

# 滑动窗口应用循环展开及其数据通路生成

董亚卓<sup>1)</sup> 刘明政<sup>2)</sup> 夏 飞<sup>1)</sup> 窦 勇<sup>1)</sup>

<sup>1)</sup>(国防科学技术大学计算机学院 长沙 410073)

<sup>2)</sup>(总后勤研究所 北京 100071)

**摘 要** 滑动窗口广泛应用于图像处理、模式识别和数字信号处理中,它具有数据量大、计算密集等特点,可重构硬件为滑动窗口应用提供了一个灵活高效的实现平台.文中基于一种存储、数据调度模型及其相应的数据通路生成技术,研究循环展开对滑动窗口应用的面积、时钟频率和吞吐率的影响.实验结果表明内层循环展开相对于外层循环展开将带来更大的控制复杂度,增加了对芯片面积的需求,然而外层循环展开需要更多的存储资源保存重用数据;当片内存储模块个数增加到一定规模时,时钟频率将随着循环展开不断降低;不同维度的应用,吞吐率随循环展开提升程度不同.

**关键词** 滑动窗口操作;高级综合;循环展开;数据通路;存储结构  
**中图法分类号** TP302

## Loop Unrolling and Data-Path Generation of Sliding-Window Operation

DONG Ya-Zhuo<sup>1)</sup> LIU Ming-Zheng<sup>2)</sup> XIA Fei<sup>1)</sup> DOU Yong<sup>1)</sup>

<sup>1)</sup>(School of Computer, National University of Defense Technology, Changsha 410073)

<sup>2)</sup>(Logistics Science Research Institute, Beijing 100071)

**Abstract** Window operations which are computationally intensive and data intensive are frequently used in image compression, pattern recognition and digital signal processing. Reconfigurable hardware boards provide a convenient and flexible solution to speed up these algorithms. Based on a memory and data schedule method as well as the method of data-path generation, this paper studies the effect of loop unrolling on the area, clock speed and throughput for sliding window operations. The results indicate that due to the unique design of the compilation framework, inner loop unrolling makes the controllers become more complicated than outer loop unrolling and increase more requirements of areas at the same time. However, outer loop unrolling demands more memory elements to keep the reused data. The clock speed begins to decrease when the number of RAM modules extends to a certain size, and the throughput increase in different degrees for different operations.

**Keywords** sliding-window operations; high level synthesis; loop unrolling; data-path; memory architecture

## 1 引 言

在数字化、信息化的时代,数字集成电路应用得

非常广泛,可编程逻辑器件随着制造工艺的发展取得了长足的进步,到今天已经发展成为可以完成超大规模逻辑设计的复杂组合逻辑与时序逻辑的现场可编程逻辑器件(FPGA),并在此基础上发展起来

收稿日期:2007-11-26;最终修改稿收到日期:2008-03-13. 本课题得到国家自然科学基金重点项目(60633050)资助. 董亚卓,女,1979年生,博士研究生,研究方向为高级综合技术. E-mail: dongyazhuo@nudt.edu.cn. 刘明政,男,1981年生,硕士,研究方向为模式识别技术. 夏 飞,男,1980年生,博士研究生,研究方向为高性能计算机体系结构. 窦 勇,男,1966年生,博士,研究员,博士生导师,研究领域为可重构计算和高性能计算机体系结构.

了实时电路重构技术. 实时电路重构技术的出现使过去传统意义上硬件和软件的界限变得模糊, 让硬件系统软件化, 其本质是利用可编程器件可多次重复配置逻辑状态的特性, 在运行时根据需要动态改变系统的电路结构, 从而使系统兼具灵活、简捷、硬件资源可复用、易于升级等多种优良性能. 基于此技术设计的可重构系统在高速数字滤波器、图像压缩、硬件演化计算、定制计算、嵌入式系统等方面, 都有着广泛的应用前景<sup>[1]</sup>.

随着问题规模的不断扩大, 传统的 RTL 级硬件设计方法表现出的设计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. 因此, 硬件高级综合技术成为系统级设计的前沿研究课题, 并涌现出大量的研究成果, 比较著名的有 System C, Handel\_C<sup>[2]</sup>, ASC<sup>[3]</sup>, SPC<sup>[4]</sup>, Stream-C<sup>[5]</sup>, ROCCC<sup>[6-7]</sup>等, 其中滑动窗口应用因为其数据量大、计算密集、顺序访存等特点, 成为高级综合中的一个研究热点.

本文面向滑动窗口应用, 提出了一种存储、数据调度模型及其相应的数据通路生成技术, 用于将滑动窗口应用自动映射到目标可重构硬件上, 并探讨了滑动窗口应用中的内层循环展开和外层循环展开

对面积、时钟频率和吞吐率的影响, 寻找这三个性能参数随循环展开的变化情况. 这一工作, 是本题在原有存储结构模型的基础上, 进一步进行有关设计空间探索研究的基础<sup>[8]</sup>. 所谓的设计空间探索, 也就是充分权衡源程序和片上系统提供的各种资源, 实现在满足性能要求和片上资源约束的前提下, 充分开发程序并行性, 增加资源利用率<sup>[9]</sup>.

