基于 SIMD PE 阵列的 DCT 数据并行实现方法研究

钟 升
(西安微电子技术研究所,陕西西安 710075)

摘 要: 本文为满足 G级像素帧的实时性处理需求,针对 DCT 变换计算量大和常规处理中并行度不足的问题,提出一种基于 SIMD PE 阵列的 DCT 数据并行实现方法. 该方法因 PE 阵列本身所具有的可裁减特性,可应用于不同并行度需求的嵌入式系统中. 文中提出一种基于 PE 标识的数据并行操作方式,不但解决了局部计算中的"PE 自治"问题,又省去了数据寻址时间开销. 该操作方式规则、简洁,满足 SIMD 操作规则性强的要求,符合并行处理技术的发展方向.

关键词: 并行处理; DCT; SIMD PE 阵列; 映射语言
中图分类号: TP393 **文献标识码**: A **文章编号**: 0372-2112 (2009) 07-1546-08

Research of Data-Parallel-Implementation Method of DCT Based on SIMD-PE-Array

ZHONG Sheng
(Xi'an Microelectronic Technique Institute, Xi'an, Shaanxi 710075, China)

Abstract: In order to meet the challenge of real-time processing of G-level pixel frame, to solve such problems as high-volume calculation and lack of parallelism in conventional DCT, a parallel computation method implemented on SIMD PE array for DCT transform is presented in this paper. With the inherited customizability of PE array, this method can be applied to various embedded application systems regardless of their requirements on parallelism. A specific data parallel operation method using PE identifier is proposed. It not only solves the problem of "PE autonomy" in local operation, but also eliminates the cost of time in data addressing. This well-structured and compact data operation satisfies the high-regularity required by SIMD operation, and conforms to the trend of parallel processing technology.

Key words: parallel processing; DCT; SIMD PE array; mapping language

1 引言

在数字图像变换处理中,DCT(Discrete cosine Transform)是重要的变换算法,被广泛地用于图像处理领域. 由于当前的 G级像素帧变换处理,具有较高的实时性要求,使得基于不同并行体系结构下的 DCT 数据并行实现成为研究的热点[1~4]. 网格互联的细粒度 SIMD PE 阵列是最能够自然模拟图像帧数据结构的体系结构[5,6],该结构被广泛地应用于面向大规模图像处理的 MPP(Massively Parallel Processor)计算机和加速图形处理运算的嵌入式系统中,如,Geforce 8800 图形处理器(GPU,Graphic Processing Unit). 因此,本文针对网格互联的细粒度 SIMD PE 阵列研究了 DCT 变换的数据并行实现方法. 基于阵列的数据并行实现就是面向确定的算法,拟定合适的并行计算处理步骤. 在各个并行计算步骤已确定的前提下,各个并行步中,被操作数在阵列中的存储位置、数据交互中的传送距离及处理方式也是确定的,也就是说,是已知的. 本文在各并行计算步执行之前,通过预置各 PE 单元内的 PSR(Program Status Register)状态位来标识各并行计算步中的被操作数,解决各并行步执行条件计算时的 PE"自治"(automations)问题,使得并行计算的执行更为规则、简洁,省去了数据寻址时间开销,大幅度地提高处理效率,符合算法的阵列实现特点,很好地满足了阵列操作规则性强的处理要求.

2 DCT 变换的数据并行计算

Mohamed F. Mansour 算法[7,8]是采用两个 $FFT_{M/2}$ 来实现 M 个点的 DCT 变换的快速算法,以下简称为 MF

收稿日期:2008-05-26;修回日期:2008-11-19
基金项目:国防微电子预研项目(No. 41308010203)

算法.该算法规则性强,复杂度低,且其逆变换独立,不依赖于 IFFT 变换.式(1)给出了 DCT 变换的计算公式,IDCT 将在 3.3 节中给出.

$$\begin{cases} X_{DCT}(2k) = \\ \quad C(2k)\,\mathrm{Re}\{\omega_{2M}^{5k}\mathrm{FFT}_{M/2}[x(2k)+x(M-2k-1)]\} \\ X_{DCT}(2k+1) = \\ \quad \mathrm{Re}\{\omega_{2M}^{7k+(1/2)}\mathrm{FFT}_{M/2}[x(2k)-x(M-2k-1)]\} \end{cases} \tag{1}$$

$k=0,1,2,\cdots,(M/2)-1$, $C(0)=-\sqrt{2}$; $C(2k)=1$, $k\neq 0$. 其中,Re 为取实部运算.由上式不难看出,MF 算法的计算量和复杂度主要集中在 $\mathrm{FFT}_{M/2}$ 上.为描述的方便起见,令:

$$a(2k)=x(2k)+x(M-2k-1);$$
$$a(2k+1)=x(2k)-x(M-2k-1).$$

在此基础上讨论式(1)的数据并行计算.

2.1　FFT 变换的数据并行计算

FFT 是采用多次蝶式变换和位序反转来实现 DFT 变换的快速计算方法.我们首先给出 $\mathrm{FFT}_8\{a(k)\}$ 蝶式变换架构和相应的数据并行计算公式,此基础上再给出 $\mathrm{FFT}_8\{a(2k)\}_{k=0,1,\cdots,7}$ 与 $\mathrm{FFT}_8\{a(2k+1)\}_{k=0,1,\cdots,7}$ 蝶式变换的数据并行计算公式.图 1 给出了 8 点 FFT 变换的计算架构,其中,$\{a(k)\}_{k=0,1,\cdots,7}$ 为输入序列;$\{b(k)\}_{k=0,1,\cdots,7}$ 为输出序列.各次蝶式变换中的 ω_N^k 为确定的变换系数或旋转因子.

