文章编号 1674-2915(2009)05-0452-08

DSP 图像处理的程序优化

黄德天,陈建华
(1. 中国科学院 长春光学精密机械与物理研究所,吉林 长春 130033;
2. 中国科学院 研究生院,北京 100039)

摘要:中值滤波在滤除噪声的同时,能够很好地保护图像边缘,是最常用的图像预处理方法之一。本文针对中值滤波算法排序量多、处理速度慢的缺点,提出了对中值滤波进行优化的方法。介绍了中值滤波的原理,使用基于 TMS320C6000 系列 DSP 的 C 程序优化方法对中值滤波代码进行优化,包括使用编译器选项、内联函数、软件流水和循环展开等。介绍了算法优化的基本步骤和具体方法,在 CCS2 软件平台下选择 C64DSP 做仿真,实验结果表明,优化后代码的运行时间从优化前的 8.37 ms 减少到 2.91 ms,提高了执行效率,能够满足当今实时图像处理的需要。

关 键 词:中值滤波;程序优化;内联函数;软件流水
中图分类号:TP391.4 **文献标识码**:A

Code optimization of DSP image processing

HUANG De-tian, CHEN Jian-hua

(1. *Changchun Institute of Optics, Fine Mechanics and Physics, Chinese Academy of Sciences, Changchun* 130033, *China*;
2. *Graduate University of Chinese Academy of Sciences, Beijing* 100039, *China*)

Abstract: The median filter is one of the image preprocessing methods most in use, for it can not only filter the noise, but can also protect the image edge. For a lot sorting processing and low processing speed existing in the algorithm, the optimized algorithms for the median filter were presented in the paper. The principle of median filter was introduced and the C code optimization based on the TMS320C6000 Series DSP, including compiler options, intrinsics, software pipelining, loop-opened, etc., were used to optimize the code of the median filter. Experimental results indicate that the running time of the optimized code has decreased from 8.37 ms to 2.91 ms as compared with that of the original coding method. It is concluded that the proposed algorithms can enhance the efficiency of the code execution, reduce the processing time of median filter and can meet the requirement of real-time image processing further.

Key words: median filter; code optimization; intrinsics; software-pipelining

收稿日期:2009-06-15;**修订日期**:2009-08-19

1　引　言

图像信号在产生、传输和记录过程中,经常受到各种噪声的干扰,严重地影响了图像的视觉效果。同时,这些噪声的干扰还会使得目标与背景的对比度较小、信噪比较低,从而给后续的图像处理工作(如:边缘检测、图像分割、特征提取、模式识别等)带来困难,所以通常对于含有随机噪声的图像信号需先考虑进行滤波预处理,这样既可以消除噪声影响又不会使图像的边缘轮廓和线条细节变模糊[1]。目前常使用的噪声滤波器,从整体上可分为线性和非线性滤波两种。在数字信号处理和数字图像处理的早期研究中,线性滤波器是主要处理手段,它对加性高斯噪声有较好的平滑作用。然而当信号中含有非叠加性噪声时,线性滤波结果很难令人满意。非线性滤波器在细化脉冲噪声和保护边缘性能方面具有很好的效果,中值滤波器是应用最为广泛的一种非线性滤波器。

常规中值滤波算法一般都采用排序的思想[1,2],这种算法存在循环迭代结构和计算次数不确定的缺陷,同时需要处理的数据量大,较难以满足实时处理的要求。基于以上考虑,本文提出采用基于 TMS320C6000 系列 DSP 的 C 代码优化方法对中值滤波代码进行优化。以中值滤波算法的实现为例,简要介绍基于 TMS320C6000 系列 DSP 的 C 代码优化方法,包括使用编译器选项、内联函数、字访问短型数据、软件流水和循环展开等,并给出了具体的优化过程。优化后,代码的执行性能提高,可以满足实时图像处理的要求。

2　中值滤波原理

中值滤波是一种邻域运算,该算法把邻域中包含的图像像素按灰度级升序或降序排列起来,取灰度值居中的像素灰度为该邻域中点像素的灰度。显然,中值滤波是一种非线性的处理方法,其排序可以使图像邻域内的高频成分因其突变特性而位于中值两旁。这样,中值以高概率事件为原图像信息得到保留[3]。

