DSP 体系结构下视觉监控优化方法研究

李 斌,李功燕,许世颐,田 原
LI Bin,LI Gong-yan,XU Shi-yi,TIAN Yuan

中国科学院 自动化研究所 综合信息系统中心,北京 100080
Research Center of Integrated Information System,Institute of Automation,CAS,Beijing 100080,China
E-mail:lixiebin2003@yahoo.com.cn

LI Bin,LI Gong-yan,XU Shi-yi,et al.Research on optimization methods for intelligent vision surveillance algorithm on DSP.Computer Engineering and Applications,2008,44(34):231-233.

Abstract: First,a methodology for migrating intelligent visual monitoring algorithms onto the TI DSP platform is discussed.Then,the methods for optimization,algorithms of optimization in C language and in assembly directive,and the technologies of optimization in memory configuration are illuminated in detail.System of surveillance for multi-objects has been carried into effect in TMS320DM642 DSP.The performance has stepped up after optimizing as a result.

Key words: intelligent vision surveillance;Digital Signal Processor(DSP);TMS320DM642;software optimization

摘 要:探讨了智能视觉监控算法在 TI DSP 平台上的移植方法。详细介绍了算法优化、CCS 编译器优化、C 代码优化、汇编指令优化及存储器配置优化等优化技术。通过具体的监控实例,在 TMS320DM642 平台上实现了多个运动目标的实时检测、跟踪算法,结果表明,对 DSP 进行优化之后其性能获得了指数级的增长。

关键词:智能视觉监控;数字信号处理技术(DSP);TMS320DM642 技术;软件优化
DOI:10.3778/j.issn.1002-8331.2008.34.070 **文章编号**:1002-8331(2008)34-0231-03 **文献标识码**:A **中图分类号**:TP301.6

1 引言

随着 DSP 技术的快速发展,其运算能力显著提高,能实时处理的信号带宽范围不断扩大,同时其性价比和开发手段也在不断完善。以 TI C6000 为代表的 DSP 处理器已经成为图像、视频等众多领域理想的硬件实现平台。但是由于 DSP 处理器自身的硬件体系结构特点,拥有多个运算单元、两级 Cache、超长指令结构(VLIW)[1-2],这使得它在执行程序时与 PC 机的 X86 处理器有很大差别,如果按照传统的 VC 编程思想对 DSP 进行程序设计,会不可避免地导致代码效率低下、执行速度缓慢、实时性极差等现实问题。因此,为了充分发挥 DSP 处理器的性能,有必要在对 VC 下开发的程序代码移植到 DSP 的开发环境 CCS 后,增加一个算法再优化环节,这样才能实现各种视频图像算法的实时处理。

从 TI C6000 DSP——TMS320DM642 的硬件体系结构特点出发,以智能视觉监控算法的实时处理为目标,针对视频图像的计算复杂度高、数据量庞大、数据相关性强及帧、场时间限制严格等难点问题,系统研究了 DSP 程序设计过程中的优化思路与方法,具体包括算法级优化、编译器优化、C 语言程序优化、汇编指令优化及存储器配置优化等。最后,通过具体的监控实例,在视频图像的预处理、背景更新、目标检测与跟踪等各个环节应用本文所提的软件优化方法,实现了运动目标的实时、准确跟踪。

2 视觉监控算法概述

采用常规的监控算法,选择了建立背景模型作为检测前景区域的方法,其计算复杂度适中,利于 DSP 的实现。背景更新采用了直接的帧间变化检测方法,即相对于静止背景或背景变化缓慢的运动检测,利用图像序列逐渐恢复、更新背景图像;然后利用背景减除并阈值化的方法检测出运动目标区域;最后通过 Kalman 滤波实现对运动目标的跟踪。具体的流程图如图 1 所示。

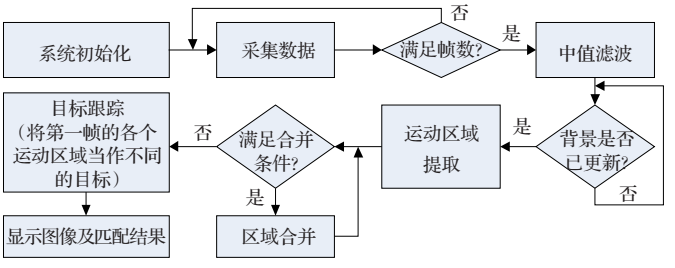

图 1 视觉监控算法流程图

2.1 背景更新

定义各个图像点的通道 $\eta(x,\mu,\Sigma)$为满足高斯分布的背景模型,其中均值为 μ,协方差矩阵为 $\boldsymbol{\Sigma}$;单通道单个点的高斯分布模型为 $\eta(x,\mu_t,\Sigma_t)$,其中下标 t 表示时间。设当前图像点当前通道的度量为 X_t,若 $\eta(X_t,\mu_t,\Sigma_t)\leqslant T_p$($T_p$ 为阈值),则该通