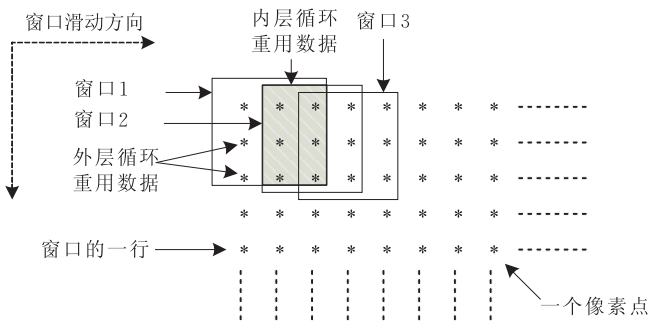
本文第 2 节简介滑动窗口应用程序; 第 3 节介绍相关工作; 第 4 节举例说明存储和数据调度模型; 第 5 节介绍相应的数据通路生成技术; 第 6 节将几个测试程序循环展开, 进行实验和性能评测, 得出相关结论; 最后为总结.

2 滑动窗口应用示例

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)<sup>[10]</sup>. 图 1(a) 中所示的 Sobel 边缘检测程序就是一个典型的滑动窗口应用, 图 1(b) 中给出了这个程序的滑动窗口示意图. 输入数组大小为  $SIZE \times SIZE$ , 滑动窗口大小为  $3 \times 3$ , 窗口滑动方向为从左到右, 从上到下.

```
char I[SIZE][SIZE], O[SIZE][SIZE];
int u1, u2, threshold;
for (i=0; i < SIZE; i++) {
    for (j=0; j < SIZE; j++) {
        u1 = I[i-1][j+1]+2*I[i][j+1]+I[i+1][j+1]-
            (I[i-1][j-1]+2*I[i][j-1]+I[i+1][j-1]);
        u2 = I[i+1][j-1]+2*I[i+1][j]+I[i+1][j+1]-
            (I[i-1][j-1]+2*I[i-1][j]+I[i-1][j+1]);
        if ((abs(u1))+abs(u2)) < threshold)
            O[i][j] = "0xff";
        else
            O[i][j] = "0x00";
    }
}
```

(a) 源程序



(b) 滑动窗口示意图

图 1 Sobel 边缘检测程序

Sobel 边缘检测程序中, 当  $j$  步进时, 需要更新 3 个窗口数据, 其余 6 个数据已经保存在窗口中, 存在内层循环数据重用; 当  $i$  步进时, 又要访问刚才访问过的  $i$  和  $i+1$  行的阵列数据, 存在外层循环数据重用. 在滑动窗口应用中, 存在着大量的数据重用. 充分开发数据重用, 是加快滑动窗口应用执行速度的关键.

3 相关研究

最原始的解决滑动窗口应用的方法没有数据重

用的概念, 需要每次从片外存储系统中调入全部窗口数据, 存在对片外存储系统中同一位置的多次访问, 而访问片外存储系统速度非常慢, 所以这种方法极大地降低了程序的执行速度.

Diniz 和 Park<sup>[11-12]</sup> 在高级综合技术中引入了数据重用方法, 将所有重用数据保存在寄存器组中, 以备后续使用, 这种方法使用了大量的寄存器资源, 而寄存器资源是片上系统中非常宝贵的资源.

Weinhardt 和 Luk<sup>[13]</sup> 将整行重用数据保存在片上存储系统中, 因为片上系统中的存储资源数量比寄存器资源的数量多得多, 所以, 这种方法可以保存

更多的重用数据,但是如果片上存储资源不够保存所有重用行,这种方法就失去了作用.

California 大学的 ROCCC<sup>[6-7]</sup>将窗口数据保存在寄存器组中并直接与片外存储系统相连,这种方法的缺点是只能开发内层循环数据重用,不能处理外层循环数据重用,其次,寄存器组与运算阵列的连接需要复杂的仲裁机制,导致通道连接复杂,硬件代价大.同时,这种体系结构模型也没有考虑如何进行设计空间开发.

Yu 等人<sup>[14]</sup>针对滑动窗口应用提出了一种层次化的存储结构组织方法 SWOOP. SWOOP 将所有重用数据保存在片内存储系统中,将窗口数据保存在寄存器中,并提出了相应的设计空间开发方法.当片上存储资源不够保存所有重用数据时,SWOOP 采用在存储系统中保存一个更大的窗口数据的方法,当前存储系统中保存的所有数据处理完毕后,再调入下一个大窗口中的数据. SWOOP 没有考虑到流水化的数据调度策略,所以这种方法带来了大量对片外存储系统的重复访问,同时 SWOOP

没有明确的体系结构模型做支撑.  
目前国内关于这方面的研究还处于起步阶段<sup>[8]</sup>.

4 存储和数据调度模型

4.1 Sobel 示例

图 2 给出了解决图 1(a)中的 Sobel 边缘检测程序的数据调度流图(令  $SIZE = 62$ )<sup>[15]</sup>. 设计 3 个 RAM 存储体,每个 RAM 存放输入阵列的一行,分别标识为 RAM1, RAM2 和 RAM3, SmartBuffer 是一个寄存器组,用于保存当前窗口数据.第  $i$  行处理完毕后,新输入的  $(i+2)$  行数据将替换原  $(i-1)$  行占据的 RAM 块,其余两个 RAM 中的数据不变,实现外层循环数据重用.每个时钟周期 SmartBuffer 中的数据平移,输入 3 个新操作数,淘汰 3 个旧操作数,同时控制开关调整 RAM 和 SmartBuffer 的连接通路,保证 SmartBuffer 与计算阵列的连接固定.这种体系结构充分利用数据重用,加快了程序的执行速度,并简化了硬件设计.

