文章编号:1001 - 2486(2011)02 - 0140 - 05

面向大尺寸滑动窗口应用的并行计算模型*

庞征斌[1],徐金波[1],董亚卓[2],窦 勇[1],张 峻[1]
(1. 国防科技大学 计算机学院,湖南 长沙 410073; 2. 中国人民解放军 91655 部队,北京 100036)

摘 要:大尺寸滑动窗口的应用在数据输入速度与处理速度之间存在较大差距。为了缩短差距,提出了一种并行计算模型,使用尽可能少的存储资源与尽可能简单的存储器读写控制逻辑实现了尽可能高的数据重用性与并行性。该模型将不同滑动窗口之间的并行处理与单个窗口内不同数据之间的并行处理结合起来:对于不同窗口,按列进行分组并映射到多个处理单元上并行处理;对于单个窗口内的数据,使用多体存储结构进行缓存,并设计了存储体分配机制与寻址函数以实现多个数据的无冲突并行访问。在 FPGA 上的实验结果表明:提出的计算模型在没有明显增加存储资源使用代价与读写控制逻辑复杂性的情况下大大提高了处理速度。

关键词:并行处理;无冲突并行访问;滑动窗口
中图分类号:TP302;TP391.41 **文献标识码**:A

A Parallel Processing Scheme for Large-size Sliding-window Applications

PANG Zheng-bin[1], *XU Jin-bo*[1], *DONG Ya-zhuo*[2], *DOU Yong*[1], *ZHANG Jun*[1]
(1. College of Computer, National Univ. of Defense Technology, Changsha 410073, China;
2. Unit 91655 People's Liberation Army, Beijing 100036, China)

Abstract: There exists a large gap between the data input speed and processing speed in large-size sliding-window applications. To shorten this gap, a parallel processing scheme was proposed, which achieved high data reusability and parallelism with memory resources as few as possible and memory access control logics as simple as possible. The scheme combined the advantages of parallelism among different sliding-windows and parallelism among different data in a single window. For different windows, they were divided into groups and mapped into multiple processing elements. For the data in a single window, multi-module memory structure was introduced to buffer them, where module assignment and addressing scheme was designed for conflict-free parallel access. Experimental results on FPGA show that this approach can improve the processing speed significantly without incurring too much memory resources and too complicated memory access control logics.

Key words: parallel processing; conflict-free parallel access; sliding-window

运算部件的数据消耗速度只有与存储系统的数据供应速度保持均衡,才能实现效率最大化。滑动窗口操作具有数据量大、计算密集等特点,被处理数组的输入速度通常比数据处理速度快得多。窗口在被处理数组上最多需要$(M-m+1)\times(N-n+1)$次滑动,每次滑动需要处理$m\times n$个数据(其中,M, N, m 和 n 分别表示被处理数组和滑动窗口的高度和宽度),如果运算部件处理一个数据的延迟为 1 个时钟周期,完成整个数组的遍历扫描操作延迟为$m\times n\times(M-m+1)\times(N-n+1)$个时钟周期,而存储系统若以 1 pixel/cycle 的速度输入数据仅需要约 $M\times N$ 个时钟周期,这就出现了运算部件的数据消耗速度与存储系统的数据供应速度不均衡的现象。当 $m\times n$ 较大时,这种不均衡现象更为严重。

目前已经存在很多关于滑动窗口操作的相关研究,但是这些工作大多只关注于存储系统的存取效率,很少有工作研究存储系统与运算部件协同工作的速度均衡问题。为缓解访存问题,人们提出了数据重用[1-2]的概念。目前普遍使用的方法是将重用数据保存在一个由寄存器组成的 smart buffer 中,以备后续使用[3]。按照运算部件的数据处理模式对片内存储系统进行无冲突并行

* 收稿日期:2010 - 07 - 15
基金项目:国家 863 高技术计划项目(2008AA01A202,2008AA01A201);国家自然科学基金资助项目(60903057,60833004)
作者简介:庞征斌(1972—),男,副研究员,博士。