在中值滤波实际应用中,图像邻域通常用一个有奇数个点的滑动窗口确定。以一维为例,取窗口长度为 m(m 为奇数),对一维序列 $\{f_1, f_2, \cdots, f_m\}$ 进行中值滤波,即从输入序列中相继抽出 m 个数 $\{f_{i-c}, \cdots, f_{i-1}, f_i, f_{i+1}, \cdots, f_{i+c}\}$,其中 f_i 为窗口中心点值,$c = (m+1)/2$,将这 m 个点值按其数值大小排序,取其第 $(m+1)/2$ 个数作为中值滤波的输出,数学表达式为:

$$y_i = \text{median}\{f_{i-c}, \cdots, f_{i-1}, f_i, f_{i+1}, \cdots, f_{i+c}\},$$
$$i \in 2,\ c = \frac{m+1}{2} \qquad (1)$$

如一窗口长度为 5,像素灰度分别为 {20,10,25,15,100},它的中值为 20,灰度值为 100 的那个像素为随机脉冲噪声,经过中值滤波后被滤除。但是,一维滤波只考虑了图像在垂直或水平方向的相关性,滤波效果不是很明显,所以在实际应用中通常采用二维中值滤波。

一维中值滤波的概念很容易推广到二维,其数学表达式为:

$$y_{ij} = \underset{A}{\text{med}}\{f_{ij}\}, \qquad (2)$$

其中 A 为窗口,$\{f_{ij}\}$ 为二维数据序列。

二维中值滤波采用一个含有奇数个点的 $n \times n$ 滑动窗口(n 为滑动窗口的行数和列数),从左至右,从上至下逐行移动,将窗口正中像素的灰度值用窗口内各点的像素灰度值的中值代替[3~5]。例如:假设图 1(a)为一幅图像,经过一个中值滤波窗口处理后,得到图 1(b)的结果。

$$\begin{bmatrix} 1 & 2 & 1 & 4 & 3 \\ 1 & 2 & 2 & 3 & 4 \\ 5 & 9 & 8 & 4 & 5 \\ 5 & 8 & 7 & 9 & 5 \\ 3 & 5 & 3 & 6 & 3 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 1 & 4 & 3 \\ 1 & 2 & 3 & 4 & 4 \\ 5 & 5 & 7 & 5 & 5 \\ 4 & 5 & 7 & 5 & 5 \\ 3 & 5 & 3 & 6 & 3 \end{bmatrix}$$

(a)　　(b)

图 1　3×3 中值滤波

Fig. 1　3×3 median filter

二维中值滤波器的窗口尺寸(n 的大小)的选择对滤波效果的影响很大。一般而言,使用大的窗口,可以有效地抑制噪声,而使用小的窗口可以保留图像重要的结构特征。对于不同的图像内容和要求,应选用不同的窗口尺寸,一种可行的办法是窗口尺寸由小到大逐步增加点数,直到效果满

意为止。

3 算法优化简介

3.1 算法优化的基本步骤

TMS320C6000(简称 C6000)环境支持的 C 语言与标准 C 语言并无本质上的差别,标准 C 语言规定的一切内容在 C6000 环境下完全适用,并且 C6000 环境下的 C 语言还对标准 C 语言进行了一定的补充和拓展。当 C6000 硬件平台选用标准 C 语言编程时,可利用 C6000 优化方法优化 C 代码。这些方法主要包括使用编译器选项、内联函数和代码转换(字访问短型数据、软件流水和循环展开等)[6]。

代码优化流程如图 2 所示,整个工作流程分为 3 个阶段:

第 1 阶段:直接根据需要用 C 语言实现 DSP 功能,测试代码的正确性;然后移植到 C6000 平台,利用 C6000 开发环境 Profile 测试程序的运行时间;若不满足要求,则进入下一阶段。