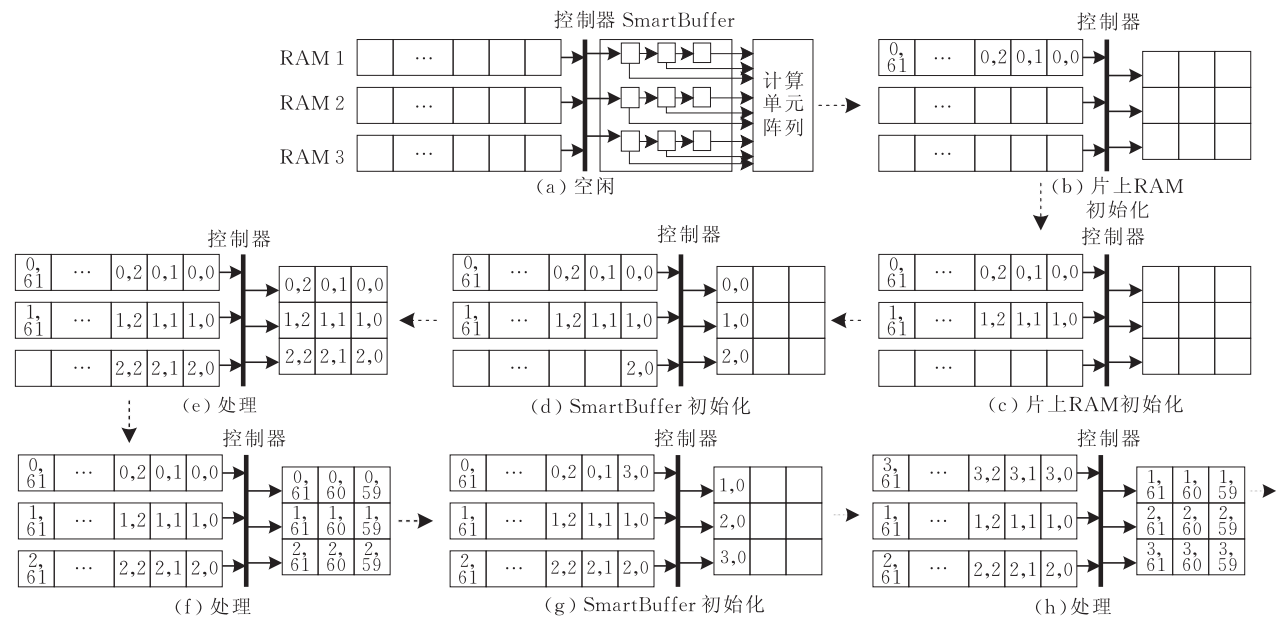


图 2 Sobel 边缘检测程序的数据调度流图

4.2 循环展开示例

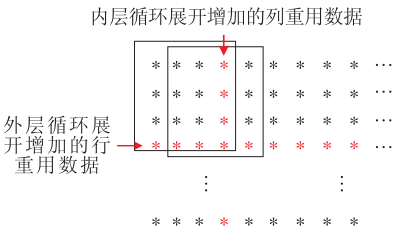
举例进一步说明循环展开后程序的数据存储方法和调度过程.图 3 所示为经过  $2 \times 2$  循环展开的 2D\_Lowpass\_filter 程序及其窗口操作示意图,每个时钟周期计算 4 个窗口,产生 4 个计算结果.

为保证计算流水线无空转的每个时钟周期产生

4 个计算结果,就要求 SmartBuffer 每个时钟周期更新 8 个窗口数据.假设片上存储资源足够保存所有重用数据,片上系统的数据带宽能够提供足够的数据供应速度,即每个时钟周期片外存储系统向片内存储系统输入 4 个数据,在这种情况下的理想的数据调度流如图 4 所示.

```
for(i=1;i<62;i=i+2){
  for(j=1;j<62;j=j+2){
    C[i-1][j-1]=(A[i-1][j-1]+A[i-1][j]+A[i-1][j+1]+A[i][j+1]+
      A[i+1][j-1]+A[i+1][j]+A[i+1][j+1])>>3+(A[i][j]>>1)-B[i-1][j-1];
    C[i-1][j]=(A[i-1][j]+A[i-1][j+1]+A[i-1][j+2]+A[i][j+2]+A[i+1][j]+
      A[i+1][j+1]+A[i+1][j+2])>>3+(A[i][j+1]>>1)-B[i-1][j];
    C[i][j-1]=(A[i][j-1]+A[i][j]+A[i][j+1]+A[i+1][j+1]+A[i+2][j-1]+
      A[i+2][j]+A[i+2][j+1])>>3+(A[i+1][j]>>1)-B[i][j-1];
    C[i][j]=(A[i][j]+A[i][j+1]+A[i][j+2]+A[i+1][j+2]+A[i+2][j]+
      A[i+2][j+1]+A[i+2][j+2])>>3+(A[i+1][j+1]>>1)-B[i][j];
  }
}
```