访问是一种有效的降低存储器瓶颈效应的方案(例如,文献[4]从体系结构设计的角度提出了对数组中任意位置矩形数据块进行并行访存的想法)。通过在片外存储器和运算部件之间加入片内多体存储结构,并使用针对特定数据处理模式而预设的固定存储体分配函数和寻址函数,可以大大提高数据访问效率。人们提出了各种方法来实现数据在多体存储器中有效的分配与映射[5]。

对于大尺寸滑动窗口应用,现有的多体存储结构会因为存储体个数较多而导致读写控制逻辑复杂性过大,从而影响系统性能。为了使用尽可能少的存储体实现尽可能高的并行程度,本文针对大尺寸滑动窗口应用提出一种并行计算模型,目的是实现运算部件的数据消耗速度和存储系统的数据供应速度之间的均衡。这种计算模型使用片内存储系统对数据进行缓存,实现了数据重用,减少了片外存储器与运算部件之间的访存次数;为了使得由片外存储器输入的数据能够被运算部件等速地消耗,本文使用高度并行与流水化的运算部件实现数据的并行处理;为了满足运算部件的数据消耗速度,提出了一种组合式多体存储结构,这种结构通过将不同窗口之间的并行处理与单个窗口内不同数据之间的并行处理结合起来,大大减少了当滑动窗口尺寸较大时的访存读写控制逻辑复杂性。

1　面向大尺寸滑动窗口应用的并行计算模型

令 W 表示片外存储器到片内存储器的输入带宽(pixels/cycle),P 表示运算部件中处理单元(Processing Element,PE)的个数,$a \times b$ 表示每个 PE 可并行处理的子窗口大小,S 表示 P 个 PE 的片上存储资源需求(字节),T 表示片上的可用存储资源(字节),那么从片外存储器向片内存储器输入 $M \times N$ 数组所需的时钟周期数可表示为 $M \times N/W$。要实现运算部件的数据消耗速度和存储系统的数据供应速度之间的均衡,执行 $(M-m+1)\times(N-n+1)\times m \times n$ 次运算操作所需要的理想时间为 $M \times N/W$ 个时钟周期,此时运算部件的并行度为 $[(M-m+1)\times(N-n+1)\times m \times n \times W]/(M \times N)$。对于滑动窗口应用,可以通过不同窗口间与窗口内不同数据间两个层次上的并行处理来实现这种均衡。

1.1　不同窗口间的并行处理

可以通过配置多个片内存储体和处理单元 PE 同时对不同窗口进行操作而实现并行处理。如图 1 所示配置 P 个 PE,将 $(M-m+1)\times(N-n+1)$ 大小的数组按列划分为 P 组,PE_k 负责从第 $\lceil \frac{N-n+1}{P} \rceil \times k$ 列到第 $\lceil \frac{N-n+1}{P} \rceil \times (k+1)-1$ 列的平移匹配,所需要的数据为从第 $\lceil \frac{N-n+1}{P} \rceil \times k$ 列到第 $\lceil \frac{N-n+1}{P} \rceil \times (k+1)-1+n$ 列的元素,每个 PE 所需的数据保存在各自的内部存储器中。

