ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.2, February 2009, pp.246−255
doi: 10.3724/SP.J.1001.2009.03296
© by *Institute of Software*, *the Chinese Academy of Sciences*. All rights reserved.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于参数化存储结构的滑动窗口IP核自动生成*

窦 勇+, 董亚卓, 徐进辉, 邬贵明

(国防科学技术大学 计算机学院,湖南 长沙 410073)

Automatic Generation of IPCore for Sliding-Window Operations Based on a Parameterized Memory Architecture

DOU Yong+, DONG Ya-Zhuo, XU Jin-Hui, WU Gui-Ming

(School of Computer, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: E-mail: yongdou@nudt.edu.en

Dou Y, Dong YZ, Xu JH, Wu GM. Automatic generation of IPCore for sliding-window operations based on a parameterized memory architecture. *Journal of Software*, 2009,20(2):246−255. http://www.jos.org.cn/1000-9825/3296.htm

Abstract: In order to deal with the memory bottleneck in high level synthesis tools for sliding-window operation, this paper presents a parameterized memory architecture for high level synthesis to automatically generate the hardware frames for all window processing applications. A three-level memory structure is designed to use inner-loop and outer-loop data completely, and at the same time shifted registers are used to make hardware design simpler. Compared with the related works, this approach can achieve a speedup of 2.13 to 3.8 times, and enhance the execution frequency from 69MHZ to 238.7MHZ.
Key words: sliding-window operation; high level synthesis; memory structure; data schedule; template

摘 要: 为解决目前高级综合方法在处理滑动窗口程序时存在的存储系统设计瓶颈问题,提出了参数化存储体系结构模型.采用三级存储层次,充分开发内层循环、外层循环的数据重用;采用寄存器平移策略,简化硬件设计.与相关工作相比,这种体系结构模型使用相对较少的存储资源,将程序执行速度提高了2.13倍~3.8倍,将执行频率由相关工作的69MHZ提升到了238.7MHZ.
关键词: 滑动窗口操作;高级综合;存储结构;数据调度;模板
中图法分类号: TP311 **文献标识码**: A

SoC核心技术之一是IP核(intellectual property)重用.SoC设计需要尽可能地使用现有IP,以搭积木的方式完成大部分设计.面向应用的IP核设计是SoC创新性的体现,也是制约SoC快速构建的关键.目前,开发硬件IP核仍然沿用传统RTL级硬件设计的方法,随着问题规模的不断扩大,其设计的复杂性和较长的设计周期已经越来越成为困扰芯片开发人员的难题.传统的设计方法使得在软件和硬件之间很难进行早期的平衡和优化,并严重影响开发成本和开发周期.因此,在系统级设计与电路级设计之间架设一座桥梁已经成为集成电路设计领域极为迫切的任务,硬件高级综合技术成为系统级设计的前沿研究课题.

* Supported by the National Natural Science Foundation of China under Grant No.60633050 (国家自然科学基金)
Received 2007-03-13; Accepted 2008-02-27

高级综合技术是一个把高层次描述转化为低层次实现的过程,设计目标是提高系统抽象级别,简化硬件开发过程,缩短设计周期.比较著名的研究成果有 System C,Handel_C[1],ASC[2],SPC[3],Stream-C[4],ROCCC[5]等.然而,当前高级综合工具尚未达到理想的实用效果,其中有关存储系统的性能优化是所有高级综合系统都必须要解决的挑战性问题[6].

本文主要探讨如何针对一种特殊的应用类型——滑动窗口应用,自动生成高效的存储结构.滑动窗口广泛应用于图像处理、模式识别、数字信号处理和多媒体处理等领域,具有数据量大、计算密集等特点.针对滑动窗口应用的硬件IP核生成,特别是优化的存储结构设计,是当前高级综合的一个研究热点[7].但是,目前的体系结构模型都存在各种不足,或者没有充分的开发数据重用,或者为了实现数据重用使用了过多的硬件资源,或者没有提出自动的数据调度方法.我们针对滑动窗口应用,提出了一种新的参数化 3 层存储体系结构模型,给出了相应的参数自动提取和硬件模块自动生成算法[8,9].我们的设计目标是充分开发滑动窗口应用中的内层循环和外层循环数据重用,减少冗余的存储访问;同时,将访存和计算过程重叠,提高并行度;模块化数据调度状态机和窗口寄存器组,降低硬件复杂度,提高硬件时钟频率.

