TMS320C6416的软件编程优化

■ 西安电子科技大学 曾志斌
■ 武警工程学院 周子琛 申振宁

TI公司开发的TMS320C6416是目前业界性能最强的芯片之一，目前，它的最高频率为1GHz，指令执行的并行度最高达到8，最大运行处理能力为8000MIPS。本文就基于TMS320C6416芯片的软件编程及其优化进行研究。

1 器件简介

嵌入式系统软件的优化与系统的资源息息相关。TMS320C6416内部资源主要包括三部分：CPU内核、外设和存储器。处理能力强大的CPU包括有2个数据通道A和B，每个通道有32个32位字长的寄存器组，每个通道包括4个功能单元(L，S，M，D)，每个功能单元负责完成一定的算术或者逻辑运算。A、B两通道的寄存器并不是完全共享，只能通过TMS320C6416提供的两个交换数据通道1X、2X，才能实现处理单元从不同通道的寄存器堆那里获取32位字长的操作数。外设包括为CPU访问外围设备提供了无缝接口的灵活的外部存储接口EMIFA和EMIFB，3个多信道带缓冲能力的串口McBSP，每个最多可支持128多信道数字信号处理，1个提供64数据通道访问的增强型EDMA。1个使得DSP很容易通过PCI接口无缝连接到1个具有PCI功能的外部主CPU上的PCI接口，1个16/32bits宽的异步并行接口HPI(它和PCI共用相同引脚)，3个定时器，还有专门针对Viterbi译码的VCM和Turbo译码的TCM等。它的内存空间大，其内置的超大容量RAM达到1MB，片内存放数据和参数使得访问速度加快。

2 系统编程

系统编程首先要有一个良好的开发环境，良好的开发环境有利于程序开发，从而能节省开发时间。这里选用的是CCS2.2，它是集编程、编译、连接、实时调试、跟踪及实时分析应用程序于一体的开发平台。

系统编程语言可以是纯C/C++语言、汇编语言或是混合编程。采用纯C/C++语言编程方便，调试简单，但效率不高；采用纯汇编语言编程，代码效率高，实时性强，但调试不方便，需要具有一定编程经验，开发周期较长。为此，笔者采用混合编程进行系统软件开发。

在DSP系统编程中，合理利用系统的资源，可以有效地提高系统的性能。根据TI的C6416器件特性知道，它支持8、16和32位数据，它的指令执行的并行度最高可达到8，但是它要求这8条指令相互之间没有依赖关系，并且数据存储地址单元须按字边界排列，因此，优化好坏有讲究。

虽然系统软件设计应着眼点于软件系统框架的制定，但是如果没有良好的算法和控制策略，程序的效率要大打折扣。针对嵌入式开发，对于系统资源的利用尤为必要，一般编程时需要注意下述事项。

(1) 数据格式的选择

TI C6416在处理时是根据数据类型及自身资源进行的，在编写C/C++代码时要小心。例如，数据能用int格式表达的不要选long格式，因为Ti C6416寄存器只有32位长，选long格式需要更多指令，而且会缩小功能单元选项。

在编写线性汇编代码时，为了充分利用系统资源，对于数据类型应切合实际，以便选用最好的指令。例如，读取输入数据为12位宽的数据后进行点乘，最好选用short型数据类型，这样可以利用线性汇编指令LDDW，一次就可以读取4个数据。在点乘时选用组合指令完成，它可以充分利用系统A、B通道每组32个寄存器。

(2) 内存依赖关系

为了最大化代码性能，C6000编译器尽可能将指令并行处理，但是要做到这点，就得确定指令间关系。如果一条指令需在另一条指令执行后才能执行，那么这两条指令就存在依赖关系。当编译器不能识别指令间是否独立时，就得告诉编译器它们之间的关系，可以通过关键字‘restrict’或优化选项命令来通知。