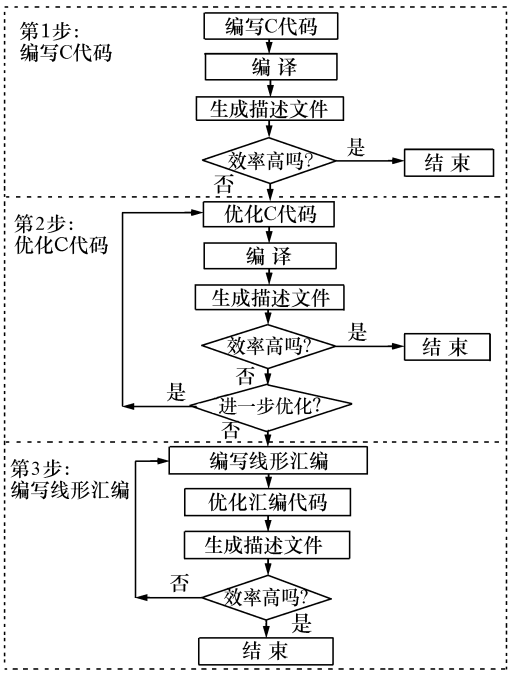

图 2 代码优化流程[7,8]

Fig. 2 Flow of code optimization

第 2 阶段:利用 C6000 提供的优化方式和其它各种优化技巧,如使用不同的编译器选项使用软件流水、循环展开、字访问短型数据等,优化 C 语言代码。如果还不能满足要求,则进入第 3 阶段。

第 3 阶段:将 C 语言代码中耗时最长的部分抽取出来,用线性汇编语言重写,用汇编优化器进行优化,使用 Profile 确定这段代码是否需要进一步优化[7]。

3.2 C6000 优化方法介绍

3.2.1 使用编译器选项

TI 所提供的集成开发环境 CCS 提供了灵活而强大的编译功能,通过合理的选择编译选项,能够使 C/C++ 代码的性能大幅度的提升,一般可达到汇编代码效率的 70% 到 80%,甚至 90% 以上。例如,-O3 选项的使用,该选项对文件级的代码进行最强的优化,可以使代码的并行性达到最高,是一个十分重要的选项[7]。表 1 为部分编译器选项列表。

表 1 最常用编译器选项[9]

Tab. 1 Compiler options most in use

选项	说明
-ox	文件级优化选项,与 -pm 合用,可以进行程序级优化,x 是数字 0、1、2 或 3。
-pm	使语法分析器在启动优化器和代码产生器之前,把所有的 C 文件合成一个文件来处理。这样可以对整个程序进行优化,使优化效率更高。
-mt	使能编译器假设程序中没有数据存储混淆,可进一步优化代码。
-mu	禁止软件流水。
-ms	确保不产生冗余循环,减小代码尺寸。
-mx	使能软件流水循环重试,该选项基于循环次数对循环试用多个方案,以便选择最佳方案。

当编译器被激活时,将完成图 3 所示的过程。C/C++ 语言源代码首先通过一个完成预处理的解析器,生成一个中间文件(.if)作为优化器的输入。优化器生成一个优化文件(.opt),这个文件作为完成进一步优化的代码生成器的输入,最终生成汇编文件(.asm)。当选择编译选项时,-O2 和 -O3 将尽可能地优化代码。但是,在使用该选

项优化循环、组织流水线时可能会出现错误,因此需要对优化后的结果进行正确性检验。

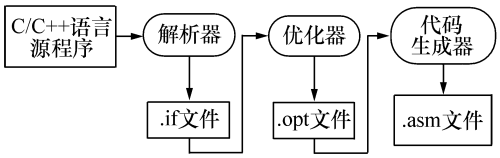

图 3　编译器的工作流程[10]

Fig. 3　Flow of compiler work

3.2.2　内联函数(intrinsics)

TMS320C6000 提供了很多内联函数,它们直接映射为内嵌 C6000 汇编指令的特殊函数,这样可迅速优化 C 语言代码。C 编译器以内联函数的形式支持所有 C 语言代码不易表达的指令。内联函数用下划线"_"开头,在使用上和普通函数类似,但执行效率和汇编指令完全一样,因此是一种既方便又高效的函数。表 2 中列出了一些较常用 Intrinsics 函数。