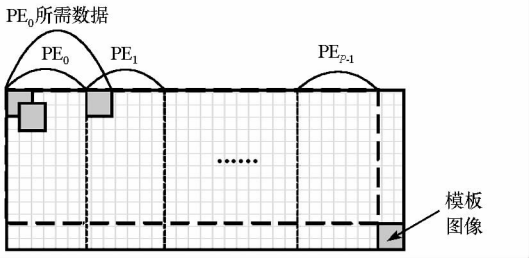

图 1　不同窗口间的并行处理

Fig.1　Parallel processing among different windows

对于这种方式,其并行度为 P,理论上 P 最大值可达到 $(N-n+1)$,也就是每个 PE 只负责一列滑动窗口的计算。但是,这种方式受到存储资源的限制,原因是不同的 PE 中内部存储器所保存的数据存在数据重叠,如图 1 所示,每个 PE 需要额外存储 n 列数据,P 越大,PE 内部存储器的使用效率越低。由于滑动窗口应用中每个数据的处理通常只需要其 $m \times n$ 邻域内的数据,因此局部性较好,通常每个 PE 内部存储器只需保存 $m+1$ 行数据即可保证每个数据在被运算部件使用时都能在内部存储器中找到。根据以上分析,P 个 PE 的内部存储器需求可近似表示为 $S=[(N-n+1)+n\times P]\times(m+1)$(假定每个数据大小为 1 字节)。受到片上存储资源的限制,S 必须不大于片上的可用存储资源 T,即 $P \leqslant \frac{T-(N-n+1)\times(m+1)}{(m+1)\times n}$。因此,这种方式所能够获得的并行度很大程度上受到存储资源的限制。

1.2　窗口内不同数据的并行处理

对于滑动窗口应用的每次滑动,若希望实现对当前窗口内数据的并行操作,就需要对片内存储器中的这些数据进行无冲突并行访问。运算部件每个时钟周期从片内存储器中并行读取一个 $a \times b(a \leqslant m, b \leqslant n)$ 子窗口进行处理,这种并行计算方式处理一个 $m \times n$ 窗口的延迟为 $\lceil \frac{m}{a} \rceil \times \lceil \frac{n}{b} \rceil$。

这种方式的并行度为 $a \times b$,理论上 $a \times b$ 可以达到 $m \times n$,即运算单元可以每个时钟周期完

成一个滑动窗口的处理。但是,要实现从片内存储器中无冲突地并行读取 $a \times b$ 数据块,必须采用多体存储器结构,且存储体的个数必须不少于 $a \times b$。而太多的存储体将使得访存操作的读写控制逻辑复杂性大大增加,且较大的 $a \times b$ 也会大大增加运算单元的流水线级数,因此,$a \times b$ 不宜太大,从而导致这种方式所能够获得的并行度也是受限的。

1.3 基于组合式多体存储结构的并行处理

可以看出,以上两种方式在滑动窗口尺寸较大,即 $m \times n$ 较大时,所能实现的并行度都无法达到较理想的水平。为了进一步提高并行度,本节提出了面向大尺寸滑动窗口应用的并行计算模型,该模型将以上两种方式充分结合起来,削弱了硬件资源不足对并行度的限制,减小了访存读写控制逻辑复杂性。图 2 给出了这种并行计算模型的框架结构与数据访问模式示意图。

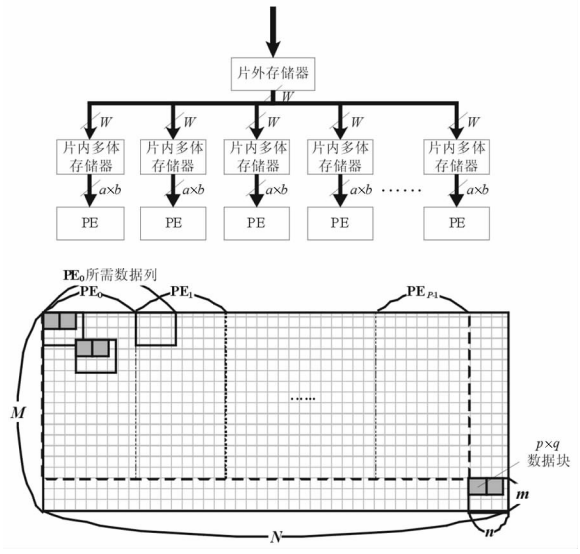

图 2 本文的并行计算模型框架结构与数据访问模式示意图

Fig.2 Structure of the proposed parallel processing scheme and illustration of the data access pattern