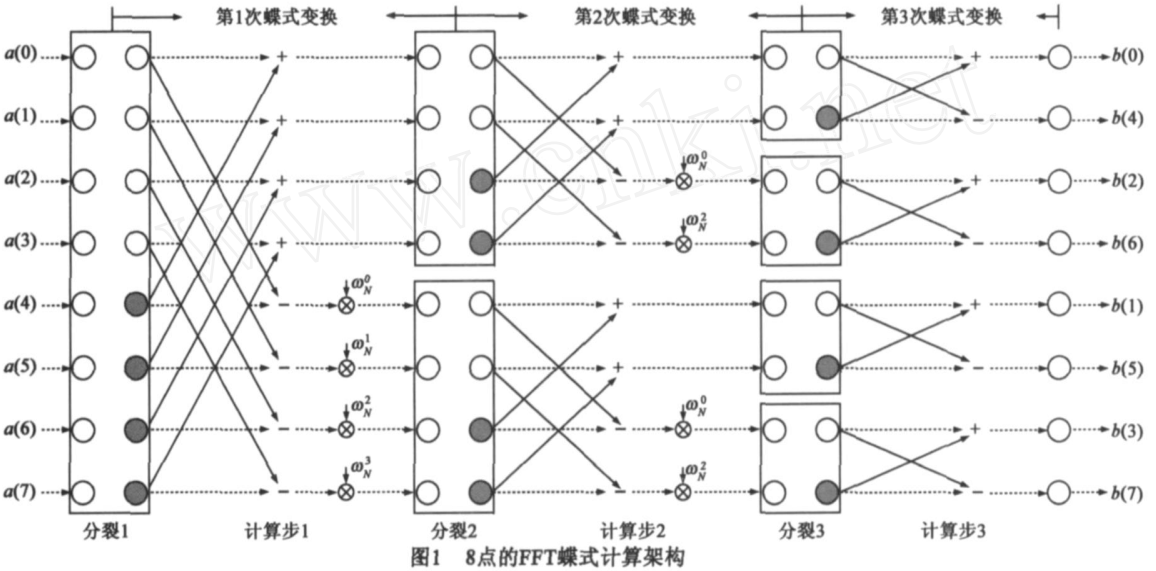


图1　8点的FFT蝶式计算架构

表 1　FFT_8 蝶式变换的分裂步计算步的数据并行计算公式

步骤名称	8 点碟式变换的计算公式
分裂 1	$A_0=\{a(0),a(1),a(2),a(3)\}$; $B_0=\{a(4),a(5),a(6),a(7)\}$
计算步 1	$A_1=\{s_1(0),s_1(1),s_1(2),s_1(3)\}=A_0+B_0$; $B_0=\{s_1(4),s_1(5),s_1(6),s_1(7)\}=\omega_0\cdot(A_0-B_0)$; $\omega_0=\{\omega_8^0,\omega_8^1,\omega_8^2,\omega_8^3\}$
分裂 2	$A_1'=\{s_1(0),s_1(1),s_1(4),s_1(5)\}$; $B_1'=\{s_1(2),s_1(3),s_1(6),s_1(7)\}$
计算步 2	$A_2=\{s_2(0),s_2(1),s_2(4),s_2(5)\}=A_1'+B_1'$; $B_2=\{s_2(2),s_2(3),s_2(6),s_2(7)\}=\omega_1\cdot(A_1'-B_1')$; $\omega_1=\{\omega_8^0,\omega_8^2,\omega_8^0,\omega_8^2\}$
分裂 3	$A_2'=\{s_2(0),s_2(2),s_2(4),s_2(6)\}$; $B_2'=\{s_2(1),s_2(3),s_2(5),s_2(7)\}$
计算步 3	$A_3=\{s_3(0),s_3(2),s_3(4),s_3(6)\}=A_2'+B_2'$; $B_3=\{s_3(1),s_3(3),s_3(5),s_3(7)\}=A_2'-B_2'$

我们采用不同基色(带填充色和无填充色)的点来标识不同的操作数据,两类基色点的分布及相应的操作,在各次蝶式变换中呈现出明显的规律.为方便后续的描述,我们将各次蝶式变换中的数据标识或“染色”,称为分裂步,将数据的交互和运算操作称为计算步.这样一来,8 点的蝶式变换就由 3 个分裂步及 3 个计算步实现.如图 1 中所示.表 1 以向量计算的形式,给出各个分裂步和计算步的数据并行计算公式.

表 1 中的向量数据:A_0、B_0;A_1'、B_1';A_2'、B_2',是由各个分裂步来确定的,各个计算步的“+”、“-”、“·”与“:=”是两个向量中对应分量间的运算,是基于分裂步下的计算.对于 $\mathbf{FFT}_8\{a(2k)\}_{k=0,1,\cdots,7}$ 与 $\mathbf{FFT}_8\{a(2k+1)\}_{k=0,1,\cdots,7}$,我们可以参照 8 点的蝶式变换进行,如表 2 所示.

对 $\mathbf{FFT}_{M/2}\{a(2k)\}$ 和 $\mathbf{FFT}_{M/2}\{a(2k+1)\}$ 而言,其计算公式是类似的,只不过是计算数据的规模不同而已.下面将主要讨论如何在网格互联的 SIMD PE 阵列上实现 DCT 变换.

3 基于 SIMD PE 阵列的 DCT 变换的数据并行实现方法

为具体起见,我们以一幅 16 × 16 的像素帧为例,来说明 DCT 变换在 SIMD PE 阵列中的数据并行实现方法. 由于 SIMD PE 阵列本身具有可裁减性,其阵列的规模是可依据应用需求进行裁减的. 因此,我们假定 SIMD PE 阵列的大小与图像帧的大小一致的,也是 16 ×16. 图像帧中的像素 Pixel[i][j]是放在 PE 阵列的处理元 PE[i][j]中,i,j 表示像素在图像帧中的位置,也就是在 PE 阵列中的位置. 阵列中各 PE 之间的互连关系,是按东西南北四个方向互连(DW(DE 和 DN(DS)的,如图 2 中所示,PE 之间只有相邻的局部通信.