表 2　一些常用的内联函数

Tab. 2　Some Intrinsics most in use

Intrinsics	
_add2	_sadd
_clr	_set
_mpy	_smpy
_mpyh	_smpyh
_mpylh	_sshl
_mpyhl	_ssub
	_sub2

表 3　几种编程方式的比较

Tab. 3　Comparison of several forms of coding

C 代码	$y = a * b$;	代码效率低
嵌入汇编 asm 汇编代码	asm ("MPY A_0, A_1, A_2"); MPY A_0, A_1, A_2, ; a, b, y;	容易破坏 C 环境 编程工作量大
使用 Intrinsics	y = _mpy (a, b);	高效率

下面分别采用几种不同的编程方式来实现一个简单的乘法,然后做一个简单的比较,如表 3 所示。

Intrinsics 的主要特点:函数参数使用 C 变量名(不是寄存器),与 C 环境兼容;不增加 C 的编程工作量;代码效率与汇编相同。

3.2.3　字访问短形数据

如果要做 16 位 short 类型数据的点积运算,由于 C6000 的内部数据总线和寄存器都是 32 位的,因此可以采用字访问短形数据的方法进行优化。具体做法是:在做点积运算时,每次用 1 个 LDW 指令取 2 个 16 位数据,并进一步用 C6000 的 2 个 16 位乘法器在 1 个周期内并行完成 2 个 16 位乘法。这样,就有可能进一步提高点积运算速度。这些功能只要用指令 LDW 和 MPY/MPYH 就可以实现,或者也可以采用上面所介绍的 C6000 的所提供的特殊内联函数(intrinsics) _mpy 和_mpyh 来解决。在采用了字长优化等特殊的优化手段后,C 代码的效率可以达到 90% 以上。

3.2.4　软件流水

软件流水是用来安排循环指令,使这个循环的多次迭代并行执行的一种技术。在编译时使用 -O2 和 -O3 选项,编译器可对循环代码实现软件流水。图 4 是一个循环代码的软件流水示意图。图中 A、B、C、D 和 E 分别表示各种迭代,其后的数字表示各次迭代的第几条指令,同一行中的指令是同一周期内并行执行的指令。显然,同一周期内可最多执行多次迭代的不同指令(阴影部分)。图中阴影部分称为循环内核,核中5次迭

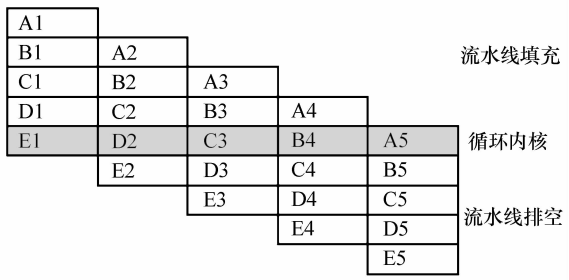

图 4　软件流水循环[11]

Fig. 4　Software-pipelined loop

代的不同指令并行执行。循环内核前面执行的过程称为流水线填充,后面执行的过程称为流水线排空。

通常,算法中存在大量的循环操作,如果能充分运用软件流水线方式,一旦软件流水建立,即8条或多条指令并行,则算法效率就相当高了。另外,有时为建立软件流水,表面上多做的工作(循环展开)是完全值得的。

需要注意的是,在一系列嵌套循环中,最里面的循环是唯一可以进行软件流水的循环。并且,有以下几种特殊情况:

(1)尽管软件流水循环可包含内联函数,但不能包含函数调用;

(2)在循环中不能有条件终止指令;

(3)如果在循环体中修改循环计数,这个循环也不能进行软件流水;

(4)代码尺寸太大,大于C6000中的32个寄存器时,这个代码不能进行软件流水,需要简化循环或将循环拆成几个小循环。

3.2.5 循环展开

改进性能的另一种方法就是展开循环,这种优化方法可增加并行执行的指令数。当单次迭代操作没有充分利用C6000结构的所有资源时,可使用循环展开提高性能。编译器仅对内部循环执行软件流水,因此为提高代码的执行性能,可创造一个比较大的内循环。创造大的内循环的一个方法就是完全展开执行周期很小的内循环。另外,对于一个结构简单的循环,可以通过直观判断,确定是否应该展开这个循环,这是因为展开循环会增加代码尺寸,并且流水线填充和流水线排空的总开销可能很大,甚至影响代码的执行性能。如果确定这个循环的展开能够提高代码的执行性能,那么在C代码中完全展开内循环,这样外循环就可进行软件流水,且填充和排空软件流水的总开销仅仅在调用这个函数时发生一次,这样就节省了程序在循环过程中所需的时间,程序的运行效率提高[11]。