(a) 2\*2循环展开的2D\_Lowpass\_Filter源程序



(b) 滑动窗口示意图

图 3 2D\_Lowpass\_filter 程序 2×2 循环展开后的代码和滑动窗口示意图

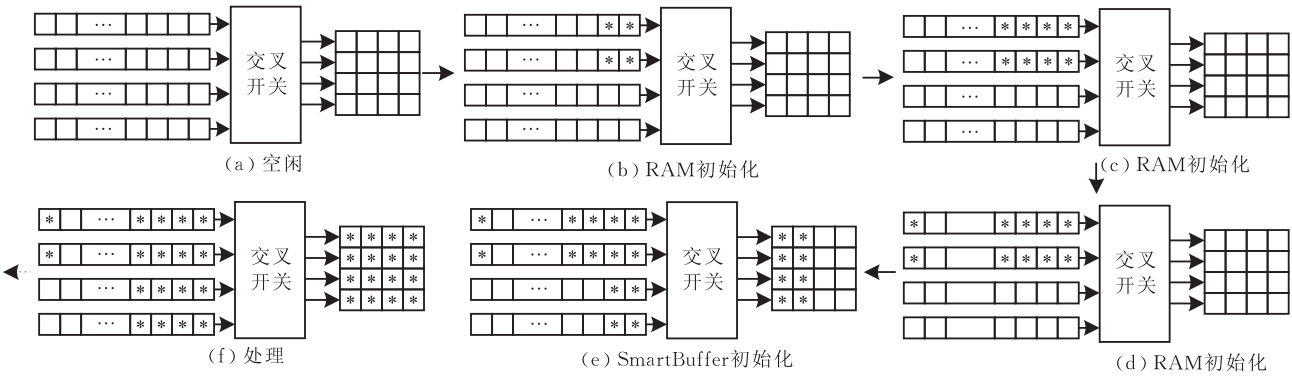


图 4 2×2 循环展开后的 2D\_Lowpass\_filter 程序的数据调度流图

5 数据通路生成

要将上述滑动窗口应用程序映射到目标可重构硬件上,需要解决的关键问题是控制状态机及其数据通路的自动生成.有关控制状态机的自动生成,在本课题的前面文章中有介绍<sup>[15]</sup>,本文将介绍运算流水线的数据通路生成技术.输入 C 语言描述的源程序,在 lance 编译器分析的基础上,输出循环体语句的数据通路抽象描述文件.

Lance 编译器主要实现了 4 个功能:源代码分析、中间表示生成、数据流图生成和执行机器无关的优化. Lance 编译器由 C 前端编译器,一系列优化工具和特定机器后端生成工具组成.在 lance 编译器上生成循环体表达式语句的数据通路抽象描述文件,分为 3 个步骤:

- (1) 生成三地址代码中间表示;
- (2) 定义信息库,保存循环控制和运算信息;
- (3) 输出循环体语句运算表达式的数据流图抽象描述文件.

5.1 三地址代码生成

在 C 源程序中定义每个函数,在 lance 编译器中对应的中间表示都包含了一个符号表和一个三地址代码段.三地址是指每个语句最多有两个源操作数,产生一个计算结果.中间表示表达式主要有符号、二元表达式、一元表达式、函数调用、强制类型转换、字符常量、整形常量、浮点常量八种类型.图 5 是一个 C 源程序经过 lance 前端编译器编译之后生成

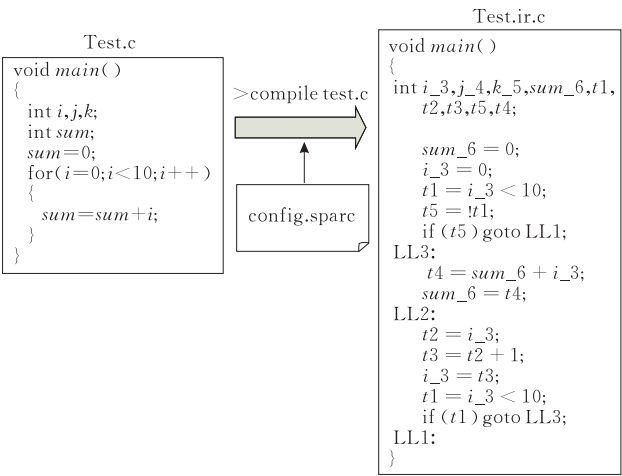


图 5 Lance 编译器生成的三地址代码示例

的中间表示。中间表示为源程序中的每个变量赋了一个标识号,并增加了一些新的中间变量。图 5 的三地址表示中,第一段是程序初始化部分,LL3 中是循环体中的运算语句,LL2 中是循环控制部分。

5.2 定义信息库

在三地址代码的基础上,定义信息库,保存循环程序的相关信息。图 6 是本文所定义的新的数据结构,为每个函数定义一个 Loop 信息链表,每个 Loop 包含了循环 ID 号、循环层数、循环的归纳变量表、归纳变量的初始值、终止值和步进值以及由循环体内语句所组成的表达式链表。表达式链表由多个表达

式树组成,每个表达式树又由多个节点组成。节点信息包括标识该节点的 ID 号、节点的类型、父节点的 ID 号、左右子节点的 ID 号、该节点所在的循环的 ID 号、所在表达式树的 ID 号。而节点又分为操作节点、数组节点和变量节点三类。每个数组节点会有一个数组信息包。数组信息包 ArrayNode 所记录的信息包括数组的编号、数组名、数组的维数、数组的下标表达式树的链表、此数组在这里是被定义还是被使用及数组的所在循环的编号和所在表达式树的编号,还有一个标识该数组是否在我们能处理的范围内的标识符。