表 2 $FFT_8\{a(2k)\}_{k=0,1,\cdots,7}$ 与 $FFT_8\{a(2k+1)\}_{k=0,1,\cdots,7}$ 的分裂步与计算步的数据并行计算公式

步骤名称	计 算 公 式
分裂 1	$A_0=\{a(0),a(1),a(2),a(3),a(4),a(5),a(6),a(7)\}$; $B_0=\{a(8),a(9),a(10),a(11),a(12),a(13),a(14),a(15)\}$
计算步 1	$A_1=\{s_1(0),s_1(1),s_1(2),s_1(3),s_1(4),s_1(5),s_1(6),s_1(7)\}=A_0+B_0$ $B_1=\{s_1(8),s_1(9),s_1(10),s_1(11),s_1(12),s_1(13),s_1(14),s_1(15)\}=\omega_0\cdot(A_0-B_0)$ $\omega_0=\{\omega_{16}^0,\omega_{16}^0,\omega_{16}^1,\omega_{16}^1,\omega_{16}^2,\omega_{16}^2,\omega_{16}^3,\omega_{16}^3\}$
分裂 2	$A_1'=\{s_1(0),s_1(1),s_1(2),s_1(3),s_1(8),s_1(9),s_1(10),s_1(11)\}$ $B_1'=\{s_1(4),s_1(5),s_1(6),s_1(7),s_1(12),s_1(13),s_1(14),s_1(15)\}$
计算步 2	$A_2=\{s_2(0),s_2(1),s_2(2),s_2(3),s_2(8),s_2(9),s_2(10),s_2(11)\}=A_1'+B_1'$ $B_2=\{s_2(4),s_2(5),s_2(6),s_2(7),s_2(12),s_2(13),s_2(14),s_2(15)\}=\omega_1\cdot(A_1'-B_1')$ $\omega_1=\{\omega_{16}^0,\omega_{16}^0,\omega_{16}^2,\omega_{16}^2,\omega_{16}^0,\omega_{16}^0,\omega_{16}^2,\omega_{16}^2\}$
分裂 3	$A_2'=\{s_2(0),s_2(1),s_2(4),s_2(5),s_2(8),s_2(9),s_2(12),s_2(13)\}$ $B_2'=\{s_2(2),s_2(3),s_2(6),s_2(7),s_2(10),s_2(11),s_2(14),s_2(15)\}$
计算步 3	$A_3=\{s_3(0),s_3(1),s_3(4),s_3(5),s_3(8),s_3(9),s_3(12),s_3(13)\}=A_2'+B_2'$ $B_3=\{s_3(2),s_3(3),s_3(6),s_3(7),s_3(10),s_3(11),s_3(14),s_3(15)\}=A_2'-B_2'$

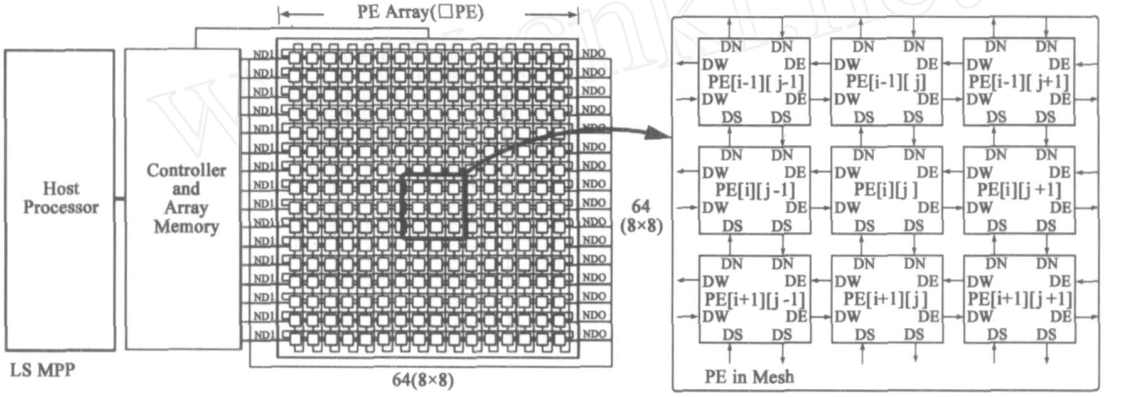


图2 SIMD PE 阵列体系结构(LS-MPP)

该阵列中,处理元 PE 由数据处理部件(算逻部件、SR 寄存器与寄存器组、局部控制逻辑)、以及相应的路由器与缓冲器三者组成,如图 1 所示. 路由器和缓冲器构成了处理元与外部通信的功能部件,而 PE 内功能单元之间的通信是通过 3 条总线,总线 A 和 B 是源操作数总线,总线 C 是结果总线. 图 3 给出处理元 PE 的内部数据路径,图 4 给出了阵列的网络互连与 IO 端口的连接示意图,芯片的内部逻辑、IO 端口、控制逻辑及路由器的 VLSI 设计参文献[9],此处限于篇幅不做深入讨论.

在 SIMD PE 阵列上对一幅二维图像进行 DCT 变换,就是对存放像素值的所有 PE,先按行方向进行数据并行计算处理,计算后的结果放入对应的 PE 单元中,然后按列方向再进行数据并行计算处理,计算后的结果,即,二维 DCT 的谱图像就存放于 PE 阵列中. 以下先以单行执行 16 点 DCT 变换为例来具体说明,其它行的

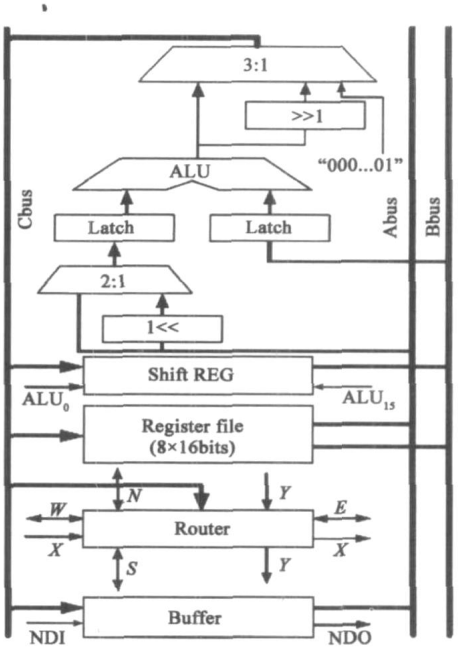


图3 处理元PE的数据路径

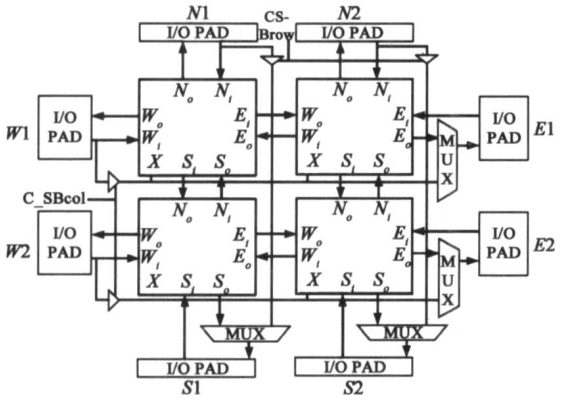


图4　PE阵列的网络互联与I/O端口逻辑图

处理与此是一致且并发执行的. 列的处理于此类似,只不过是所有列并发执行而已. 在此基础上再给出 M 点 DCT 变换的实现方法.