4 中值滤波的程序优化

由中值滤波原理可知,中值滤波算法中包括数据排序的工作。如果采用传统的冒泡排序算法,很明显它需要进行大量的数据比较和交换工作,耗时比较大,占用了中值滤波算法的大部分时间,不利于图像的快速实时处理。如果改用快速排序的算法[12,13],它的基本思想是,在要排序的元素数组中任意选取一个元素作为中间值,并将其与其它元素进行比较,将所有比这个元素小的元素都放在它之前,将所有比它大的元素放在它之后;经过一次排序之后,可按该元素所在的位置分界,将数组分成两个部分;然后对剩下的这两个部分重复上述过程进行排序,直到每一部分只剩下一个元素为止。当所有排序完成后,取排序后集合中位于中间位置元素的值作为中值。例如,对于同样一个数值数组:data[25]={9,9,8,1,6,5,4,1,2,3,4,5,6,7,8,9,4,2,3,4,2,7,9,11,22},采用冒泡法排序算法,数据需要交换的次数是120;而快速排序算法,数据需要交换的次数仅为35。由此看来,采用快速排序算法比冒泡法排序算法的代码运行效率要高。这样,可以把这个步骤看作是对算法的初步优化("优化一")。

作为举例分析,选取窗口为5×5,对一幅16 bit有符号图像进行中值滤波处理,它的部分程序如下:

```
for( irows = 0; irows < aSz. height; irows ++ )
  {
    for( icols = 0; icols < aSz. width; icols ++ )
    {
        for (j = 0; j < 5; j ++ )
        {
            for (i = 0; i < 5; i ++ )
            {
    temp[j * 5 + i] = inptr - > pData[(irows +
j) * aSz. width + icols + i];
            }
        }
    quicksort (&temp,0,24);//调用排序函数
    outptr - > pData[irows * aSz. width + icols] =
temp[12];
    }
  }
```

先对上述程序进行初步的优化,代码如下:

优化二:

```
for( irows =0;irows < aSz. height;irows ++ )
    {
        f1 = irows  *  aSz. width;
        for( icols =0;icols < aSz. width;icols ++ )
    {
            f2 = f1  +  iclos;
            for ( j 0; j <5; j ++ )
            {
                f3 = j * aSz. width;
                f4 = f2 + f3;
                f5 = j * 5;
                for ( i =0; i <5; i ++ )
                {
                    temp[ f5 + i ] =
                    inptr - > pData[ f4 + i ];
                }
            }
            quicksort ( &temp,0,24);
            outptr - > pData[ f2 ] = temp[ 12 ];
        }
```

与原始 C 程序相比,上面的程序中多定义了 5 个变量 f1、f2、f3、f4 和 f5,这是因为程序中存在大量的循环工作,需要进行大量重复的计算。例如,f1 = irows * aSz. width 的定义,与原程序相比,至少减少了((aSz. width - 1) × aSz. height)次的乘法(irows * aSz. width)运算;f2 = f1 + iclos 的定义,与原程序相比,省去了((24 × aSz. width × aSz. height))次相同的加法运算;f5 = 5 * j 的定义,省去了(20 × aSz. height × aSz. width)次相同的乘法运算。由此看来,这些变量的定义都是非常有效的,它可以大大地减少程序中那些冗余的计算,减少程序的运行时间。

程序运行时需要进行大量的乘法运算,由于用普通 C 语句来实现乘法和加法运算的代码执行效率低,可以用相应内联函数来代替普通的 C 语句,使程序的执行效率进一步提高。代码如下:

优化三:

```
for( irows =0;irows  <  aSz. height;irows ++ )
    {
        f1 = _mpy( irows,aSz. width);
        for( icols =0;icols < aSz. width;icols ++ )
    {
        f2 = _add2( f1,iclos);//f2 = f1 + iclos;
        for ( j =0; j <5; j ++ )
        {
            f3 = _mpy( j,aSz. width);
            f4 = _add2( f2,f3);//f4 = f2 + f3;
            f5 = _mpy( j,5); //f5 = j * 5;
            for ( i  =  0; i <5; i ++ )
            {
                temp[ f5 + i ] =
                inptr - > pData[ f4 + i ];
            }
        }
        quicksort ( &temp,0,24);
        outptr - > pData[ f2 ] = temp[ 12 ];
    }
}
```

其次,程序中存在着多重循环,循环体占用了程序运行的大量时间,需要进行优化。为此,将最内层循环展开。

优化四:

```
for ( j =5; j >0; j - - )//将循环变量改为递减,让此段代码执行软件流水
    {
        f3 = _mpy( j,aSz. width);
        f4 = _add2( f2,f3);
        f5 =  _mpy( j,5);
        temp[ f5 ] =  inptr - > pData[ f4 ];
        temp[ f5 + 1 ] = inptr - > pData[ f4 + 1 ];
        temp[ f5 + 2 ] = inptr - > pData[ f4 + 2 ];
        temp[ f5 + 3 ] = inptr - > pData[ f4 + 3 ];
        temp[ f5 + 4 ] = inptr - > pData[ f4 + 4 ];
    }
```

为了充分利用 DSP 芯片的内部资源,将循环变量改为递减(通知编译器循环的次数),然后,将最内层循环完全展开,因为编译器只对内部循环才执行软件流水。在CCS2软件平台下,使用

编译器选项中的 -O3 和 -pm,使这段代码实现软件流水。

5 实验结果

利用文中介绍的基于 TMS320C6000 程序优化方法对原始中值滤波程序进行了4次不同程度的优化,在 CCS 2 软件平台下选择 C64XX DSP 做仿真,编译命令是:c16x-o-g-k medianfilter. c -z medianfilter. cmd -o medianfilter. out,程序运行的时间如表4所示。

表4 实验结果

Tab. 4 Experimental results

程序	原始程序	优化一	优化二	优化三	优化四
时间(ms)	8.37	4.62	4.13	3.85	2.91

由表中可以看出,程序执行时间从原来的 8.37 ms 减少到了 2.91 ms,但是中值滤波的效果相同。如图5所示,图5(a)是带有椒盐噪声的 Lena 图像,图5(b)是使用优化前的中值滤波程序对图(a)进行优化的结果,图5(c)是用优化后的中值滤波程序对图(a)进行优化的结果。

(a) 带有椒盐噪声的Lena图像
(a) Lena image with salt and papper noise

(b) 优化中值滤波程序前的Lena图像
(b) Before optimization for median filter

(c) 优化中值滤波程序后的Lena图像
(c) After optimization for median filter

图5 实验图像比较

Fig. 5 Comparison of experimental images

6 结 论

本文介绍了基于 TMS320C6000 代码优化方法,主要包括:使用编译器选项、内联函数、字访问短型数据、软件流水和循环展开等。文中在 CCS 2 软件平台下选择 C64XX DSP 做仿真,采用上述方法对中值滤波程序进行优化,使中值滤波处理的时间由 8.37 ms 减少到 2.91 ms,这对图像预处理具有一定的意义;同时也说明,使用这些程序优化方法可以提高 C6000 的 C 语言代码的执行性能。根据不同的情况使用相应的程序优化方法,使程序的执行时间达到了要求。需要指出一点,如果使用这些方法后,程序的运行时间仍不能满足要求,可将 C 语言代码中耗时最长的部分抽取出来用线性汇编语言重写,用汇编优化器优化,使其最终达到所需的指标。

参考文献:

[1] 李兰英. Nios Ⅱ嵌入式软核 SOPC 设计原理及应用[M]. 北京:北京航空航天大学出版社,2006.
LI L Y. *Embedded Soft-core Nios Ⅱ Design Principles and Applications of SOPC*[M]. Beijing:Publishing of Beijing University of Aeronautics and Astronautics,2006. (in Chinese)