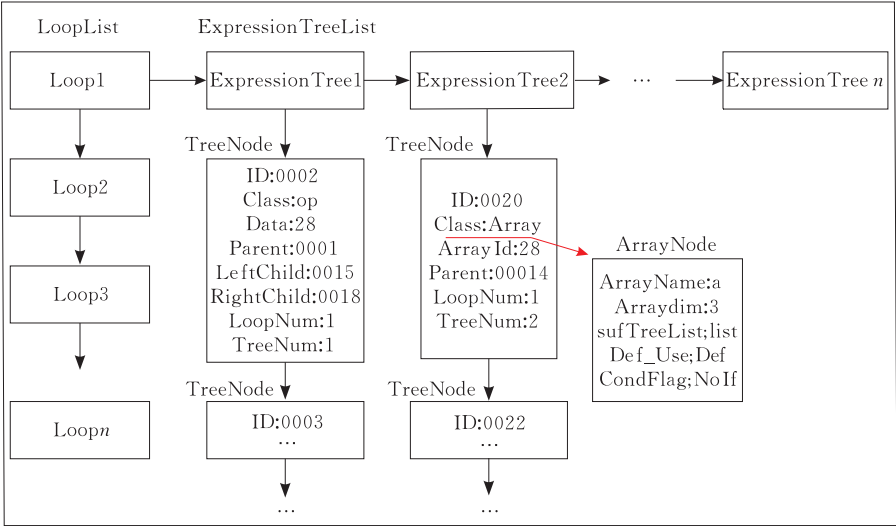


图 6 定义的数据结构概况

当找出了源程序中所有的循环后,需要对每个循环进行分析,构造循环的信息库。对于最内层循环的循环体中的每条语句都构造一棵表达式树,即构造一条表达式树链。如图 7 所示为循环体语句及其对应的表达式树。圆形节点代表操作,矩形节点代表操作数。如果是赋值语句,则树的根节点是赋值操作

节点。对于操作节点来说,该节点的左子节点是它的左操作数,右子节点是它的右操作数。操作数节点则只有父节点,没有左右子节点。每个数组操作数都有一棵其地址表达式的子树。这样就把所需要的信息存储在我们所定义的数据结构中,为下一步生成运算表达式树描述构造了一个通用的信息库。

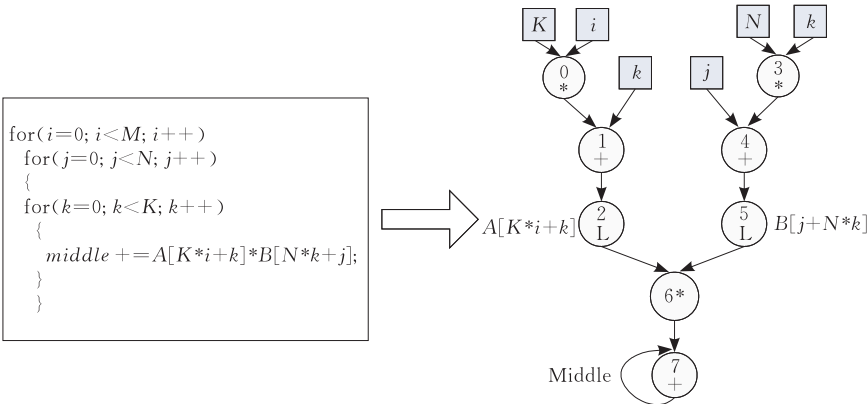


图 7 信息库表达式树示例

5.3 数据通路抽象描述

在定义的信息库的基础上,输出循环体运算表  
达式的数据通路抽象描述文件.图 8 中所示为图 1

(a)中 Sobel 边缘检测程序的部分运算表达式数据  
通路抽象描述结果.

# node_id op_name:		# edge_id dep_type distance src_node_id dest_node_id					
Nodes		operand_type:					
0	ADD	Edge					
1	MUL	0	FlowDepend	0	0	1	LeftOperand
2	ADD	1	FlowDepend	0	1	3	LeftOperand
3	ADD	2	FlowDepend	0	2	3	RightOperand
4	LOAD	3	FlowDepend	0	3	4	LeftOperand
5	MUL	4	FlowDepend	0	4	10	LeftOperand
6	ADD	5	FlowDepend	0	4	48	RightOperand
7	ADD	6	FlowDepend	0	5	7	LeftOperand
8	LOAD	7	FlowDepend	0	6	7	RightOperand
9	MUL	8	FlowDepend	0	7	8	LeftOperand
10	ADD	9	FlowDepend	0	8	9	RightOperand
-----		10	FlowDepend	0	9	10	RightOperand
-----		-----					

图 8 Sobel 程序的部分数据通路描述语句

数据通路抽象描述文件包括两个部分,一部分  
定义节点 ID 及其执行的操作,其中 LOAD 操作表  
示从存储系统中的某一地址取操作数,STORE 操  
作表示将运算结果保存到存储系统的某一地址中.另  
外一部分定义语句之间的相关和操作数传递关系,包  
括 6 个域:边 ID 号(edge\_id)、相关类型(dep\_type)、  
相关距离(distance)、源节点 ID 号(src\_node\_id)、  
目标节点 ID 号(dest\_node\_id)、操作数的类型  
(operand\_type).操作数的类型包括左操作数、右  
操作数和预测位.数据通路抽象描述文件标识出  
了循环体内运算语句的操作类型及其个数、运算  
部件之间的相关关系和中间结果传递关系.在此  
基础上,调用 Xilinx IP 库函数,生成底层 Verilog  
代码.

6 实验评测及其结论

这一小节将选择几个滑动窗口应用程序沿不同  
方向进行不同规模的循环展开,观察面积、吞吐率和  
时钟频率随循环展开的变化情况,得出结论并分析  
原因.本实验选择 5 个典型的滑动窗口应用:5-tap  
FIR(FIR5)、9-tap moving filter(FILTER9)、图  
像锐化 image sharp(SHARP)、Sobel 边缘检测  
(SOBEL)和 2D\_Lowpass\_filter.其中,FIR5 和  
FILTER9 是针对一维数组(1D)的滑动窗口操作,  
其余三个程序是针对二维数组(2D)的滑动窗口操  
作.假设一维数组的规模为 256,二维数组的规模为  
64×64,数据输入/输出宽度为 8bits.