3.1　PE 标识配置及蝶式变换的数据并行实现

在 SIMD PE 阵列中,不是每条指令都能在每个 PE 上同时进行的,而是基于 PE 坐标位置的条件操作(条件计算和条件传送操作). 如果每次数据寻址都去查找相应的被操作数据所在的 PE[i,j]位置的话,则需要进行坐标的计算与比较,是很麻烦和费时的. 但由于各个计算步中的被操作数是在执行计算操作前,依据表 1 或表 2 所给各个分裂步事先确定的. 因此,我们不采用坐标表示的办法,而是利用各个 PE 单元内的 PSR 中应的状态位来标识各个分裂步的结果,完成图 1 中的"染色"工作. 对应于整个算法的 PE 标识,即 PSR 值是事先确定的,是在初始化阶段采用与图像数据相同的方式(行播或列播)加载入 PE 阵列中的,只不过是写入的寄存器不同而已,对执行相同算法的所有像素帧而言,加载时间仅为单帧图像数据的加载时间,对片上(SIMD PE 阵列)的资源开销影响不大. 该过程不占用计算时间,而且节省了计算过程中依 PE 坐标进行数据寻址的计算量和时间开销. 表 3 给出了表 2 对应的配置,其中未设定的 bit 位默认为"don't care"位.

表 3　16 点 DCT 分裂步执行结果在阵列中的位置分布及其 PSR 设置

数据名称	PE 单元的坐标[i][j]	PSR 状态位设置
	j 为奇/偶数	b0 = 1/0
A_0	[i][j] = [i][0,1,2,3,4,5,6,7]	b1 = 0
B_0	[i][j] = [i][8,9,10,11,12,13,14,15]	b1 = 1
A_1'	[i][j] = [i][0,1,2,3,8,9,10,11]	b2 = 0
B_1'	[i][j] = [i][4,5,6,7,12,13,14,15]	b2 = 1
A_2'	[i][j] = [i][0,1,4,5,8,9,12,13]	b3 = 0
B_2'	[i][j] = [i][2,3,6,7,10,11,14,15]	b3 = 1

这样一来在 SIMD 方式下,就可通过同步地判别各个 PE 单元内的 PSR 状态位来指定执行运算操作的各个 PE[i,j]单元. 此外,各个计算中的变换系数在 PE 阵列中的位置分布,也是可依据表 2 中的计算公式事先确定的,并且作为立即数被事先预置于对应位置的 PE 中的立即寄存器内,如表 4 所示.

表 4　16 点 DCT 的变换系数在 PE 阵列中的位置分布及相应的预置值

寄存器名称	PE 单元的坐标	变换系数值
$r_{Literal\_1}$	[i][j] = [i][10,11,12,13,14,15]	$\{\omega_8^1,\omega_8^1,\omega_8^2,\omega_8^2,\omega_8^3,\omega_8^3\}$
$r_{Literal\_2}$	[i][j] = [i][6,7,14,15]	$\{\omega_8^2,\omega_8^2,\omega_8^2,\omega_8^2\}$

图 5 给出了基于 SIMD PE 阵列的 DCT 变换的处理示意图.

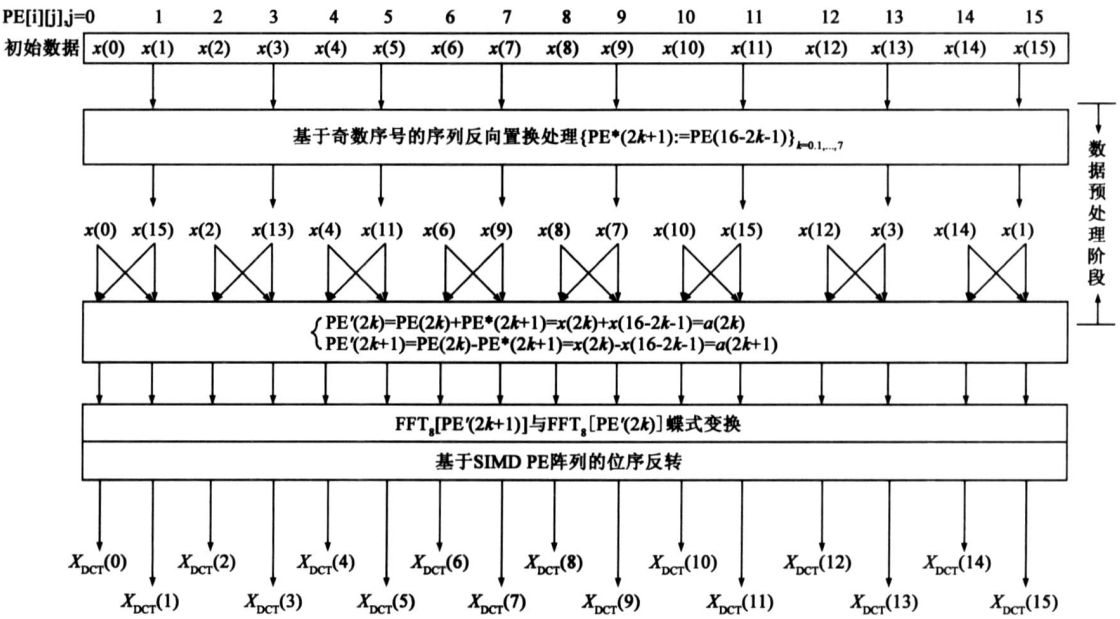


图5　基于SIMD PE阵列的DCT变换的处理示意图

这里将主要讨论如何基于表 3 的设置,按表 2 给定的计算公式,来完成数据预处理和蝶式变换的数据并行操作,位序反转将放入下一节来单独讨论. 在数据预处理阶段中,由于参与加、减运算的两个数据 $x(2k)$ 和 $x(16-2k-1)$,参式(1),在 PE 阵列中的初始位置分布呈现不同的间隔距离,直接进行操作将难以并行处理且 PE 间的通信开销会很大. 因此,在执行运算操作前增加了序列反向置换处理,这样一来,以下的加、减运算操作将仅在相邻 PE 单元间进行,易于并行且通信开销极小. 我们以第 i 行数据的序列反向置换处理为例,来具体说明,其它行的处理与此是一致且并发进行的,其过程示意图由图 6 给出.