基金项目:中国科学院科技创新基金(No.CXJJ-09-MZ1)。
作者简介:李斌(1984-),男,博士研究生,主研领域:机器视觉,图像处理;李功燕(1979-),男,博士研究生,主研领域:嵌入式系统与视频处理。
收稿日期:2007-12-17 **修回日期**:2008-03-18

道该点被判定为前景点,否则为背景点(又称 X_t 与 $\eta(x,\mu_t,\sum_t)$ 相匹配)。记 $d_t=X_t-\mu_t$,则可以根据 $\boldsymbol{d}_t^{\mathrm{T}}\boldsymbol{\Sigma}_t^{-1}\boldsymbol{d}_t$ 的值设置前景检测阈值[3]。

单通道各图像点高斯分布背景模型的更新可以通过引入参数 α 来表示更新的快慢,则表示为:

$$\mu_{t+1}=(1-\alpha)\mu_t+\alpha\boldsymbol{d}_t$$

$$\boldsymbol{\Sigma}_t=(1-\alpha)\boldsymbol{\Sigma}_t+\alpha\boldsymbol{d}_t\boldsymbol{d}_t^{\mathrm{T}}$$

同理对每个图像点多个通道则采用多个高斯模型的混合表示,设用来描述多个通道的高斯分布共有 N 个,分别记为 $\eta(x,\mu_{t,i},\sum_{t,i})$,$i=1,2,\cdots,N$。各个高斯分布分别具有不同的权值 $w_{i,j}(\sum w_{i,j}=1)$和优先级 $p_t=w_{i,j}|\sum_{t,j}|^{-1/2}$,它们总是按照优先级从高到低的次序排序。取定适当的背景权值部分和阈值,在取定阈值之内的前若干个分布为背景分布,其他则是前景分布。在检测前景点时,按照优先级次序将 X_t 与各高斯分布逐一匹配,若没有表示背景分布的高斯分布与 X_t 匹配,则判定该点为前景点,否则为背景点。

背景模型的更新要满足两点:(1)背景模型对背景变化的响应速度要足够快;(2)背景模型对运动目标要有较强的抗干扰能力。

2.2 目标检测

图像序列为 $I(x,y,i)$,其中 x、y 代表空间坐标,i 代表帧数,$M(x,y)$代表静止分段中点的对应帧号;$B(x,y)$为背景图像;其中 $B(x,y)=I(x,y,M(x,y))$;

在恢复场景背景之后,可以得到每一时刻的背景帧差图 $ID(x,y,i)$;

$$ID(x,y,i)=\begin{cases}d,d>T\\0,d<T\end{cases},d=|I(x,y,i)-B(x,y)|$$

前景目标分割的依据通常是目标的空间连续性和颜色通道的一致性。由于后者很多时候不可靠,所以本系统选择了根据空间连续性来分割目标的连通区域。连通检测算子采用 8-连通检测。

3 视频监控算法在 DSP 上的优化方案

基于 TI C6000 DSP 软件开发流程可分为 3 个阶段:

第一阶段:直接用 C 语言编程实现需要的功能,并验证其正确性。然后使用开发环境集成的 profiling 工具,或测试程序对代码进行测试后找出低效率代码段。

第二阶段:结合硬件的特点,利用 C 语言编译器提供的优化选项对代码进行优化。

第三阶段:用线性汇编代码重新编写低效率段。

软件优化的两个目标:更快的执行速度和更小的代码长度,两者在很多情况下是相互矛盾的,根据需要达到折衷。具体的优化参考指标为:(1)需要读取/存储操作的数量;(2)需要乘法操作的数量;(3)逻辑操作的数量;(4)以短整(short)型、字节(byte)型数据计算的大小。优化的内容包括以下。

3.1 缓存结构优化

所设计的视频监控系统把 256 KB 大小的二级缓存 *L2* 配置为 Cache 和 SRAM,*L2* 的一半空间作为片上内存,另一半作为 Cache 使用。由于存储器空间的限制,必然涉及到数据在片内和片外存储器之间的搬移操作。数据搬移操作由 DMA642 的 EDMA 单元完成[4]。EDMA 可在没有 CPU 参与的情况下,完成片内 *L2* 存储器与片外存储器传输。在程序中使用 EDMA 搬运数据可直接调用一系列 DAT 函数,如下所示:

```
DAT_open(DAT_CHAANY,DAT_PRI_LOW,DAT_OPEN_2D);//开始 DMA 操作;
id=DAT_copy(fgndcr+i*180,disFrameBuf->frame.iFrm.cr1+i*(disLinePitch>>1),
        180);//源数据从片内缓冲区搬运到片外存储器;
DAT_wait(id);//检测 DMA 复制是否已经完毕;
DAT_clost();//释放资源;
```