(3) 合理编写循环体

在DSP进行数据处理中，循环体是用得最多的，处

理量最大，也是最应该注意的，坏的编写风格只会浪费不必要的时间。一般应注意：在C代码内不要插入一个汇编声明，在循环体内不要包含调用，循环体不要有过多指令；对于多重循环，如果内层循环体较短，可以考虑打破该循环体，与外部循环结合。

(4)变量的声明

对于内存相对充裕的系统，函数调用时尽可能采用全局变量，这样可减少对变量的堆栈处理，同时也节省了寄存器的使用，利于优化。

(5)流水线冲突问题

如果多个流水线同时访问到相同的资源，就可能发生流水线冲突，有些冲突可以由CPU通过延迟寻址的方法自动缓解，而有的冲突是不能防止的，需要由程序重新安排指令或插入空操作来解决。在进行手工优化的时候，就要特别注意这个问题，大部分流水线冲突都是因为同时访问到某些寄存器，只要根据等待周期表加入相应的NOP指令就可以解决。

(6)多调用TI提供的固有函数和功能函数

TI基于C语言开发提供了许多很实际有效的固有函数，如进行一次short型数据的点乘，采用TI提供的int _dotp2 (int src1，int src2)函数，它一次完成两个数(src1，src2)低16位的乘积与高16位的乘积和。如果采用一般C语言，就需要更多语句，耗时也就更多。对于常用的FFT变换，采用TI提供的子函数一般比自己编的效率要高得多。

3 系统优化

CCS为系统的优化提供了多种选择：O_0寄存器级别、O_1局部级别、O_2函数级别、O_3文件级别。

从上面初看，显然直接选用O_3级最简捷，它一步到位。然而在实际中，选择时应根据系统实际情况而定。一般较大的系统软件不宜选用O_3级优化，因为这样一般容易造成系统死循环。在实际工作中，一般采取多点优化措施，它根据系统的实时性要求，对于关键处的函数选用O_2级，对于次要的可以选用O_1级、O_0级或不选，这样不会造成系统程序不执行或程序混乱的现象。

选用合理的程序优化设计，可以获得较理想的效果。下面用一个实例来说明编程中如何对程序进行优化，下面的函数是一个一般的关于求和运算函数。

```
void vecsum(short *sum, short *in1, short *in2, unsigned int N)
{
    int i;
    for (i = 0; i < N; i++)
    sum[i] = in1[i] + in2[i];
}
```

由于它没有指明存储单元的依赖关系，因此它每次需要先执行一次求和运算后再进行下一次读数，然后才进行下一次求和，依次类推。为了提高效率，对它进行如下改进。

```
void vecsum4(short *restrict sum, restrict short *in1,
restrict short *in2, unsigned int N)
{
int i;
#pragma MUST_ITERATE (10);
for (i = 0; i < (N/2); i++)
_amem4(&sum[i]) = _add2(_amem4(&in1[i]), _amem4(&in2[i]));
}
```

restrict(限制字)指明该函数读取的数据没有依赖关系，MUST_ITERATE(10)告诉编译器将要进行循环的最少次数，_amem4指出读取的数据是按字边界排列，它的和的存储也按字边界方式进行，_add2表示每次完成2个short型运算。采用以上优化设计，运算速率约为之前的5倍。

采用优先级设计也可以提高程序运行效率，仍就以上两个函数进行对比。第一个未采用优化设计的程序，采用函数级优化后的执行时间约为未采用函数级优化的五分之一；对于采用优化设计的程序二，它采用函数级优化后的执行时间约为未采用函数级优化的五分之一。同时可以发现优化设计十分必要，它在同一优化级别下要比没有采用优化设计的好得多。

一般进行优化设计时，要考虑该段程序的某些特点。比如上述改进后的程序，它获取的数据是无依赖关系的，这在实时读取外部输入的数据时不可取；它处理的数据数为偶数，如果处理的数据数量为奇数就得另外变形后处理；并且它知道程序循环次数大于10，这些信息提供给编译器，有助于编译器进行优化设计。