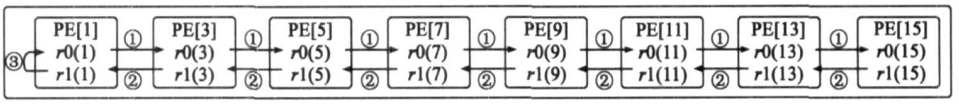


图6 奇数序号的序列反向置换处理的数据并行操作示意图

其程序实现是采用映射语言 M(Mapping language/ Middle language)[10]中基于 PSR 状态位的条件传送与条件算术指令来实现的.

```
//数据 x(0) 已存储于各 PE 单元的 r0 寄存器中;
PE[i][j] if r1 = (b0 = = 1) ?r0:NOP;//[j]为奇数的 PE 单元执行本地寄存器间传送操:r1 ←r0;
For (k = 1,k < = 7,k + +){
PE[i][j] if r0 = (b0 = = 1) ?r0[j - 2]:NOP;// ①PE[i][j] ←PE[i][j - 2];
PE[i][j] if r1 = (b0 = = 1) ?r1[j + 2]:NOP;// ②PE[i][j] ←PE[i][j + 2];
PE[i][1] r0 :r1;// ③PE[i][j = 1]执行:r1 ←r0;};
```

序列反向置换完成后,相邻 PE 间的加、减运算和蝶式并行的执行如右表.

```
//表 4 中各个计算步的变换系数已预置于相应 PE 单元的立即数寄存器 r_Literal_1 与 r_Literal_2 中.
1、If r1 = (b0 = = 0) ?r0[j + 1]:NOP;// PE[j] ←PE[j + 1];
2、If r1 = (b0 = = 1) ?r0[j - 1]:NOP;// PE[j] ←PE[j - 1];
3、If r0 = (b0 = = 0) ?{r0 + r1}:NOP;
4、If r0 = (b0 = = 1) ?{r1 - r0}:NOP;
//计算步 1 的程序实现
1、If r1 = (b1 = = 0) ?r0[j + 8]:NOP;// PE[j] ←PE[j + 8];
2、If r1 = (b1 = = 1) ?r0[j - 8]:NOP;// PE[j] ←PE[j - 8];
3、If r0 = (b1 = = 0) ?{r0 + r1}:NOP;
4、If r0 = (b1 = = 1) ?{r1 - r0}:NOP;
5、If r0 = (b1 = = 1) ?{r0 ×r_Literal_1}:NOP;
//计算步 2 的程序实现
1、If r1 = (b2 = = 0) ?r0[j + 4]:NOP;// PE[j] ←PE[j + 4];
2、If r1 = (b2 = = 1) ?r0[j - 4]:NOP;// PE[j] ←PE[j - 4];
3、If r0 = (b2 = = 0) ?{r0 + r1}:NOP;
4、If r0 = (b2 = = 1) ?{r1 - r0}:NOP;
5、If r0 = (b2 = = 1) ?{r0 ×r_Literal_2}:NOP;
//计算步 3 的程序实现
1、If r1 = (b3 = = 0) ?r0[j + 2]:NOP;// PE[j] ←PE[j + 2];
2、If r1 = (b3 = = 1) ?r0[j - 2]:NOP;// PE[j] ←PE[j - 2];
3、If r0 = (b3 = = 0) ?{r0 + r1}:NOP;
4、If r0 = (b3 = = 1) ?{r1 - r0}:NOP;
```

对于 M 点的处理方法与上是类似的,表 5 给出了各个 PE 单元的的 PSR 设置,表 6 给出了各个计算步中的变换系数在 SIMD PE 阵列中的配置.

表 5 M 点 DCT 变换对应的各分裂步执行结果在阵列中的位置分布及相应 PE 单元的 PSR 设置

数据名称	PE 单元在 $M \times M$ 阵列中的位置坐标	相应 PE 单元的 PSR 状态位设置
$A_0[i][j]$	$[i][j] = [i][j = 0,1,\cdots,(M/2)-1]$	$b1 = 0$
$B_0[i][j]$	$[i][j] = [i][j = M/2, M/2+1, \cdots, M-1]$	$b1 = 1$
$A'_1[i][j]$	$[i][j] = [i][j = \{\{[k + H(M/2)]\}_k\}_H]$	$b2 = 0$
$B'_1[i][j]$	$[i][j] = [i][j = \{\{[k + (H+0.5)(M/2)]\}_k\}_H]$ $k = 0,1,\cdots\cdots,(M/4)-1;\quad H = 0,1$	$b2 = 1$
……	……	……
$A'_{l-1}[i][j]$	$[i][j] = [i][j = \{\{[k + H(M/2^{l-1})]\}_k\}_H]$	$b(l-1) = 0$
$B'_{l-1}[i][j]$	$[i][j] = [i][j = \{\{[k + (H+0.5)(M/2^{l-1})]\}_k\}_H]$	$b(l-1) = 1$
$(l = 1,2\cdots log_2 M/2)$	$k = 0,1,\cdots\cdots,(M/2^l)-1\quad ; H = 0,2\cdots,2^l-2$	

各个计算步的程序实现,也是基于 PSR 标识进行的,与 16 点计算步的处理方法是类似的,仅需更改相应的数据传送步距、PSR 状态位和立即数寄存器即可,此处不再赘述.