[2] 罗坤,肖铁军. 基于 DSP Builder 的并行中值滤波算法的设计与实现[J]. 计算机工程与设计,2009,30(6):1413-1416.
LUO K,XIAO T J. Implementation of parallel median filtering algorithm based on DSP Builder[J]. *Computer Eng. Des.*, 2009,30(6):1413-1416. (in Chinese)

[3] 冯新宇,方伟林,杨栋. 基于中值滤波与 Sobel、Canny 算子的图像边缘检测研究[J]. 黑龙江水专学报,2009,(1):101-103.
FENG X Y,FANG W L,YANG D. Image edge detection based on the median filtering and Sobel, Canny Operator[J]. *J. Heilongjiang Hydraulic Eng. College*,2009,(1):101-103. (in Chinese)

[4] 尹剑仑,卫武迪. 一种改进的自适应中值滤波算法研究[J]. 设计与研究[J],2009,(1):31-32.
YIN J L,WEI W D. Research of an improved adaptive median filtering algorithm[J]. *Design and Research*,2009,(1):31-32. (in Chinese)

[5] 刘继业,费如纯. 一种滤除椒盐噪声的改进的中值滤波算法[J]. 辽宁科技学院学报,2008,12(10):24-26.
LIU J Y, FEI R C. An improved median filtering algorithm for salt and pepper noise[J]. *J. Liaoning Inst. Sci. Technol.*,2008,12(10):24-26. (in Chinese)

[6] 任丽香. TMS320C6000 系列 DSPs 的原理与应用[M]. 北京:电子工业出版社,2000.
REN L X. *Principle and Application of TMS320C6000 DSPs*[M]. Beijing:Publishing House of Electronics Industry,2000. (in Chinese)

[7] 李方慧,王飞,何佩琨. TMS320C6000 系列 DSPs 原理与应用[M]. 2 版. 北京:电子工业出版社,2003.
LI F H,WANG F,HE P K. *Principle and Application of TMS320C6000 DSPs*[M]. 2nd ed. Beijing:Publishing House of Electronics Industry,2003. (in Chinese)

[8] Texas Instruments Co. Code Composer Studio Development Tools v 3.3[M/OL]. (2008-10)[2009-01-11]http://focus.ti.com/lit/ug/spru509n/spru509n.pdf.

[9] 纪铁军,任丽军,赵爱明. 基于 TMS320C6000 系列 DSP 的 C 代码优化方法[J]. 单片机与嵌入式系统应用,2003,(7):76-77.
JI T J,REN L J,ZHAO A M. Code optimization of TMS320C6000 series DSP[J]. *Microcontrollers & Embedded System*[J],2003,(7):76-77. (in Chinese)

[10] 赵训威,王东昱,张平. TMS320C6000 嵌入式系统优化编程的研究[J]. 电子技术应用,2001,27(4):71-73.
ZHAO X W,WANG D Y,ZHANG P. Study of code optimization based TMS320C6000 embedded system[J]. *Appl. Electronic Technique*,2001,27(4):71-73. (in Chinese)

[11] Texas Instruments Co. TMS320C6000 Optimizing Compiler v 6.0 Beta[M/OL]. (2005-07)[2009-01-11]http://focus.ti.com/lit/ug/sprus198n/spru187n.pdf.

[12] 骆剑锋. 简单快速排序算法[J]. 电脑与电信,2007,6(9):11-12.
LUO J F. An easy fast sorting algorithm[J]. *Computer and Telecom.*[J],2007,6(9):11-12. (in Chinese)

[13] 吴加富,樊景峰. 一种改进的快速中值滤波算法[J]. 济源职业技术学院学报,2009,3(8):21-23.
WU J F,FAN J F. An improve fast median filter algorithm[J]. *J. Jiyuan Vocational and Technical College*,2009,3(8):21-23. (in Chinese)

作者简介:黄德天(1985—),男,汉族,福建龙岩人,硕士研究生,主要从事数字图像的采集与处理方面的研究。
E-mail:dthuang@sina.com
陈建华(1987—),男,汉族,浙江义乌人,硕士研究生,主要从事数字图像的采集与处理方面的研究。
E-mail:jhchen_zju@163.com