我们将这 5 个测试程序及其循环展开后的程序  
在 Xilinx ISE 7.1 工具上综合,目标器件为 Xilinx

xc2s50e-5FPGA.将 1D 测试程序分别展开 2,4,8  
和 16 次(在图 9 中分别标识为 Un1,Un2,Un4,Un8  
和 Un16).图 9 中的  $Un_x \times y$  表示将 2D 测试程序  
的内层循环展开  $x$  次,外层循环展开  $y$  次.以  
SHARP 程序为例,循环展开  $4 \times 4$  次表示将源程序  
中的  $2 \times 2$  大小的窗口沿水平和垂直方向分别复制  
4 次,即每个时钟周期计算  $4 \times 4$  个窗口,产生  $4 \times 4$   
个计算结果.假设片上存储资源足够保存所有重用  
数据,存储带宽提供足够的数据传输速度,确保流水  
线无空转的正常执行.图 9 所示为 5 个测试程序的  
面积、吞吐率和时钟频率随循环展开的变化情况.其  
中吞吐率的计算方法为结果个数/执行节拍,执行节  
拍为 ModelSim5.8 的模拟结果.

6.1 面积和吞吐率

图 9 中给出的芯片面积(即 Slices 个数)结果同  
时包括控制状态机和计算阵列(数据通路)占用的面  
积.从该结果中可以看出,1D 测试程序从无循环展  
开到将循环展开 16 次,所占用芯片面积增加了  
13.4 倍,吞吐率提高了 10 倍.而 2D 循环程序从无  
循环展开到循环展开  $4 \times 4$  次,所占用芯片面积增加  
了 7.5 倍,而获得的吞吐率增加了近 15.5 倍.也就  
是说 2D 程序随循环展开获得了更大的吞吐率的提  
升.这是因为 2D 测试程序只需初始化前面几行数  
据到片内 RAM 中,即开始流水化计算过程.而 1D  
测试程序要初始化整个一维输入数组,数据初始化  
时间在总的执行时间中占相对较大的比重.而数据  
初始化时间并不随循环展开而改变,也就是说 1D  
测试程序中的不可加速部分占更多的比重,所以,  
1D 测试程序获得吞吐率的提升没有 2D 测试程序  
明显.



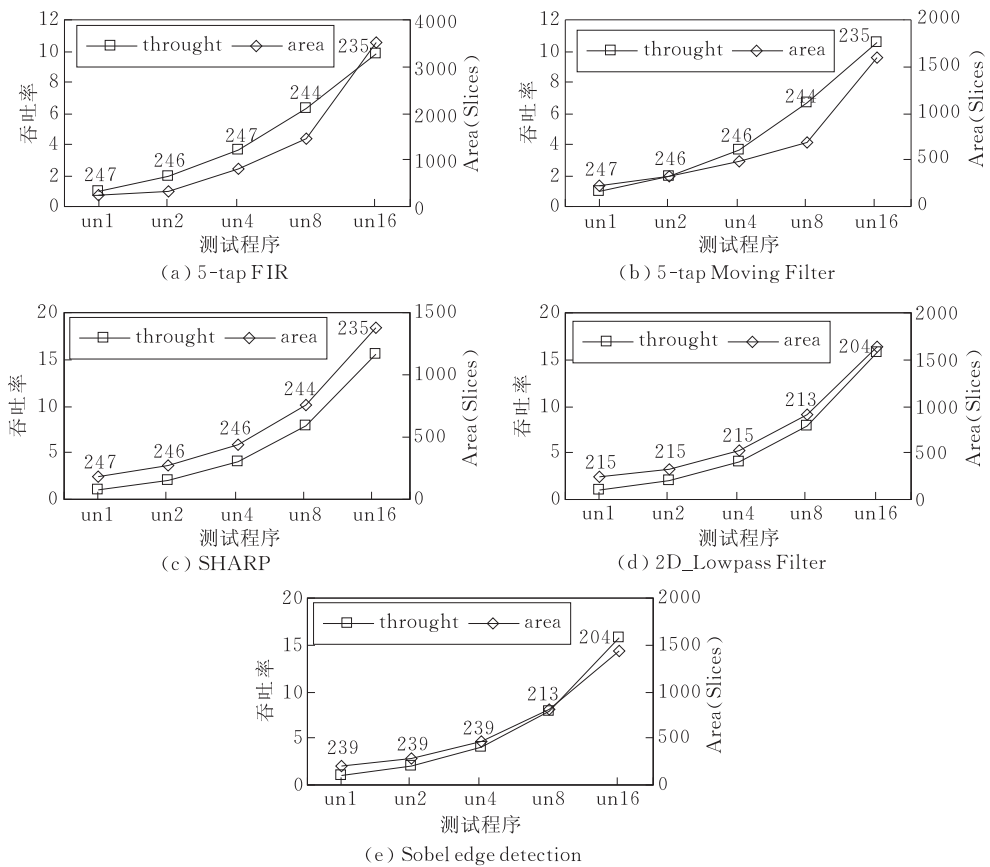


图 9 面积、吞吐率和时钟频率随循环展开的变化情况

为进一步研究循环展开对控制状态机和运算阵列的影响,我们将控制逻辑和计算流水线模块分别在目标器件上综合,分别得到运算阵列和控制逻辑随循环展开的变化情况,综合结果如表 1 所示,同时表 1 中也给出了各个程序的 SmartBuffer 大小和存储资源使用数量。