本文第 1 节介绍滑动窗口应用的定义及其研究现状.第 2 节介绍参数化存储结构模型和组成部件.第 3 节介绍参数提取方法.第 4 节介绍硬件模块自动生成算法.第 5 节给出实验结果和性能分析.第 6 节作总结.

1 滑动窗口应用及其研究现状

滑动窗口应用是对一维或二维数组的循环操作(二维以上可以转换为多个二维滑动窗口),如图1所示.具有固定尺寸的数据区域称为窗口,在被处理数组上向特定方向移动.数组上的窗口数据与程序对应的特定模板进行某种操作,模板的大小与滑动窗口的大小相同,通常产生一个计算结果.

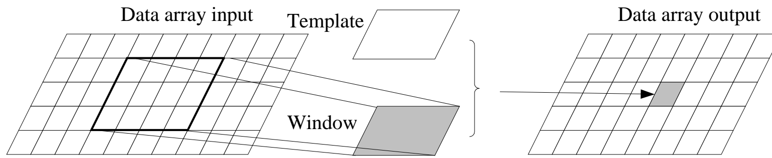


Fig.1 Sliding-Window operation

图 1 滑动窗口应用

定义I为一个$M\times N$大小的输入数组,O为产生的输出数组.定义I_d_i为数组I在i方向上的最大输入相关距离,I_d_i为数组I在j方向上的最大输入相关距离.定义$\langle i,j\rangle$为循环访问顺序,$\langle 0,1\rangle$表示i为外层循环,j为内层循环;$\langle 1,0\rangle$表示j为外层循环,i为内层循环.假设我们所处理程序的运算单元规模不超过片上SoC系统的资源总量,那么一个滑动窗口应用应满足如下两点:

(1) 循环初值、终值和步进值固定,不在执行过程中改变;

(2) I_d_i和I_d_i为两个与i和j无关的常数.

一个滑动窗口应用可以表示为

$$O_{rc}=F(f(w_{pq},I_{r+p,c+q}))\ \forall(p,q)\in m\times n,\ \forall(r,c)\in M\times N,$$

其中,$m\times n$为滑动窗口大小, $m=\begin{cases}I_d_j, & \text{if } \langle i,j\rangle=\langle 0,1\rangle\\ I_d_i, & \text{if } \langle i,j\rangle=\langle 1,0\rangle\end{cases}$, $n=\begin{cases}I_d_i, & \text{if } \langle i,j\rangle=\langle 0,1\rangle\\ I_d_j, & \text{if } \langle i,j\rangle=\langle 1,0\rangle\end{cases}$.

w_{pq}表示模板中坐标为p,q位置的系数;$I_{r+p,c+q}$表示滑动窗口中坐标为p,q的操作数;f定义对模板数据和窗口数据的操作函数;F定义其他与数组I无关的操作.在滑动窗口应用中存在着大量的数据重用.充分地开发数据重用,是加快滑动窗口应用执行速度的关键.

2 参数化的存储体系结构模型

针对滑动窗口应用的特点,本文提出了参数化的存储体系结构,如图 2 所示.