片外存储器与片内存储器之间的输入带宽为 W pixel/cycle,$M \times N$ 数组首先保存在片外存储器中,然后线性输入 P 个 PE 所对应的片内存储器中,PE_k 对应存储器中只保存从第$\lceil \frac{N-n+1}{P} \rceil \times k$ 列到第$\lceil \frac{N-n+1}{P} \rceil \times (k+1) - 1 + n$ 列的元素;每个 PE 从内部存储器中并行读取 $a \times b$ 数据块进行处理,每个 $m \times n$ 扫描窗口的处理延迟为$\lceil \frac{m}{a} \rceil \times \lceil \frac{n}{b} \rceil$个时钟周期。要实现对数组中任意位置 $a \times b$ 数据块的无冲突并行访问,比较有效的方式是采用多体存储方式。在每个 PE 的内部存储器中,配置了以二维矩阵形式组织的 $a \times b$ 个存储体,并使用合理的存储体映射策略与寻址策略以实现无冲突并行访问,本文将在后续内容中对多体存储结构具体的映射策略和寻址策略进行详细介绍。

采用本文提出的并行计算模型,完成 $M \times N$ 数组上 $m \times n$ 扫描操作的延迟将为$(M - m + 1) \times \lceil \frac{N-n+1}{P} \rceil \times \lceil \frac{m}{a} \rceil \times \lceil \frac{n}{b} \rceil$个时钟周期。当该延迟与从片外存储器向片内存储器输入 $M \times N$ 数组所需时钟周期数 $M \times N/W$ 相当时,即达到了运算部件的数据消耗速度和存储系统的数据供应速度之间的均衡。举例说明,假设 $M = N = 1024$ 像素,$m = n = 64$ 像素,$W = 1$ 像素,那么均衡条件为 $961 \times \lceil \frac{961}{P} \rceil \times \lceil \frac{64}{a} \rceil \times \lceil \frac{64}{b} \rceil = 1024 \times 1024$。如果只采用 1.1 节的方式,即 $a = b = 1$,那么 P 至少应为 3607,而事实上,即使不考虑存储资源的限制,P 最多也只能为 961;如果只采用 1.2 节的方式,即 $P = 1$,那么 $a \times b$ 至少应为 3607,而在每个 PE 中配置 3607 个存储体将使得读写控制电路异常复杂。而采用本节提出的基于组合式多体存储结构的并行计算模型,P、a 和 b 的数值均可大幅减小,从而降低片内存储资源需求与每个 PE 所对应内部存储器结构的读写控制逻辑。比如,$a = b = 8$ 且 $P = 56$ 时,即可以实现运算单元的数据消耗速度和存储系统的数据供应速度之间的均衡。

2 单个 PE 中多体存储器的设计实现

在本文提出的并行计算模型中,每个 PE 所对应的多体存储器的设计是其中的重点与难点。本节给出了数据在多体存储器中的存储体分配机制与寻址机制,能够对当前 PE 所处理的任意位置 $m \times n$ 大小的滑动窗口中任意一个 $a \times b$ 大小的子窗口进行无冲突并行访问。

2.1 数据缓存策略、存储体分配机制与寻址机制

令 V 表示$\lceil \frac{N-n+1}{P} \rceil$,那么每个 PE 需处理的数组大小为 $M \times V$。由于滑动窗口应用是对待处理数组进行顺序扫描,数据从片外存储器向片内存储器也是顺序传输,因此每个 PE 的内部存储器中只需缓存当前一行滑动窗口操作所需的数据即可。因为 PE 处理数据的粒度为 $a \times b$,因此内部存储器缓存数据的大小应为$\lceil \frac{m}{a} \rceil \times a \times (V +$

$\lceil \frac{n}{b} \rceil \times b - 1)$。

对于当前 PE 所处理的 $M \times V$ 大小的数组,希望能够对位于任意位置(i, j)的 $a \times b$ 数据块 $B(i, j)$进行无冲突并行访问。运算单元根据下面所述的存储体分配机制与寻址机制从内部存储器中将 $a \times b$ 个数据从 $a \times b$ 个存储体中同时读出。