表 1 测试程序和实验结果

测试程序	循环展开次数	SmartBuffer 大小	使用的存储资源数量/B	运算阵列		控制状态机		使用 Slices 总数
				Slices	%	Slices	%	
5-tap FIR	Un1	5	512	166	63	96	37	262
	Un2	6	512	228	68	108	32	336
	Un4	8	512	228	84	129	16	789
	Un8	12	512	1321	90	151	10	1472
	Un16	20	512	3314	94	204	6	3518
9-tap Moving Filter	Un1	9	512	79	35	145	65	224
	Un2	10	512	155	49	160	51	315
	Un4	12	512	310	63	181	37	491
	Un8	16	512	626	73	236	27	682
	Un16	24	512	1254	78	351	22	1605
SHARP	Un1×1	4	1024	69	38	115	62	184
	Un1×2	6	1536	136	51	133	49	269
	Un2×1	6	1024	136	47	152	53	288
	Un2×2	9	1536	270	63	162	37	432
	Un2×4	15	2560	548	73	204	27	752
	Un4×2	15	1536	548	70	232	30	780
	Un4×4	25	2560	1100	80	283	20	1383
Lowpass Filter	Un1×1	9	1536	84	36	141	64	235
	Un1×2	12	2048	174	53	152	47	326
	Un2×1	12	1536	174	51	170	49	344
	Un2×2	16	2048	348	67	174	33	522
	Un2×4	24	3072	681	75	224	25	905
	Un4×2	24	2048	681	74	241	26	922
	Un4×4	36	3072	1354	83	287	17	1641

(续 表)

测试程序	循环展开次数	SmartBuffer 大小	使用的存储 资源数量/B	运算阵列		控制状态机		使用 Slices 总数
				Slices	%	Slices	%	
Sobel	Un1×1	9	1536	73	35	137	65	210
	Un1×2	12	2048	142	49	148	51	290
	Un2×1	12	1536	142	47	163	53	305
	Un2×2	16	2048	288	63	168	37	456
	Un2×4	24	3072	580	73	220	27	800
	Un4×2	24	2048	580	71	238	29	818
	Un4×4	36	3072	1160	80	282	20	1442

从表 1 中可以看出,运算阵列占用的 Slices 个数基本上随循环展开呈线性增长,这种变化规律的原因比较显然,是因为循环程序每展开一次,计算窗口个数增加一倍,计算流水线单元复制一次.我们还发现,内层循环展开相对于外层循环展开在更大的程度上增加了控制单元的复杂度,如在 SHARP 测试程序中,Un2×1 循环展开中的控制单元所占用的 Slices 个数要多于 Un1×2 循环展开中的控制单元所使用的 Slices 个数,这是因为内层循环展开使由 RAM 模块向 SmartBuffer 送数据的交叉开关变得更加复杂,不再是一个 RAM 模块每个时钟周期送出一个数据,而是随着循环展开要同时送出多个数据,也就是说 RAM 模块的宽度不再是输入数据的宽度,而可能要根据需要将多个输入数据合并保存在 RAM 存储体中的一个位置上,存储逻辑和数据传输逻辑都将变得更加复杂.然而,从表 1 中可以看出,外层循环展开相对于内层循环展开需要更多的存储单元保存重用数据,也就是说内层循环展开以增加复杂性和降低时钟频率为代价,而外层循环展开以增加存储需求为代价,根据这一规则,在今后本课题进行的有关设计空间探索研究中,我们可以做如下考虑:如果片上系统中有足够的 Slices 单元而存储资源有限,应优先考虑内层循环展开,而如果存储资源足够多,为获得更高的时钟频率,应优先考虑外层循环展开.

6.2 时钟频率

一个电路的时钟频率受到很多因素的影响,通常情况下,设计规模越小,越容易获得高频率的设计.图 9 中每个点上的值标识该程序的执行频率.从图 9 中我们可以看出,1D 测试程序的执行频率随着循环展开没有很大变化,而对 2D 测试程序,当循环展开到一定规模,时钟频率开始明显降低.这是因为,1D 测试程序的目标体系结构中只设计了一个存储体,控制复杂度增加不明显,还没有成为性能的瓶颈.而 2D 测试程序,随着循环展开,RAM 个数及宽度增加的更快,控制复杂度的增加一旦成为性能的

瓶颈,时钟频率就将随着循环展开不断下降.可见,循环展开一方面增加了控制复杂度,降低了时钟频率,另一方面提高了程序并行性,增加了吞吐率,在实际设计中,我们要根据设计目标调整循环展开的规模 and 方向,在时钟频率和吞吐率上实现合理的权衡.

7 总结和未来工作

本文基于一种存储、数据调度模型及其相应的数据通路生成技术,研究了滑动窗口应用在可重构片上系统中映射时,面积、吞吐率和时钟频率随循环展开的变化情况,实验结果表明内层循环展开相对于外层循环展开带来更多的控制复杂度,增加了对芯片面积的需求,然而外层循环展开需要更多的存储资源保存重用数据;当片内存储模块个数增加到一定规模时,时钟频率将随着循环展开不断降低;2D 滑动窗口应用相对于 1D 滑动窗口应用随着循环展开能够获得更高的吞吐率提升.这一分析是我们在原有工作的基础上,进一步进行有关设计空间探索研究的基础,即权衡源程序和目标器件,充分利用片上系统提供的各种资源,开发程序并行,加快程序的执行速度.