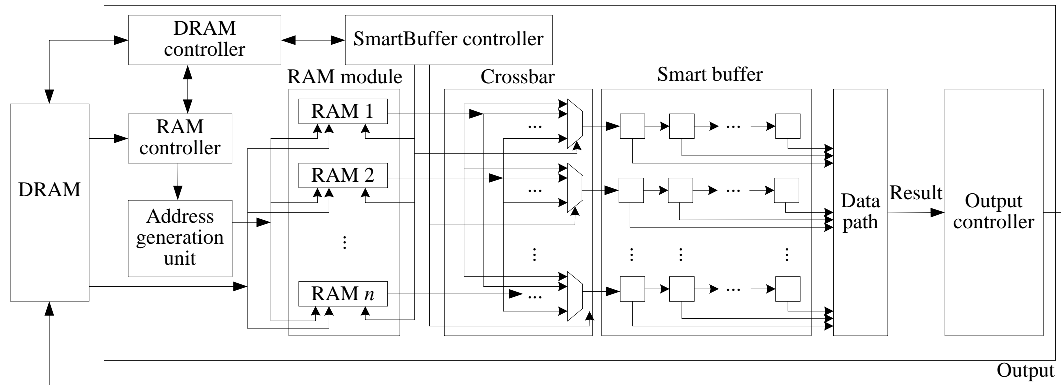


Fig.2 Parameterized memory architecture model

图 2 参数化的存储体系结构模型

我们提出的参数化存储体系结构模型主要包括 3 个存储层次:外部存储器 DRAM 用于存储所有输入阵列数据;片内暂存器 RAM 用于存储数据阵列的部分行或列,开发外层循环的数据重用;寄存器组 Smart buffer 用于存储窗口数据,开发内层循环的数据重用.存储层次由 3 个数据调度状态机控制,分别对应 DRAM 存储调度器、RAM 调度器和 Smart buffer 调度器.Smart buffer 与运算单元阵列直接相连.参数化存储结构模型的数据调度遵循两条规则:一是片内 RAM 保存后续循环迭代中将用到的数据,直到这些数据不再被使用为止;二是 Smart buffer 仅保存当前窗口数据.第 1 条规则保证可以充分利用数据重用,第 2 条规则保证 Smart buffer 尽可能地小.参数化存储体系结构的结构固定,其规模和调度方式由窗口应用算法的特点决定,编译器可以通过分析程序,自动来提取决定底层硬件结构的参数值.

3 参数提取

3.1 参 数

根据给定的滑动窗口应用,自动生成上述体系结构模型,需要编译器获得如下一些参数值:

1. 访存起始地址.
2. 窗口滑动方向:⟨0,1⟩或者⟨1,0⟩.
3. 输入数组的宽和高:*width* 和 *height*.
4. 滑动窗口的宽和高:*buffer_width* 和 *buffer_height*.
5. 滑动窗口的水平跨度和垂直跨度:*buffer_span_width* 和 *buffer_span_height*.
6. 设计的 RAM 模块的个数、位宽和深度:*Ram_num*,*Ram_width* 和 *Ram_depth*.
7. 每个时钟周期要产生的结果个数:*Result_num*.
8. 从片外存储器 DRAM 中读出的一个数据要写入几个片内 RAM 中:*Control_num*(因为在一些情况下,可能要将输入数据的高位写入一个 RAM,将低位写入另外一个 RAM).
9. j 层循环到达循环终值,步进 i 层循环时,更新 RAM 所需要的时钟周期数:*Update_Ram*.
10. 每执行一次窗口计算,准备下一窗口数据所需要的时钟周期数:*Update_buffer*.

3.2 参数提取

计算这些参数,需要的输入是:内层循环的初值 *loopj_i*0、终值 *loopj_inn*、步进值 *loopj_ip*,外层循环的初值

$loopi_i0$、终值 $loopi_inn$、步进值 $loopi_ip$,输入阵列 I 在 j 方向上的最大相关距离 I_d_i、输入阵列 I 在 i 方向上的最大相关距离 I_d_i、输出阵列 O 在 j 方向上的最大相关距离 O_d_i、输出阵列 O 在 i 方向上的最大相关距离 O_d_i,相关距离为+∞表示不存在对数组阵列在该维的访问.

根据重用方式的不同,将所有的输入数组分为 4 种类型,我们穷举了一维和二维滑动窗口应用中输入数组可能的访存情况,归纳出了这 4 种重用类型的判断条件.限于篇幅,具体过程不在本文中详细叙述.

C^{ϕ}:不存在数据重用,数据一旦使用后,不会在后续迭代中再次被使用.判断条件为$(loopi_ip{=}{=}0)\&\&(loopj_ip>I_d_j)$ 或者 $(loopi_ip>I_d_i)\&\&(loopj_ip>I_d_i)$ 或者 $(I_d_i=+\infty)\&\&(loop1_ip>I_d_j)$ 或者 $(loopi_ip>I_d_j)\&\&(I_d_j=+)$.

$C^{i\phi}$:只存在层内重用,不存在层间数据重用.判断条件为$((loopi_ip{=}{=}0)\&\&(loopj_ip<I_d_j))$或者$((loopi_ip>I_d_j)\&\&(loopj_ip<I_d_j))$或者$((I_d_j=+\infty)\&\&(loopi_ip<I_d_j))$.

C^{c}:只存在层间数据重用,不存在层内数据重用.判断条件为$((loopi_ip<I_d_j)\&\&(loopj_ip>I_d_j))$或者$((loopi_ip<I_d_j)\&\&(I_d_j=+\infty))$.