DMA642 只在 *L1* 和 *L2* 存储器之间执行一系列检查[5],要保证 CPU 和 EDMA 读写数据不会出现覆盖问题,则可以让 CPU 只在 Cache 内读写数据,EDMA 只在存储器中读写数据。在使用 EDMA 之前先用 CACHE_clean()函数清空 CACHE,把要读取的数据再装入 CACHE。

3.2 减少指令及数据相关性

C 编译器会尽可能地将指令并行执行以便最大限度地优化代码,但如果指令之间具有存储器相关性,则它们在操作时必须按照一定的先后次序,不能并行执行。通常 C 编译器要考虑各种情况而很难识别代码数据之间是否相关,为使本来不具有存储相关的指令数据并行执行,则要让编译器知道指令数据间的不相关。

(1)使用关键字 restrict 修饰对象

restrict 用于修饰一个指针、引用或数组,表示在该指针或数组起作用的范围内,没有其他指针或数组能访问到该数组所代表的存储区域,说明两个目标指针是相互独立的[6]。一旦编译器得到此信息,它就会流水存储数据。例如,如下所示代码片段为本文设计的监控系统中某一段程序及其汇编代码,可以看出汇编代码的并行性很高。

```
void difference(unsigned char *restrict pfgnd,unsigned char *restrict pbgnd,unsigned char *restrict pprf,unsigned char *restrict pdiff,unsigned char *diff_img)
{
    unsigned int i,temp;
    #pragma MUST_ITERATE(LENTH4);
    for(i=0;i<(LENTH4);i++){
      temp=_xpnd4(_cmpeq4(_amem4_const(pdiff),0)); //更新图像与 0 比较;
      _amem4(pdiff)=(track_config.framecount4&temp)|(_amem4_const(pdiff)&(~temp));
      _amem4(pbgnd)=(_amem4_const(pbgnd)&(~temp))|(_amem4_const(pfgnd)&temp);
    }
}
```

相应的汇编程序段为:

```
AND     .S2     B18,B5,B18              ;|89| <0,18>
||      AND     .L2     B8,B7,B5        ;|93| <0,18>
||      AND     .L1X    B5,A3,A7        ;|90| <0,18>
||      XPND4   .M2X    A7,B4           ;|93| <0,18>
||      LDW     .D1T1   *+A8(4),A17     ;|90| <1,7>
||      LDW     .D2T2   *B9++(8),B18    ;|88| <1,7>
||      MVKH    .S1     _track_config+20,A6 ;|89| <1,7>
        OR      .L2X    B18,A17,B18     ;|89| <0,19>
||      OR      .S1X    B4,A7,A4        ;|90| <0,19>
||      AND     .L1     A9,A4,A9        ;|93| <0,19>
||      CMPEQ4  .S2     B7,B17,B21      ;|88| <1,8>
```

```
||      LDW     .D2T2   *B21,B8          ;|93| <1,8>
||      LDW     .D1T1   *A6,A6           ;|89| <1,8>
```

(2)使用关键字 const 来修饰对象

const 修饰的对象表示该变量或其所指存储区的数值保持不变,利用其他变量或指针的写入操作不会针对该变量,所以一定程度上消除了存储器之间的相关性。

(3)循环展开

循环展开是在程序里把小循环的迭代展开,使可能并行的指令数增加,从而改变软件流水线编排。为了提高性能应当尽可能地是内循环展开,内循环展开后就可以对原来的外循环进行软件流水,展开循环的方法有使用 Pragma UNROLL 和手工展开。

(4)DATA_ALIGN 的使用

DATA_ALIGN 为数据对齐指令,对于 DM642 支持的内存空间对不对齐的双字(64 位)一次性读取,但是在同一时钟周期内,如果内存地址对齐则可以同时进行两组双字读取。因此在定义数组的时候使用 DATA_ALIGN 可以很大程度上增加程序的并行性。

如:#pragma DATA_ALIGN(fgndy,4)

unsigned char fgndy[LENTH];

3.3 数据打包处理

数据打包处理是一种用单一指令对多个独立的数据(SIMD)执行相同操作的处理方法。C64 系列支持两种基本的数据打包处理优化方法:向量化和宏操作。向量化是同时对多个数据元素执行完全相同的单一操作,即充分利用 C64X 宽的存储空间访问通路,换用字或双字的读取和存储操作来进行字或半字数据访问;宏操作与矢量化很相似,主要差别是它在相邻元素间有数学运算。