首先需要一个存储体分配函数 $m_{p,q}(i,j)$来描述 $M \times V$ 数组中位于(i, j)位置的数据与某个存储体$(p, q)(0 \leqslant p < a, 0 \leqslant q < b)$的映射关系。本文把 $m_{p,q}(i,j)$分割为两个正交的函数 $m_p(i)$和 $m_q(j)$,对于每个存储体(p, q):

$$\begin{aligned} m_p(i) &= (i-p) \bmod a, \\ m_q(j) &= (j-q) \bmod b \end{aligned} \tag{1}$$

另外,还需要一种寻址机制确定 $B(i, j)$在二级存储体中的偏移地址。首先假定每个 PE 的内部存储器可以缓存所有 M 行数据,使用寻址函数 $A_{p,q}(i,j)$来确定 $B(i, j)$中每个元素在某个存储体(p, q)中的地址:$A_{p,q}(i,j) = (i \mathrm{div} a + c_i) \cdot \lceil \frac{V + \lceil \frac{n}{b} \rceil \times b - 1}{b} \rceil + j \mathrm{div} b + c_j$,其中,$c_i = \begin{cases} 1, & i \bmod a > p \\ 0, & \text{otherwise} \end{cases}$,$c_j = \begin{cases} 1, & j \bmod b > q \\ 0, & \text{otherwise} \end{cases}$。实质上,$c_i$ 和 c_j 相当于内嵌到 $A_{p,q}(i,j)$中的存储体分配函数。但是由于每个 PE 的内部存储器只缓存 $\lceil \frac{m}{a} \rceil \times a$ 行数据,且存储空间循环使用,因此,$B(i, j)$的实际地址应稍作调整,令 $i \mathrm{div} a + c_i$ 模 $\lceil \frac{m}{a} \rceil$,即:

$$A_{p,q}(i,j) = \left((i \mathrm{div} a + c_i) \bmod \lceil \frac{m}{a} \rceil \right) \cdot \lceil \frac{V + \lceil \frac{n}{b} \rceil \times b - 1}{b} \rceil + j \mathrm{div} b + c_j \tag{2}$$

图 3 给出了一个例子,假定当前 PE 处理滑动窗口操作所需数据大小 $M \times (V + \lceil \frac{n}{b} \rceil \times b - 1) = 16 \times 16$,滑动窗口大小 $m \times n = 4 \times 8$,而 PE 能够并行处理的子窗口 $a \times b = 2 \times 4$,图中每个小的矩形框表示一个数据,三个数值依次是该像素的 $m_p(i)$、$m_q(j)$、$A_{p,q}(i,j)$。

考虑位于(3, 6)坐标的 2×4 数据块 $B(3,6)$,使用式(1)、(2)可以计算出其中每个像素在对应存储体中的地址:

(1)module$(p, q) = (0, 0)$:

$$\left.\begin{aligned} i \bmod a = 1 > p &\Rightarrow c_i = 1 \\ j \bmod b = 2 > q &\Rightarrow c_j = 1 \end{aligned}\right\} \Rightarrow A_{0,0}(3,6) = 2;$$

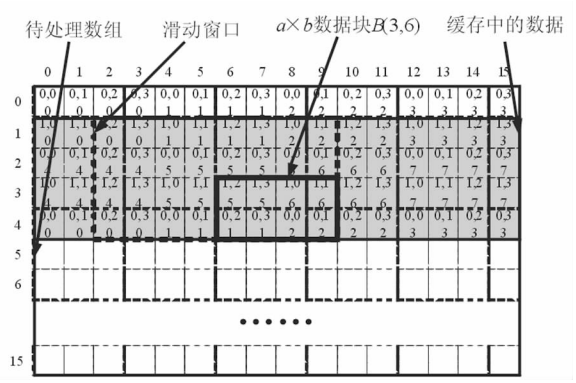

图 3　地址映射举例

Fig.3　Example of the address mapping function

(2)module$(p, q) = (1, 1)$:

$$\left.\begin{aligned} i \bmod a = 1 = p &\Rightarrow c_i = 0 \\ j \bmod b = 2 > q &\Rightarrow c_j = 1 \end{aligned}\right\} \Rightarrow A_{1,1}(3,6) = 6。$$