C^{ic}:既有层内重用又有层间重用,判断条件为$(I_d_j\geq loopi_ip)\&\&(I_d_j\geq loopj_ip)$.

我们穷举了 4 种数据重用模式中一维和二维数组的访存方式,归纳参数提取公式如下(限于篇幅,计算公式的归纳过程不在本文中详细介绍,定义 $WIDTH$ 为片外存储系统和片内存储系统的数据输入带宽;I_width 为输入数组中每个元素的位宽):

1. 窗口滑动方向:由 i,j 维的访问顺序决定,$\langle 0,1\rangle$沿水平方向滑动,$\langle 1,0\rangle$沿垂直方向滑动.
2. $width=loopj_inn-loopj_i0$; $height=loopi_inn-loopi_i0$.
3. $buffer_width=\max\{I_d_j,loopj_ip\}$; $buffer_height=\max\{I_d_j,loopi_jp\}$.
4. $buffer_span_width=loopj_ip$; $buffer_span_height=loopi_ip$.
5. $$Ram_num=\begin{cases}buffer_height=\max(I_d_i,loopi_jp), & C^{i\phi}\ C^{c}\ C^{ic}\\ 0, & C^{\phi}\end{cases};$$

   $$Ram_depth\begin{cases}\begin{cases}loopj_inn-loopj_i0, & \text{if } I_d_j\neq+\infty\\ loopi_inn-loopi_i0, & \text{otherwise}\end{cases}, & C^{i\phi}\ C^{C}\ C^{ic}\\ 0, & C^{\phi}\end{cases};$$

   $Ram_width=I_width$.
6. $Result_num=O_d_j\times O_d_j$.
7. $Control_num=\min\{\lfloor WIDTH/I_width\rfloor,Ram_num\}$.
8. $$Updata_Ram\begin{cases}0, & C^{\phi},C^{c},C^{ic}\\ (\lceil Ram_num/Control_num\rceil-1)*Ram_depth, & C^{i\phi}\end{cases}.$$
9. $$Updata_buffer=\begin{cases}I_d_j, & C^{\phi}\\ (buffer_width-1)\times buffer_height, & C^{C}\\ 0, & C^{i\phi},C^{ic}\end{cases}.$$

4 模块自动生成算法

模块自动生成算法将根据不同应用确定的参数值,自动产生 VHDL 代码.根据提取的参数,首先构建各个独立模块,包括多体 RAM、RAM 调度器、Smart buffer 及数据调度状态机和运算阵列等;其次,构建连接各个模块的连线模块;最后构建滑动窗口的整体封装程序.整体封装程序和各个逻辑模块描述了实现滑动窗口 IP 的硬件逻辑结构,该逻辑结构引用了存储块、运算单元、多路选择器等基本操作部件(有关基本操作部件的优化和选择不在本文中叙述).本节对几个主要调度模块的生成算法进行说明.算法中粗斜体部分为编译器确定的参数,根据不同目标应用的特点确定相应参数后,调度模块的执行过程也就确定了.

4.1 DRAM调度器

DRAM 调度器控制程序执行全过程,包括 4 个状态,如算法 1 所示.

1. Idle:空闲状态,等待启动信号为真,启动整个程序的执行.
2. RAM_initialization:片上 RAM 的初始化状态,初始化 RAM 中的数据,*Ram_in* 为计数器,记录初始完毕的 RAM 行数,直到(*Ram_in*==*Ram_num*−*Control_num*+1),所有的 RAM 行初始化完毕,跳转到 SmartBuffer_initialization 状态.
3. SmartBuffer_initialization:初始化 Smart buffer.在前面几个 RAM 中的数据初始化完毕后,在为最后一个 RAM 输入数据的同时,执行 Smart buffer 的初始化,直到第 1 个窗口数据准备就绪(*smart_buffer_ok*==1),开始计算处理,跳转到 Processing 状态.
4. Processing:Smart buffer 中的数据初始化完毕,开始执行计算.这个过程中,存在数据的流水化调度,从片外存储系统中读入的数据流入片内 RAM 中,再从片内 RAM 送往 Smart buffer,再传递给计算单元,执行计算.计数器 *counter_done* 记录当前行处理完毕的数据个数,当前行处理完毕后(*counter_done*==*width*−*buffer_width*+1),准备处理下一行数据,跳转到 SmartBuffer_initialization 状态,再次执行 Smart buffer 的初始化.计数器 *row_counter* 记录处理完毕的行数,当(*row_counter*==*height*−*buffer_height*+1),整个程序执行完毕,返回 Idle 状态.