数据打包处理机制的核心是打包数据类型——用来将多片数据元素保存到一个寄存器内。打包数据类型是 C64X 打包数据处理的基石,每一个打包数据类型将多个数据单元打包到一个 32 位的通用寄存器中。例如,matrix_add_2x2( )函数,使用了数据打包处理,其效率提高了一倍以上,并且精简了代码的长度。具体修改见下面函数,其功能可以用公式 $C_{2\times2}=A_{2\times2}+B_{2\times2}$ 来描述:

```
inline void matrix_add_2×2(short a[restrict][2],short b[restrict][2],
short c[restrict][2]){
        //c[0][0]  = a[0][0] + b[0][0];//数据打包前的代码;
        //c[0][1]  = a[0][1] + b[0][1];
        //c[1][0]  = a[1][0] + b[1][0];
        //c[1][1]  = a[1][1] + b[1][1];
    _mem4(c[0]) =_sadd2(_mem4_const(a[0]),_mem4_const(b[0]));
//数据打包后的代码;
    _mem4(c[1]) =_sadd2(_mem4_const(a[1]),_mem4_const(b[1]));
    }
```

在程序中使用 pragma directive:MUST_ITERATE 等指令向编译器提供访问的数据是字(或双字)边界且循环次数为偶数的信息,编译器将自动采用数据打包处理技术进行编译。

3.4 intrinsics 函数的使用

TI C6000 编译器可以识别许多 intrinsics 函数。intrinsics 函数直接调用某些汇编语句,而这些汇编语句可以用来替代复杂的 C 程序,可以使其执行效率显著提高。intrinsics 函数在其函数名前加一个"_"符号来特别标注,它的使用方法与函数调用类似,可以使用 C 程序中的变量。例如:

(1)使用 intrinsics 函数的程序

_cmpeq4(_amem4_const(pdiff),0);//比较 pdiff 指针所指对象与 0 的大小;

(2)汇编程序

```
ZERO    .D2     B17
CMPEQ4  .S2     *+B9(4),B17,B21
```

通过对比,本来需要多个指令周期的 C 程序,用 intrinsics 函数编写的程序结构简单且仅需一条到两条指令周期,具有极高的效率。因此,应尽量使用 intrinsics 函数提高程序的执行效率。

3.5 线性汇编和汇编的使用

当编译器不能充分利用 TI C6000 构架的潜能时,可以通过把循环写成线性汇编代码来获得更好的性能。针对程序中调用频繁,耗时较多的部分用线性汇编和汇编实现。在使用汇编语言编写 DSP 程序时,需要考虑的问题有:写并行指令,CPU 功能单元的最大化使用,填充延迟间隙,指令乱序技术,程序中的某些指令的执行没有严格的顺序要求,可以调整这些指令的位置,穿插于其它指令之间,以提高并行性。

3.6 编译器优化选项

TI C6000 的 C/C++编译器支持标准 C 语言,并做了补充和扩展,程序的优化是一个交互的过程,需要开发者利用编译器反馈回来的信息不断修改源程序,编译器再根据所编程序中提供的信息和指定的编译选项来进行优化,表 1 为编译器所提供的一些重要的编译选项机器功能例表[6]。

表 1　一些重要的编译选项与功能

选项	说明
-o3	使程序得到最高程度的优化,编译器执行文件级的优化
-k	编译器将保留编译过程中的.asm 文件,编译者可以根据需要查看.asm 中的信息
-pm	在程序级将代码优化,容许编译器对整个项目的所有源程序联合观测
-q	除了源文件名和报错之外,不输出所有工具在处理过程中产生的标语和过程信息
-mt	表示在程序中没有使用存储器混叠技术,使得编译器可以更积极的优化。使用-mt 选项,它允许编译器假定可以消除存储器相关性路径,可以消除访问这些变量之间的存储相关性
-oin	使用-o3 的时候,编译器会将函数自动展开,这样会增加代码的尺寸,-oin 选项将限制代码的尺寸,即对函数展开的程度予以限制,但保留-o3 选项的其他优化功能

编译者根据需要在 CCS 中 Build options 中自行设置编译选项[7-8],如图 2 所示。其中-pm 联合使用-o3 来指定程序级优化。在该过程中,所有源代码在编译过程中将编译成一个中间文件以给编译器一个完整的可视程序。对于确定指针位置传到函数时有着突出的优点。一旦编译器确定两个指针不访问同一个存储空间,就可以在软件流水中取得很大的改进。因为编译器访问整个程序,它将进行几个附加的、在文件级优化中很少见的优化。

在一个函数中,如果一个特定变量总是有着相同的值,则编译器用这个值取代这个变量,代替该变量传递值;如果函数的返回值从未用到,则编译器删除该返回值;如果函数无论是直接还是间接都不会被调用,则编译器消除这个函数。同时,用-pm 选项可以为一个循环安排更好的进度。如果循环重复的数目由传给函数的值确定,并且编译器可以确定被调函数返回的值,则编译器将会给出较好进度的循环最小数值。

(下转 238 页)