也就是说,$B(3,6)$中的数据将被映射到存储体(0,0)中的地址 2 上,以及存储体(1,1)中的地址 6 上。类似的,$A_{0,1}(3,6) = 2$, $A_{0,2}(3,6) = 1$, $A_{0,3}(3,6) = 1$, $A_{1,0}(3,6) = 6$, $A_{1,2}(3,6) = 5$, $A_{1,3}(3,6) = 5$。

2.2　多体存储器寻址结构

根据以上分析,本节设计了寻址机制的硬件结构,图 4 给出了以 $a \times b = 2 \times 4$ 为例的寻址结构。在访问一个 $a \times b$ 数据块 $B(i, j)$时,首先根据式(2)计算 $A_{p,q}(i,j)$,$A_{p,q}(i,j)$可以表示成 i 方向和 j 方向的两个函数的和,即 $A_{p,q}(i,j) = Ai_p(i) + Aj_q(j)$。这样,每个存储体的地址就可以通过行地址和列地址组合而成,避免了为每个存储体设计单独的地址计算电路。每个存储体从地址 $A_{p,q}(i,j)$处并行输出 $a \times b$ 个数据。这些数据首先被送入 b 个 a 路选择器进行排序,然后排序结果再被送入一个 $a \times b$ 路选择器进行排序,从而保证 $a \times b$ 个数据能够进入运算单元中的正确位置。为了控制这些选择器,首先使用函数 $R_i(i) = i \bmod a$ 和 $R_j(j) = j \bmod b$ 计算出 $B(i, j)$的左上角元素(i,j)所对应的存储体,进而判断出数

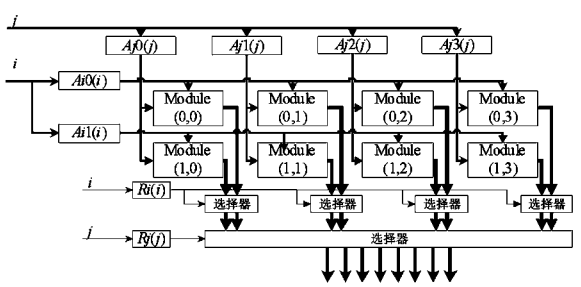

图 4　多体存储器寻址结构

Fig.4　Addressing structure of the multi-module memory

据与存储体之间的对应关系,从而控制多路选择器的路由。

3 实验结果与分析

基于图 2 所示的并行计算模型框架结构以及图 4 所示的单个 PE 中多体存储器的结构,在 FPGA 上以 7×7 中值滤波为例进行了实验(即滑动窗口大小 $m \times n$ 为 7×7)。该实验对不同的待处理数组大小 $M \times N$、不同的 PE 个数 P、不同的单 PE 存储体个数 $a \times b$ 的多种组合情况进行了实验与比较。FPGA 片外的存储器保存全部待处理数据,并以带宽 W = 8pixels/cycle 向 FPGA 输入数据。FPGA 器件采用 Altera Stratix Ⅱ EP2S130F1020C5,这种 FPGA 芯片包含 106 032 个 ALUT, 6 747 840 位的片上存储器。本文使用 Quartus Ⅱ和 Mentor Graphics ModelSim 作为开发工具。

表 1 给出了几种 $M \times N$、P、$a \times b$ 组合情况下的处理节拍数、片内存储资源使用情况及时钟频率等。从表中可以看出,采用多个 PE 和多体存储结构相结合的方式与只采用其中一种并行方式相比,在所需要的处理时间上具有很高的加速比,通过配置适当数量的 PE 和存储体,就可以实现数据输入速度与数据处理速度的均衡,避免出现瓶颈现象。例如,以表 1 中第 3 行的实验数据为例,以 8pixels/cycle 带宽输入 1024×1024 个数据所需要的时钟周期数约为 131 072,而使用 16 个 PE 和 4×8 多体存储结构进行数据处理所需要的时钟周期数为 133 059,当时钟频率为 128MHz 时,处理时间约为 1ms。另一方面,本文的这种计算结构并没有因为处理速度的大幅提高而增加较大的片内存储资源使用代价,这主要归功于每个 PE 中多体存储结构的设计,它在不增加存储资源的情况下可以大大提高数据重用性以及访存带宽。