不同的滑动窗口应用,整体调度过程固定,具体执行参数值不同.

算法 1. DRAM 调度器生成算法.

Parameter: ***Ram_num***, ***Control_num***, ***Update_Ram***, ***buffer_width***, ***width***, ***buffer_height***, ***height***.

Process whole_state (input signal: *start*. input channel: *Ram_in*, *smart_buffer_ok*, *result_num*. output signal: *finish*. output channel: *whole_state*)

```
begin
repeat begin
  case (whole_state)
    1. Idle:
       if (start==1) finish≤0;
           goto RAM_initialization;
       else goto Idle;
    2. RAM_initialization:
    if (result_num==0)&&EQ(Ram_in==Ram_num−Control_num+1),
           goto SmartBuffer_initialization;
    // Ram_in记录正在执行数据输入的RAM编号
        else wait Update_Ram时钟周期,完成RAM的更新,
           goto SmartBuffer_initialization;
    3. SmartBuffer_initialization:
        if (smart_buffer_ok)   //初始化完毕
               goto Processing;
    // smart_buffer_ok记录SmartBuffer初始化完毕
         else goto SmartBuffer_initialization;
    4. Processing:
       if (counter_done==width−buffer_width+1)   //当前行数据处理完毕
  // counter_done记录当前行送计算单元的数据个数
   row_counter≤row_counter+1;
```

```
  // row_counter记录处理完毕的阵列数组行数
 if (row_counter==height−buffer_height+1) {
    finish≤1;
    goto Idle;}   //整个程序处理完毕
  else
     Ram_done≤Ram_done+Control_num)
        // Ram_done标识当前处理的RAM编号
     if (Ram_done==Ram_num)
         Ram_done≤1;
     if (Update_RAM==0),
         goto SmartBuffer_initialization;
      else goto RAM_initialization;
    else wait Update_buffer个时钟周期
         goto Processing;
   endcase
 end
end
```

4.2 RAM调度器

RAM 调度器负责将片外存储系统中读入的操作数写入正确的 RAM 行的正确位置,如算法 2 所示.RAM 调度器包括两个状态:

1. Idle:等待输入数据有效,准备将该数据存入 RAM 行;
2. Write_RAM:将数据写入相应的 RAM 行中的相应位置.RAM 调度器有两个计数器,一个记录当前执行输入的 RAM 行编号(*Ram_in*),另一个记录当前 RAM 行已经输入数据的个数(*wr_address*).每写入一个新数组,地址计数器递增(*wr_address*≤*wraddress*+1),如果当前 RAM 写入完毕(*wr_address*==*Ram_depth*),则执行对下一个 RAM 块的写入,行计数器 *Ram_in* 递增,如果最后一个 RAM 写入完毕(*Ram_in*==*Ram_num*−*control_num*+1),则将新输入数据写入第 1 个 RAM 块.这两个计数器的计数过程根据具体参数值的不同而不同.

算法 2. RAM 调度器生成算法.

```
Parameter: Ram_depth, Control_num, Ram_num.
Processing data_input_state (input channel: data_in_av. output channel: wr_en, wr_address, Ram_in)
begin
repeat begin
  case (data_input_state)
  1. Idle: wr_en≤0   //当前RAM行写使能失效
     if (data_in_av==1); goto Write_RAM;
          else goto Idle;
  2. Write_RAM:
     wr_address≤wr_address+1;
          // wr_address记录当前RAM行的写地址
     wr_en≤1;   // wr_en为当前RAM行的写使能信号
     if (wr_address==Ram_depth)   //当前RAM行写入完毕
        Ram_in≤Ram_in+Control_num;
      if (Ram_in==(Ram_num−control_num+1))
```

```
            //最后一个编号的RAM写入完毕
        // Ram_in≤1    //重新开始对第1个RAM行执行写入
          if (data_in_av==1), goto Write_RAM;
            else goto Idle;
    endcase
  end
end
```

4.3 SmartBuffer调度器

SmartBuffer 调度器一方面负责将数据从 RAM 存储体中读出,另一方面产生控制信号控制交叉开关,将数据按照正确的顺序送往 Smart buffer 中的正确的寄存器单元,包括两个状态,如算法 3 所示.

