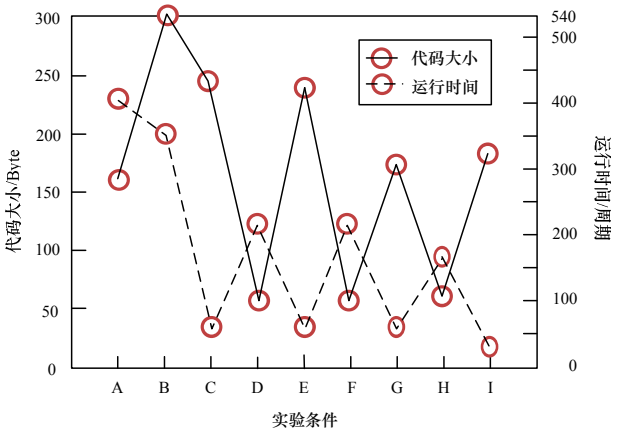

图 5　实验测得代码量与运行时间对照情况

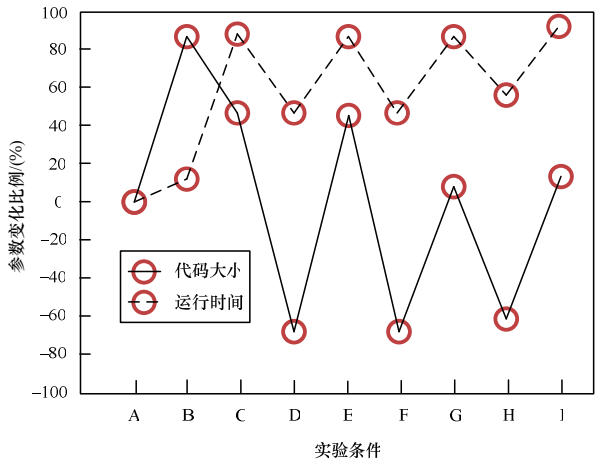

图 6　实验测得代码量参数变化情况

在图 5、图 6 中，实验条件 A~I 分别表示代码为原始 C 函数、使用长指令访问短数据与使用内联函数、C 函数使用-pm 与-o3 编译器选项、线性汇编程序(不使用-o3)、线性汇编程序(使用-o3)、线性汇编程序(使用.mptr，不使用-o3)、线性汇编程序(使用.mptr 及-o3)、基于线性汇编优化后的手工汇编程序以及并行软件流水优化的手工汇编程序。

综上所述，在使用 TMS320DM642 进行定点点积运算编程优化方法实验过程中，使用内联函数等 C/C++编程优化方式、线性汇编、手工优化汇编等优化编程方法在解决系统运行问题的基础上均可大幅提高代码运行速度。其中，以使用线性汇编函数和基本手工汇编函数所得程序代码量较小，以使用并行软件流水手工汇编优化的方式提高代码运行速度最多。

4　结束语

本文针对 TMS320C6000 系列 DSP 的硬件特点，对使用该系列 DSP 进行数据处理和存储过程中的常见编程问题给出了分析以及编程优化的针对性方法，涉及 C/C++编程优化、线性汇编编程优化和手工汇编编程优化 3 个编程层面。

总体上提高了程序代码在 TMS320C6000 系列 DSP 上的运行速度，使得嵌入式 DSP 系统的复杂算法能够充分利用 CPU 的指令并行和软件流水特点、编译器内联函数和线性汇编及汇编语言的高速运行特点。结合 DMA 外部存储操作与 CPU 数据处理的并行调度，可以提高系统高效稳定的性能。

在 TMS320DM642 DSP 上以定点点积运算进行参考实验表明，较原始 C 程序代码相比，使用 C 程序编程优化方法之后代码运行速度最高提高 85.9%，使用线性汇编语言编程优化方法以后代码运行速度最高提高 86.4%，使用基于线性汇编优化的手工汇编代码运行速度可最高提高 93.1%。

(下转第 283 页)