3.2 位序反转的数据并行实现

经多个计算步后,$FFT_{M/2}\{a(2k)\}$ 和 $FFT_{M/2}\{a(2k+1)\}$ 分别对应的蝶式变换输出数据 $b(x_0, x_1, \cdots, x_{n-2}, x_{n-1}|_2)$,$b'(x_0, x_1, \cdots, x_{n-2}, x_{n-1}|_2)$,$n = \log_2(M/2)$,存储于阵列中列序号为 $(x_{n-1}, x_{n-2}, \cdots, x_1, x_0, 0|_2)$ 与 $(x_{n-1}, x_{n-2}, \cdots, x_1, x_0, 1|_2)$ 的 PE 单元内,需调整输出序列中各数据在阵列中的位置,即,实现 $PE[i[[j], j = (x_{n-1}, x_{n-2}, \cdots x_{(n/2)}, x_{(n/2)-1}, \cdots, x_1, x_0, 1|_2)$ 与 $PE[i]$

表 6　M 点 DCT 变换对应的变换系数在 PE 阵列中的预置([i][j]为 PE 单元在阵列中的位置坐标)

立即数寄存器的名称	PE 单元在 $M\times M$ 阵列中的位置坐标	变换系数
$r_{Literal\_1}$	$[i][j]=[i][j=\{2k+(M/2)\};\{2k+1+(M/2)\}]$ $k=0,1,\cdots,(M/4)-1$	$\omega_{M/2}^{k}$
$r_{Literal\_2}$	$[i][j]=[i][j=\{(2k+M/4),(2k+1+M/4),(2k+3M/4),(2k+1+3M/4)\}]$ $k=0,1,\cdots,(M/8)-1$	$\omega_{M/2}^{2k}$
……	……	……
$r_{Literal\_l}$ $(l=1,2,\cdots\log_2(M/2-1))$	$[i][j]=[i][j=\{[2k+(H+1)(M/2^{l})]_k,[2k+1+(H+1)(M/2^{l})]_k\}_H]$ $k=0,1,\cdots,(M/2^{l-1})-1;H=0.2\cdots,2^{l}-2$	$\omega_{M/2}^{2^{l-1}k}$
$r_{Literal}$	$[i][j]=[i]\left[j=\begin{cases}2k\\2k+1\end{cases},k=0,1,\cdots,(M/2)-1\right]$	$\begin{cases}\omega_{2M}^{2k}\\\omega_{2M}^{2k+1/2}\end{cases}$

$[j',j''=(x_0,x_1,\cdots x_{(n/2)-1},x_{n/2},\cdots,x_{n-2},x_{n-1},1|_2)$ 间的数据交换处理. 本文提出一种多步置换的方式来并行实现 N 点的数据位序倒置,仅需 $\lfloor n/2\rfloor$ 个置换步便可完成. 各置换步的处理如下:

当 n 为偶数时:

置换步 1:对地址满足 $(x_{(n/2)+1},x_{(n/2)-2})_2=(10)_2$ 与 $(01)_2$ 的所有 PE 单元分别命名为 C_1^1 与 C_1^2,C_1^1 应与步距为 t_1 的 C_1^2 作数据交换,$t_1=2^{(n/2)}$.

置换步 2:对地址满足 $(x_{(n/2)+1},x_{(n/2)-2})_2=(10)_2$ 与 $(01)_2$ 的所有 PE 单元分别命名为 C_2^1 与 C_2^2,C_2^1 应与步距为 t_2 的 C_2^2 作数据交换,$t_2=(2^{(n/2)+1}+2^{(n/2)}+2^{(n/2)-1})$.

……

置换步 $n/2$:对地址满足 $(x_{n-1},x_0)_2=(10)_2$ 与 $(01)_2$ 的所有 PE 单元分别命名为 $C_{n/2}^1$ 与 $C_{n/2}^2$,$C_{n/2}^1$ 应与步距为 $t_{n/2}$ 的 $C_{n/2}^2$ 作数据交换,$t_{n/2}=(2^{n-1}+2^{n-2}+\cdots\cdots+2^1)$.

当 n 为奇数时,仅需对各步的判别位和步距进行相应的调整即可,此处不再赘述. 图 7 以行方向上 $FFT_{M/2}\{a(2k)\}$ 和 $FFT_{M/2}\{a(2k+1)\}$ 蝶式变换完成后的位序倒置处理为例,给出了相应的处理示意图,其中 $t_1=2^{\lfloor 3/2\rfloor+1}+2^{\lfloor 3/2\rfloor}=6$.

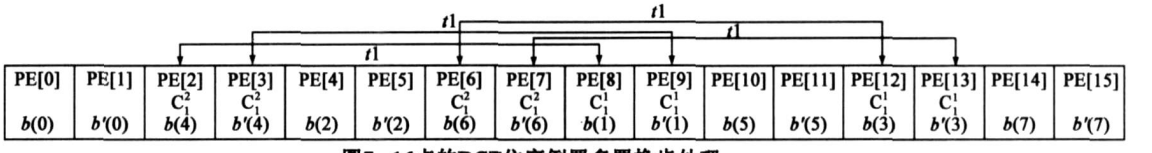


图7　16点的DCT位序倒置多置换步处理

与计算步的实现思路一致,对各置换步的实现,也采用 PE 标识预置和基于 PE 标识的并行操作处理. 由于各 PE$[i,j]$ 单元在阵列中的编号 i 和 j 是事先确定的,因此,i 和 j 相应的 bit 值也是事先确定的,可预置各 PE 中 PSR 的相应 bit 位,来标识各置换步中对应不同操作的 PE$[i,j]$. 具体设置以置换步 1,2 为例,如表 7 所示,其中未设定的 bit 位默认为"don't care"位.

表 7　各置换步中 PE[i,j]的标识,$l=1,2,\cdots\cdots,n/2$

步骤	PE 坐标 j $j=(x_{n-1},\cdots,x_{n/2},x_{,n/2-1},\cdots,x_0,(0/1)\|_2)$	PSR 状态位
1	$(x_{n/2},x_{n/2-1})=10$ $(x_{n/2},x_{n/2-1})=01$	$b(L+1)=0$ $b(L+1)=1$
2	$(x_{n/2+1},x_{n/2-2})=10$ $(x_{n/2+1},x_{n/2-2})=01$	$b(L+2)=0$ $b(L+2)=1$
……	……	……
$n/2$	$(x_{n-1},x_0)=10$ $(x_{n-1},x_0)=10$	$b(L+n/2)=0$ $b(L+n/2)=1$