1. Idle:空闲,等待 RAM 中的数据初始化完毕;

2. Write_SmartBuffer:将数据从 RAM 中读出,送往 Smart buffer.计数器 *buffer_in_num* 记录 Smart buffer 是否初始化完毕,初始化完毕后(*smart_buffer_ok*==1),将 Smart buffer 中的数据送往计算单元执行计算.计数器 *rd_address* 记录从 RAM 中读出的数据位置,如果读到 RAM 的最后一个地址,则再次从地址 0 开始读出数据.

算法 3. SmartBuffer 调度器生成算法.

Parameter: ***width***, ***buffer_width***.

Processing data_schedule_state (input channel: *whole_state*, *wr_address*. output channel: *rd_en*, *smart_buffer_ok*, *rd_address*, *Ram_en*, *buffer_in_num*)

```
begin
repeat begin
  case (data_schedule_state)
  1. Idle: rd_en≤0; smart_buffer_ok≤0;
   if (rd_address<wr_address)&&((whole_state==smart_buffer_initialization)||(whole_state==processing))
     goto Write_SmartBuffer;
   else goto Idle;
  2. Write_SmartBuffer:
      rd_en≤1;
      if (!(buffer_in_num==0)) {
         buffer_in_num≤buffer_in_num−1;
          // buffer_in_num记录SmartBuffer行初始化数据个数
        smart_buffer_ok≤0;}
      else smart_buffer_ok≤1;
         Ram_en[Ram_current_do]≤1;
         rd_address≤rd_address+1;
      if (rd_address==(width−1))
         rd_address≤0;
         Ram_current_do≤Ram_current_do+1;
          // Ram_current_do记录当前处理的RAM行编号
      if (Ram_current_do==n) {
         Ram_current_do≤1;
         buffer_in_num≤buffer_width−1;}
      if (wr_address≥rd_address), goto Idel;
```

```
            else goto Write_SmartBuffer
        endcase
    end
end
```

4.4 输出结果调度器

输出结果调度器对结果进行计数,判断程序是否执行完毕,如算法 4 所示.输出结果调度器包括两个计数器:*result_counter* 记录产生的结果个数,每产生一个计算结果,*result_counter* 递增;*accomplish_row* 记录已经处理完毕的行个数,直到所有行都处理完毕后(*accomplish_row*==(*height*−*buffer_height*+1)),整个程序执行完毕.

算法 4. 输出结果调度器生成算法.

```
Parameter: width, buffer_width, height, buffer_height.
Processing result_output_controller (input channel: result_av. output channel: accomplish)
begin
repeat begin
  if (result_av==1)
    result_counter≤result_counter+1;
// result_counter记录当前处理行已经得到的计算结果个数
    if (result_counter==(width−buffer_width+1));
    //已经得到当前行所有的计算结果
    accomplish_row≤accomplish_row+1;
            // accomplish_row记录已经处理完毕的行数
  if (accomplish_row==(height−buffer_height+1))
   //所有行处理完毕
   accomplish≤1;
  end
end
```

对于其他一些模块:RAM 暂存器、交叉开关,Smart buffer 根据具体应用的特点,需要进行不同规模的设计.我们设计多组模块,针对特定的应用调用相应的模块组.

5 实验与性能比较

选择 5 个典型滑动窗口程序:5-tap FIR,图像锐化 Image sharping,Sobel,MedianFilter 和 2D_Lowpass_filter,用于性能测试.其中,5-tap FIR 是针对一维数组的滑动窗口,其余 4 个程序是针对二维数组的滑动窗口.实验分为两个部分.一部分是在资源受限的条件下,与没有数据重用的体系结构模型和 Diniz[10]数据重用方法在存储资源、寄存器资源和执行速度方面进行比较,说明本文提出的 3 层存储结构模型采用更加合理的存储结构实现数据重用,能够实现使用较少的资源,达到更少的程序执行节拍.该实验在 ModelSim SE 6.2h 工具上模拟实现.另一部分是与 California 大学的 ROCCC[5]系统在硬件时钟频率和系统整体性能方面进行比较,同样采用窗口 Smart buffer 实现数据重用,本文的存储模型在结构上能够达到更快的时钟频率.该实验在 Xilinx ISE 7.1 工具上综合实现,目标器件为 Xilinx xc2s50e-5.