表 1 处理节拍数、片内存储资源使用情况及时钟频率

Tab.1 Cycles, on-chip memory utilization and frequencies

$M \times N$	P	a×b	节拍数	片内存储资源使用(bit)	时钟频率(MHz)
1024×1024	1	8×8	1 092 037	66 488	138
1024×1024	16	8×8	132 311	73 704	110
1024×1024	16	4×8	133 059	73 416	128
1024×1024	16	2×4	551 603	73 304	140
1024×1024	16	1×1	3 211 548	63 512	148
512×512	1	4×4	1 079 256	33 496	147
512×512	16	4×4	66 743	40 568	134
512×512	16	1×1	809 925	34 824	153
128×128	1	4×4	60 621	8 688	149
128×128	8	4×4	8 254	3 328	139
128×128	8	1×1	100 719	10 200	156

4 结论

本文针对大尺寸滑动窗口应用提出了一种并行计算模型,使用尽可能少的存储资源与尽可能简单的存储器读写控制逻辑实现了尽可能高的数据重用性与并行性,从而削弱甚至消除了数据输入速度与数据处理速度之间的不均衡性。首先介绍了并行计算模型所采用的组合式多体存储结构的设计思想,然后详细介绍了多体存储器的存储体分配机制与寻址机制。实验结果表明本文的工作与传统的并行处理方式相比在处理速度上可以获得较高的加速比,同时并没有因为处理速度的大幅提高而增加较大的片内存储资源使用代价。

参考文献:

[1] Guo Z, Buyukkurt B, Najjar W, et al. Optimized Generation of Data-path from C Codes for FPGAs [C]//Proceedings of International ACM/IEEE Design, Automation and Test in Europe Conference, Munich, Germany, 2005(1): 112 – 117.

[2] So B, Hall M W, Ziegler H E. Custom Data Layout for Memory Parallelism[C]//Proceedings of the 2nd IEEE/ACM International Symposium on Code Generation and Optimization (CGO), San Jose, CA, USA, 2004: 291 – 302.

[3] Diniz P C, Park J. Automatic Synthesis of Data Storage and Control Structures for FPGA-based Computing Engines[C]//Proceedings of the 8th IEEE Symposium on Field-programmable Custom Computing Machines (FCCM), Napa Valley, CA, 2000: 91 – 100.

[4] Kuzmanov G, Vassiliadis S, Eijndhoven J. A 2D Addressing Mode for Multimedia Applications [C]//Workshop on System Architecture, Modeling, and Simulation (SAMOS 2001), Samos, Greece, 2001. New York, NY, USA: Springer-verlag, 2002: 291 – 306.

[5] Gou C Y, Kuzmanov G, Gaydadjiev G N. SAMS: Single-affiliation Multiple-stride Parallel Memory Scheme [C]//Proceedings of Workshop on Memory Access on Future Processors: A Solved Problem, 2008: 359 – 367.

[6] Wang C, Han Y P. Twin Butterfly High Throughput Parallel Architecture FFT Algorithm [C]//Proceedings of International Symposium on Information Science and Engineering, 2008(2): 637 – 640.

[7] Vitkovski A, Kuzmanov G, Gaydadjiev G. Memory Organization with Multi-pattern Parallel Accesses [C]//Proceedings of Design, Automation and Test in Europe, 2008: 1420 – 1425.

[8] Peng J Y, Yan X L, Li D X, et al. A Parallel Memory Architecture for Video Coding[J]. Journal of Zhejiang University: Science A, 2008 (9): 1644 – 1655.

[9] Reisis D, Vlassopoulos N. Conflict-free Parallel Memory Accessing Techniques for FFT Architectures [J]. IEEE Transactions on Circuits and Systems Ⅰ: Regular Papers, 2008(55): 3438 – 3447.