各个置换步对应的并行指令如下:

```
{//第置换步 1.
1、PE[i,j];if r1=(bL==1)?r0[j-t1]:NOP;
2、PE[i,j];if r1=(bL==0)?r0[j+t1]:NOP;
3、PE[i,j];if r0=(bL==0 ‖ b2L==1)?r1:NOP;// r0←r1
//第置换步 2.
1、PE[i,j];if r1=(bL+2==1)?r0[j-t2]:NOP;
2、PE[i,j];if r1=(bL+2==0)?r0[j+t2]:NOP;
3、PE[i,j];if r0=(bL+2==0 ‖ b2L+2==1)?r1:NOP;
……
//第置换步 n/2,计算结果 SL 置于各 PE 的 r0 寄存器中.
1、PE[i,j];if r1=(b(L+n-2)==1)?r0[j-t(n/2)]:NOP;
2、PE[i,j];if r1=(b(L+n-2)==0)?r0[j+t(n/2)]:NOP;
3、PE[i,j];if r0=(b(L+n-2)==0 ‖ b(L+n-2)==1)?r1:
NOP;}
```

在执行完位序倒置处理后,所有 PE 单元执行本地计算 $r0\times r_{Lteral}$(参表 6)便可获得 DCT 变换后的数据,对所有列的 DCT 变换,与对所有行的 DCT 变换是类似的仅需在上述行变换描述的语句中,用 i 代替 j,并更改表

5、6、7 相应的状态位即可.

3.3 基于 SIMD PE 阵列的 IDCT 变换的数据并行实现方法

由 MF 算法给出的 IDCT 计算公式如下:

$$x_{IDCT}(n) = \mathrm{Re}\left\{\sum_{t=0}^{M-1} C(t)\left[X(t)\omega_{2M}^{t/2}\right]\omega_{2M}^{tn}\right\}_{n=0,1,2,\cdots,M-1}$$

令: $z(t) = X(t)\omega_{2M}^{t/2} = z_R(t) + iz_I(t)$, $t = 0,1,\cdots,M-1$, 有

$$\begin{cases} x_{IDCT}(0) = z_R(0);\ x_{IDCT}(M-1) = z_R(M/2); \\ \begin{cases} x_{IDCT}(2k) = (1/2)\{[z_R(k) + z_R(M-k)] - [z_I(k) - z_I(M-k)]\} \\ x_{IDCT}(2k+1) = (1/2)\{[z_R(k) + z_R(M-k)] + [z_I(k) - z_I(M-k)]\} \end{cases} \\ \qquad k = 1,2,\cdots,(M/2)-1 \end{cases}$$

其中, $z_R(t)$ 和 $z_I(t)$ 分别为 $z(t)$ 的实部和虚部值. 针对 IDCT 变换的阵列实现我们采用与 3.1 节相类似的方法来讨论, 首先, 我们依然以 16 点的 IDCT 变换为例, 给出本文所提并行方案的实现示意图, 如图 8 所示, 而后, 再针对该架构中的各个处理环节进行细化的讨论.

图中所示的序列位序反向是针对编号为 1 到 15 的 PE 单元进行的, 与 DCT 变换中的处理方式相类似, 此处不再重述. 以下, 仅针对序列扩展的数据并行实现作具体讨论.

数据扩展的目的是将序列反向后存储于 PE(k), $k = 1,2,\cdots,7$ 中的数据 $z(1)$ 至 $z(7)$, 复制到 PE($2k$) 中, 将 PE[8]中的数据 $z(8)$ 传送到 PE[15]中. 这样一来, 后面的计算处理能够以点操作的方式进行, 避免进一步的位序调整, 使得数据计算更加规则, 降低阵列实现的通信开销. 由于在 SIMD PE 阵列中, 数据并行执行的范围是可以指定的. 序列扩展的数据并行传送操作是采用指定范围的数据并行来实现的. 其操作的并行传送指令如下表.

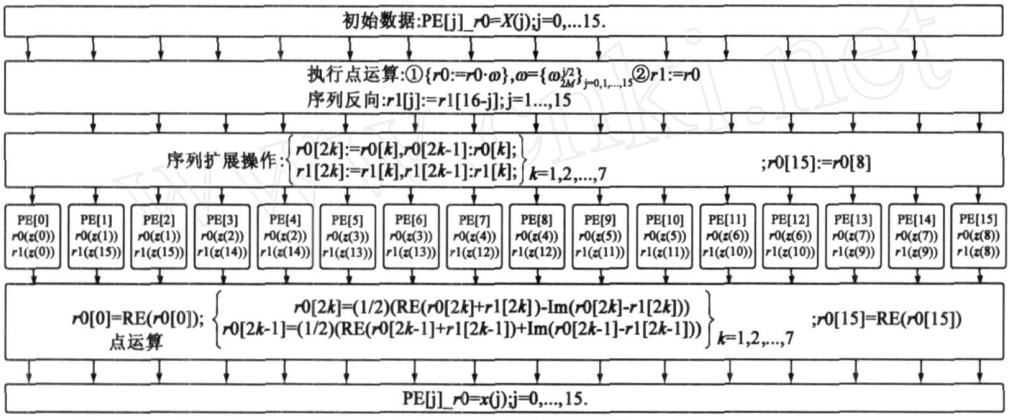


图8 16点IDCT变换的处理示意图

```
for (2 ≤j ≤15) { r0 : r0[j - 1]; r1 : r1[j - 1];}
for (4 ≤j ≤15) { r0 : r0[j - 1]; r1 : r1[j - 1];}
……
for (14 ≤j ≤15) { r0 : r0[j - 1]; r1 : r1[j - 1];}
```

在此基础上, 执行点操作运算:

```
For (1 ≤j ≤14){ For (1 ≤j ≤14)
{if (b0 = = 0) ?{ r0 = (1/2) (Re( r0 + r1) - Im( r0 - r1))} :NOP;
if (b0 = = 1) ?{ r0 = (1/2) (Re( r0 + r1) + Im( r0 - r1))} :NOP;}}.
r0[0] = Re( r0[0]) ; r0[15] = Re( r0[15]) ;
```

其中, Re 与 Im 分别为对该寄存器内的数据进行取实部与取虚部操作. 完成后, 便在 PE 阵列中, 得到一幅 IDCT 行变换后的图像. 列变换在此基础上, 以行变换相同方式, 对列进行便可.