5.1 与无Smart buffer和Diniz方法的比较

比较 3 个程序:Image sharping(SHARP),Sobel edge detection(SOBEL)和 MedianFilter(FILTER).对每个测试程序,我们选择 3 种实验规模,表 1 所示为 3 种方法在寄存器、存储资源使用数量和执行节拍上的比较结果.第 2 列为定义的程序规模:$A\times B$,其中,A 为外层循环步进次数,B 为内层循环步进次数.

无 Smart buffer 的方法没有数据重用机制,将所有阵列数据初始化到片内 RAM 中,每次计算要访问片内存储系统,获得所有的窗口数据.这种方法一方面使用了较多的存储资源;另一方面多次访问片内存储系统的同一地址,导致程序执行节拍长,执行效率不高.Diniz 方法将所有输入阵列保存在片内存储系统中,将所有重用数据保存在寄存器组中,这种方法一方面需要大量的存储资源和寄存器资源;另一方面,没有数据流水化调度机制,在计算过程不存在片内外的数据调度,所以需要更多的时间执行片上数据初始化,程序执行节拍较长.

Table 1 Comparison with no data reuse and Diniz's approach

表 1 与没有数据重用和 Diniz 方法的比较

Testbench	Scale	Without smart buffer			Diniz			Ours		
		Registers	Memory elements	Cycles	Registers	Memory elements	Cycles	Registers	Memory elements	Cycles
SHARP	32×8	5	256	1 636	17	256	985	5	16	768
	32×16	5	512	3 584	19	288	2 224	5	32	1 536
	320×32	5	1056	71 360	19	972	45 264	5	65	30 720
SOBEL	64×8	10	512	5 505	25	512	1 977	10	25	1 536
	64×16	10	1 024	11 577	31	640	4 973	10	48	3 072
	640×32	10	1 024	234 420	31	1 090	101 808	10	96	61 440
FILTER	16×8	10	128	909	25	128	489	10	16	384
	16×16	10	256	1 839	31	160	1 230	10	48	768
	160×32	10	1 088	41 895	31	820	25 032	10	96	15 360

假设 SoC 片上系统资源有限:1 100 个存储单元和 32 个寄存器单元.在这个资源限制下,因为无 Smart buffer 和 Diniz 方法对存储资源需求量较大,并不能将所有的输入数组保存到有限的片内 RAM 中,所以要将那些超出片上资源限制的程序分块执行.如为了处理 SHARP 程序的第 3 个问题规模,按照 Diniz 方法,将需要 10 240 个存储单元和 64 个寄存器,这时要将源程序划分为 12 个 108×9 问题规模的小程序.从表 1 中可以看出,本文提出的参数化存储体系结构模型合理地使用了存储资源和寄存器资源,同时将访存和计算重叠,实现数据流水化调度,使用较少的硬件资源,获得更快的执行速度.相对于无 Smart buffer 的方法,Diniz 方法将速度提高了 1.54 倍~2.78 倍,我们的参数化存储体系结构将速度提高了 2.13 倍~3.81 倍.

5.2 与ROCCC系统的比较

选择 4 个测试程序进行这一实验:5-tap FIR,2D_Lowpass_filter,Image sharping 和 Sobel,其中将 2D_Lowpass_filter 程序在水平和垂直方向上分别执行一次循环展开.设定一维数组的问题规模为 256,二维数组的问题规模为 64×64.综合结果见表 2.其中,有关 ROCCC 的实验数据来自于文献[5],因为该文献中只有前两个程序的测试结果,所以后面两个测试程序只给出了本文的参数化体系结构模型的执行结果.

Table 2 Comparison with ROCCC

表 2 与 ROCCC 的比较

	5-tap FIR		2D_Lowpass_filter		Image sharping	Sobel
	ROCCC	Ours	ROCCC	Ours	Ours	Ours
Smart buffer's statues	14	5	18	4	2	3
Control area (slices)	210	262	542	484	131	216
Clock rate (MHz)	94	238.664	69	238.664	238.664	238.664
Execution time (cycles)	262	263	5 980	2 057	4 042	4 108
Throughput (results number/cycles)	0.96	0.96	0.64	1.87	0.98	0.94