参 考 文 献

[1] Wang Hai-Li, Bian Ji-Nian, Wu Qiang, Xiong Zhi-Hui. SoC system-level design methods and techniques. Journal of Computer-Aided Design & Computer Graphics, 2006, 18(11): 45-54(in Chinese)  
(王海力,边计年,吴强,熊志辉. SoC 系统级设计方法与技术. 计算机辅助设计与图形学学报, 2006, 18(11): 45-54)

[2] Celoxica. Handel-C language reference manual for DK2. 0. document RM-1003-4. 0, 2003. 11

[3] Mencer O, Pearce D J, Howes L W, Luk W. Design space exploration with a stream compiler//Proceedings of the 2003 IEEE International Conference on Field-Programmable Technology, 2003: 270-277

[4] Weinhardt M, Luk W. Pipeline vectorization. IEEE Trans-



- actions on Computer-Aided Design of Integrated Circuits and Systems, 2001, 20(2): 234-248
- [5] Frigo J, Gokhale M, Lavenier D. Evaluation of the Streams-C-to-FPGA compiler: An applications perspective//Proceedings of the 9th ACM/SIGDA International Symposium on Field Programmable Gate Arrays(FPGA). Monterey, CA, 2001: 134-140
- [6] Schleupen K, Lekuch S, Mannion R, Guo Z, Najjar W, Vahid F. Dynamic partial FPGA reconfiguration in a prototype microprocessor system//Proceedings of the International Conference on Field Programmable Logic (FPL). Amsterdam, The Netherlands, 2007: 533-536
- [7] Guo Z, Najjar W. A compiler intermediate representation for reconfigurable fabrics//Proceedings of the International Conference on Field Programmable Logic and Applications (FPL 2006). Madrid, Spain, 2006: 1-4
- [8] Liu Ming-Ye, Zhang Xiao-Dong, Xu Qing-Ping. Technology decision for some key problems in VHDL high-level synthesis. Chinese Journal of Computers, 1997, 20(6): 501-509(in Chinese)  
(刘明业, 张东晓, 许庆平. VHDL 高级综合系统设计中某些关键技术的技术决策. 计算机学报, 1997, 20(6): 501-509)
- [9] So Byoungro, Hall Mary W, Diniz Pedro C. A compiler approach to fast hardware design space exploration in FPGA-based systems//Proceedings of the International Conference on Programming Language Design and Implementation (PLDI). Berlin, Germany, 2002: 165-176
- [10] Cesar Torres-Huttil, Miguel Arias-Estrada. FPGA-based configurable systolic architecture for window-based image processing. EURASIP Journal on Applied Signal Processing 2005, (7): 1024-1034
- [11] Park Joonseok, Diniz Pedro C, Shesha Shayee K R. Performance and area modeling of complete FPGA designs in the presence of loop transformations. IEEE Transactions on Computers, 2004, 53(11): 1420-1435
- [12] Diniz Pedro C, Park Joonseok. Automatic synthesis of data storage and control structures for FPGA-based computing engines//Proceedings of the FCCM 2000. Napa Valley, CA, 2000: 91-100
- [13] Weinhardt M, Luk W. Memory access optimization for reconfigurable systems. IEE Proceedings Computers and Digital Techniques, 2001, 148(3): 105-112.
- [14] Yu Hai-Qian, Leeser Miriam. Automatic sliding window operation optimization for FPGA-based//Proceedings of the FCCM 2006. San Francisco, CA, 2006: 76-88
- [15] Dong Ya-Zhuo, Dou Yong. A parameterized architecture model in high level synthesis for image processing//Proceedings of the ASP-DAC 2007. Yokohama, Japan, 2007: 523-528



**DONG Ya-Zhuo**, born in 1979, Ph. D. candidate. Her research interests include reconfigurable computing and high level synthesis.

**LIU Ming-Zheng**, born in 1981, M. S.. His research interests include image and signal processing.

**XIA Fei**, born in 1980, Ph. D. candidate. His research interests include reconfigurable computing and hardware acceleration.

**DOU Yong**, born in 1966, Ph. D., professor, Ph. D. supervisor. His research interests include reconfigurable computing and high-efficiency computer architecture.

## Background

This paper is mainly supported by the National Natural Science Foundation of China (60633050). This project aims to deal with the urgent science computation requirements of our country, research the pivotal technologies of high-efficiency parallel computer architecture. Based on the analysis of some typical applications, a series of studies are carried through, including memory architecture, hardware acceleration and high level synthesis etc. More than 20 papers of the team are published in international conferences and journals including FPGA2005, ASP-DAC2007 and ASAP2007 etc.

The authors' work is used in high level synthesis. High Level Synthesis (HLS) tools provide a bridge between the algorithm written in a high level language (Matlab, C, C++, etc) and a lower level Hardware Description Language (HDL). They concentrates on one class of applications called window operations. This kind of applications is widely used in signal, image and video processing and requires much com-

putation and data manipulation. Reconfigurable hardware boards provide a convenient and flexible solution to speed up these algorithms.

High level synthesis is increasingly recognized to be the key to reducing the complexity of hardware design. However, the memory structure has become the performance bottleneck. This paper presents a parameterized memory architecture for high level synthesis to automatically generate the hardware frames for all window processing applications, gives a design for three levels memory structure to realize inner-loop and outer-loop data reuse completely, and at the same time uses shifted registers to make hardware design simpler. Based on the memory and data schedule method as well as the method of data-path generation, this paper studies the effect of loop unrolling on the area, clock speed and throughput for sliding window operations.