4 性能分析与仿真验证

针对确定的变换算法, 运算量及 PE 间的通信开销, 成为衡量性能的主要指标. 表 8 给出了相关的统计值, 其中, t_i 的值参第 3.2 节内容, $N = M/2$. 表 9 以指令周期数为基本计量单位, 给出相应典型架构下的执行情况, 具体参文献[11], 其中 PE_identifier 为本文所提实现方案. 表 10 给出了 64 ×64 像素帧在 LS-MPP 计算机上不同处理阶段的具体执行时间开销. 表 11 给出了在不同规模像素帧下所能获得的加速比和效率.

表 8 DCT 的计算量及通信次数

	运算量			通信次数
	加法	乘法	位判别运算	
蝶式计算	$2\times\log_2 N$	$\log_2 N - 1$	$5\log_2 N$	$2N(1/2 + 1/4 + \cdots + 1/N)$
序列反向与位序反转	0		$3\lfloor 0.5\log_2 N\rfloor$	$3(N-1) + (2t_1 + 2t_2 + \cdots + 2t_{n/2})$

表 9 基于不同体系结构下的 8 ×8 像素帧执行周期数

	Morphosys M1	REMARC	TMS320C64x	SUN32	ARM9	PE_identifier
Cycle	37	54	92	32	1606	36

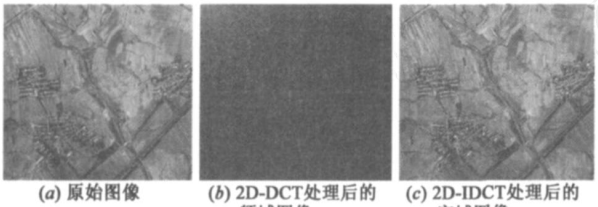
表 10 基于 LS-MPP(64 ×64Pes, 工作频率 50MHz) 下的 64 ×64 像素帧执行时间开销

处理阶段	像素帧加载	PSR 值预置	计算开销	通信开销	总执行时间
执行时间	1.25×10^{-3}μs	1.25×10^{-3}μs	1.1×10^{-3}μs	1.0×10^{-2}μs	1.4×10^{-2}μs

表 11　不同规模像素帧下的执行加速比和相应的效率

像素帧规模	8 ×8	16 ×16	32 ×32	64 ×64	128 ×128
加速比	51.89	217.42	871.78	3625.03	14633.70
效率	0.811	0.849	0.851	0.885	0.893

由表 11 可看出随像素帧规模的增大,其加速比迅速增大,其效率值渐线性或亚线性,说明该并行实现方法是可扩放的.特别是在执行过程中,其数据间的交互仅在阵列内部进行,无需访存,这将大大地降低执行时间,进一步提高执行效率,使得该实现方法能更好地满足处理的实时性需求.由于 LS-MPP 的阵列规模仅为 64 ×64 的 PE 阵列,其并行度受限,对于更大规模的图像处理采用的是软仿真方式进行的.仿真软件为 Para-llax-is- Ⅲ,验证平台为:Intel® Pentium® 4 (2.8GHz) 处理器,内存:1G,L1 data Cache: 8K,L1 Unified Cache:1024K.图 9 (a) 给出了存储于 PE 阵列中大小为 1024 ×1024 的空域图像,(b) ,(c) 为相应的变换图像.



(a) 原始图像　(b) 2D-DCT处理后的频域图像　(c) 2D-IDCT处理后的空域图像

图9

5　结束语

本文针对 G 级像帧实时处理的需要,研究了 DCT 在 SIMD PE 阵列上的数据并行计算的实现方法.这种实现方法的好处,其一是解决了在 PE 阵列中执行条件计算的 PE"自治"问题,省去了依下标寻找的计算量和相应的寻址时间.其二是各并行操作规则、简洁.其三是并行度仅受 PE 阵列大小的限制,且具有可剪裁性,能满足不同并行度需求的嵌入式系统.特别是,随着芯片集成度的提高,不仅阵列可以扩大,还可以小型化,是符合微电子和并行处理技术发展方向的.

参考文献:

[1] HSIA S C, LIU B D, et al. VLSI implementation of parallel coefficient-by-coefficient two-dimensional IDCT processor [J]. IEEE Transaction, 1995, 5(5): 396 - 406.

[2] CHIANG J S, CHUI Y F, et al. A high throughput 2-dimensional DCT/IDCT architecture for real-time and video system [A]. Proceedings of the 8th IEEE International Conference on Electronics, Circuits and Systems (ICECS) [C]. IEEE Press, 2001, Vl. 2, 867 - 870.

[3] CHANG Y T, WANG C L. New systolic array implementation of the 2D discrete cosine transform [J]. IEEE Transaction, 1995, 5(2): 150 - 157.

[4] Hyesook Lim, Changhoon Yim, et al. Finite wordlength effects of an unified systolic array for 2-d dct/idct [A]. Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP) [C]. Washington, DC, USA: IEEE Computer Society, 1996, 19(21): 35 - 44.

[5] A Rosenfeld. Parallel image processing using cellular-array computers [J]. Computer, 1983, 1(1): 177 - 191.

[6] K Preston. Cellular logic computers for pattern recognition [J]. Computer, 1983, 6(4): 36 - 37.

[7] Mohamed F Mansour. On the Odd-DFT and its applications to DCT/IDCT computation [J]. IEEE transactions, 2006, 54(7): 2819 - 2822.

[8] M Narashima, A Peterson. On the computation of the discrete cosine transform [J]. IEEE transactions, 1978, 26(6): 934 - 936.

[9] 李莉. 嵌入式 LS-SIMD 协处理器芯片的研究与设计 [D]. 西安:西安微电子技术研究所博士学位论文, 2002.

[10] 沈绪榜,等. 计算机体系结构的统一模型 [J]. 计算机学报, 2007, 30(5): 729 - 735.
Shen Xu-Bang, et al. The unified model of computer architectures [J]. Chinese Journal of Computers, 2007, 30(5): 729 - 735. (in Chinese)

[11] kchoi. Trade offs in Embeded systems design [OL]. http://vada.skku.ac.kr/ClassInfo/comp-arch/SoC-arch/%B8%F0%B5%E214-reconfigurable-architecture.pdf. 2004-06-12/2004-11-08.

作者简介:

钟　升　男,1971 年 4 月出生于西安市.博士,西安微电子技术研究所工程师.主要从事嵌入式计算机体系结构、电路设计及图像处理等研究工作.

Email: zhongsheng4166@Hotmail.com