ROCCC 只能处理层内数据重用而不能处理层间数据重用,所以,与 ROCCC 相比,本文的参数化体系结构模型能够减少程序的执行节拍.表 2 中的 5-tap FIR 测试程序的执行节拍没有减少,这是因为 5-tap FIR 是一维滑动窗口操作,本来就不存在层间数据重用.在 ROCCC 中,Smart buffer 中的寄存器和计算单元全互连,需要复杂的仲裁机制实现控制.在我们提出的参数化存储体系结构模型中,Smart buffer 采用了寄存器平移策略,保证 Smart buffer 与计算单元的连接固定不变,简化了硬件设计,因此,时钟频率达到 238.7MHz.为 4 个测试程序生成的目标体系结构,其执行频率基本相同,这是因为我们所设计的是针对所有的滑动窗口应用通用的参数化存储体系结构模型,对于不同的设计,只需修改参数值,而控制状态机不变.

6 总 结

滑动窗口操作广泛应用于多媒体、图像和数字信号处理中,这部分代码的执行速度在很大程度上制约着整个程序的执行性能.在本文中,我们针对窗口滑动应用,提出了一种参数化存储体系结构模型,实现三级存储层次,利用循环程序内外两层数据重用性,采用寄存器平移策略,保证 Smart buffer 和运算单元之间连接关系固定不变,降低了硬件复杂度.与相关工作相比,我们提出的结构模型使用较少的存储资源,达到了更少的程序执行节拍和更高的时钟频率.在后续工作中,我们将进一步探讨循环展开和折叠对性能的影响,探索设计空间开发,以最大化利用片内资源,加快程序的执行速度.

References:

[1] Celoxica. Handel-C Language Reference Manual for DK2.0. Document RM-1003-4.0, 2003.

[2] Mencer O, Pearce DJ, Howes LW, Luk W. Design space exploration with a stream compiler. In: Proc. of the IEEE Int'l Conf. on Field-Programmable Technology (FPT). Tokyo: IEEE Press, 2003. 270−277.

[3] Weinhardt M, Luk W. Pipeline vectorization. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 2001, 20(2):234−248.

[4] Frigo J, Gokhale M, Lavenier D. Evaluation of the streams-C C-to-FPGA compiler: An applications perspective. In: Proc. of the ACM/SIGDA Int'l Symp. on Field Programmable Gate Array. Monterey: ACM Press, 2001. 134−140.

[5] Guo Z, Najjar W. A compiler intermediate representation for reconfigurable fabrics. In: Proc. of the Conf. on Field Programmable Logic and Applications (FPL 2006). Madrid: IEEE Press, 2006. 1−4.

[6] Liu ZP, Bian JN, Zhou Q. An overview of power optimization in high level synthesis. Journal of Computer-Aided Design & Computer Graphics, 2007,11(19):1373−1380 (in Chinese with English abstract).

[7] So B, Hall M, Diniz P. A compiler approach to fast design space exploration in FPGA-based systems. In: Proc. of the 2002 ACM SIGPLAN Conf. on Programming Language Design and Implementation. Berlin: ACM Press, 2002. 165−176.

[8] Dong YZ, Dou Y. A parameterized architecture model in high level synthesis for image processing. In: Proc. of the 12th Conf. on Asia South Pacific Design Automation. Yokohama: IEEE Press, 2007. 523−528.

[9] Dong YZ, Dou Y, Zhou J. Optimized generation of memory structure in compiling window operations onto reconfigurable hardware. In: Proc. of the 3rd Int'l Workshop of Reconfigurable Computing Architecture, Tools and Applications. Mangaratiba: IEEE Press, 2007. 110−121.

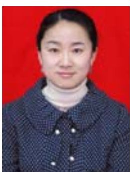
[10] Diniz P, Park J. Automatic synthesis of data storage and control structures for FPGA-based computing engines. In: Proc. of the 8th IEEE Symp. on Field-Programmable Custom Computing Machines. Napa Valley: IEEE Press, 2000. 91−100.

附中文参考文献:

[6] 刘志鹏,边计年,周强.高层次综合中面向功耗优化的方法与技术.计算机辅助设计与图形学学报,2007,11(19):1373−1380.

窦勇(1966－),男,吉林省吉林人,博士,教授,博士生导师,CCF 高级会员,主要研究领域为高性能计算机体系结构,可重构计算.

徐进辉(1978－),男,博士生,主要研究领域为可重构计算.



董亚卓(1979－),女,博士,主要研究领域为高级综合技术.

邬贵明(1981－),男,博士生,主要研究领域为高性能计算机体系结构.