当系统采用C语言优化设计不能最大发挥DSP的潜能时，为了进一步提高程序的执行速率，一般对于运算量大的以循环体为主的核心程序多采用线性汇编编程。线性汇编语言的指令系统和汇编语言的指令系统完全相同，它相对于编写汇编语言的优势在于，编写线性汇编语言时不需要考虑指令的延时、寄存器的使用和功能单元的分配，完全可以按照高级语言的方式进行编写。通过自身的汇编优化器指令系统，实现最优化编程。它的代码效率高于高级语言，而且开发难度和开发周期比汇编语言要小得多。

下面给出一个求128点‘short’型数据平方和的线性汇编程序

```
    .global      _square_arry_sa
```

```
_square_arry_sa:  .cproc    in_arry_pointer, out_arry_pointer
          .reg    temp_a_hi:temp_a_lo, temp_b_hi:temp_b_lo
          .reg    temp_a_sqrt_hi:temp_a_sqrt_lo
          .reg    temp_b_sqrt_hi:temp_b_sqrt_lo
          .reg    temp_in_pointer, temp_out_pointer
          .reg    count
          ZERO    temp_a_hi
          ZERO    temp_a_lo
          ZERO    temp_a_sqrt_hi
          ZERO    temp_a_sqrt_lo
          ZERO    temp_b_sqrt_hi
          ZERO    temp_b_sqrt_lo
          MVK     32,    count
          ADD     in_arry_pointer, 8, temp_in_pointer
          ADD     out_arry_pointer, 8, temp_out_pointer
loop:     .trip   32
          LDDW    *in_arry_pointer++[2], temp_a_hi:
temp_a_lo
          LDDW    *temp_in_pointer++[2], temp_b_hi:
temp_b_lo
          DOTP2   temp_a_hi, temp_a_hi, temp_a_sqrt_hi
          DOTP2   temp_a_lo, temp_a_lo, temp_a_sqrt_lo
          DOTP2   temp_b_hi, temp_b_hi, temp_b_sqrt_hi
          DOTP2   temp_b_lo, temp_b_lo, temp_b_sqrt_lo
          STDW    temp_a_sqrt_hi:temp_a_sqrt_lo,
*out_arry_pointer++[2]
          STDW    temp_b_sqrt_hi:temp_b_sqrt_lo,
*temp_out_pointer++[2]
[count]   SUB     count, 1, count
[count]   B       loop
          .endproc
```

以上利用了DSP的内核指令，提高编程效率。例如，采取双字读取指令LDDW，一次可以读取4个short型数据，它有效地节省读取数据时间；采用内核指令DOTP2完成一次平方和运算，节省编码指令，它明确告诉编译器循环迭代次数为32次(它每次可以处理4组半字型数据，即short型数据)。优化时，该函数优化级别选函数级优化，优化后的该段程序比原来采用C优化实现同样功能的程序要快3倍。

一般线性汇编程序的可移植性要差一些，而一般采用C优化要比采用线性汇编优化的速率相对慢些。但是也有例外，在多条件判断场合，采用C优化比采用线性汇编优化有时要强。因此系统编程采用何种优化方法要结合系统要求来决定。

4 小 结

高速数字信号处理器的应用范围越来越广，特别是在移动通信领域中，软件无线电、智能天线等新技术的实现都需要强大的实时数字信号处理的支持。TMS320C64X系列DSP具有强大的数据处理能力，它配合芯片外围硬件设计，可以满足大部分高速率、高性能的要求。本文重点研究了TMS320C6416的优化策略，从工程和系统的角度总结出一套既能满足实时性又能保证开发时效性的实用优化编程方法。它适用于C64X系列。

参考文献

1 TMS320C6000 Programmer's Guide. TEXES INSTRUMENTS Literature Number: SPRU198G, 2002-08

2 任丽香，马淑芬，李芳慧编著. TMS320C6000系列DSPs的原理与应用. 北京：电子工业出版社，2000

3 TMS320C64x Technical Overview. TEXES INSTRUMENTS. Literature Number: SPRU395A, 2000-09

（收稿日期：2004-09-08）