FFT 的数据并行计算方法研究

杨 琳[1],钟 升[2],张家田[1]
(1. 西安石油大学 光电油气测井与检测教育部重点实验室,陕西 西安 710065;
2. 西安微电子研究所,陕西 西安 710065)

摘 要:为满足 G(Gigabytes)级像素帧的实时性处理需求,针对信号处理系统中处理计算量大、实时性要求高的特点,剖析了解算过程内在的数据并行特性,深入研究了基于计算阵列的谱图解算数据并行算法。提出了一种基于 MPP(Massively Parallel Processor)计算机 SIMD PE 阵列的 FFT 的数据并行计算实现方法。首先根据 FFT 架构中的数据交互一致性,给出了数据并行计算的表达式。提出一种基于 PE 标识,进行条件操作的 SIMD PE 阵列数据并行实现方法。该方法不但省去了并行处理中的数据寻址时间开销,而且使得数据并行操作更为规则、简洁,满足了阵列操作规则性强的处理要求,大幅度地提高了 MPP 计算机并行计算处理速度。该方案是一种简洁有效的 PE 自治问题解决方案,以更合理的方法和更高的效率实现了常规经典算法,在数据并行计算领域中,无疑具有重要的理论意义和应用价值,将在嵌入式信号处理中发挥愈来愈重要的作用。

关键词:快速傅里叶变换;SIMD PE 阵列;映射语言;MPP 计算机
中图分类号:TP301 **文献标识码**:A **文章编号**:1673-629X(2017)10-0091-05
doi:10.3969/j.issn.1673-629X.2017.10.020

Research on Data Parallel Computation Method of FFT

YANG Lin[1],ZHONG Sheng[2],ZHANG Jia-tian[1]
(1. Key Laboratory of Photo-electricity Gas-oil Logging and Detecting of Ministry of Education,
Xi'an Shiyou University,Xi'an 710065,China;
2. Xi'an Microelectronic Technique Institute,Xi'an 710065,China)

Abstract:In order to satisfy the real-time processing requirement of G-level pixel frame,considering the intensive and real time computing requirement of signal processing in embedded signal system,the inner data parallelism of the calculation process is analyzed,and the data parallel algorithms of spectrogram calculation based on computing array is also researched. A data parallel computation method implemented on SIMD PE array of MPP (Massively Parallel Processor) computer for FFT transform is presented. Based on the data computing consistency of FFT,the expression of data parallel computing is given firstly. Then a method of data parallel computing based on SIMD PE array to execute conditional operations by using of PE identifier is proposed,which not only omits the time cost of addressing,but makes the data parallel operation more regular and compact (only in computation statements and move statements). It meets the features of high regularity required by SIMD and greatly improves MPP computer processing speed,which is also a simple and effective PE autonomy solution,realizing conventional classic algorithms with more rational method and higher efficiency. It has important theoretical significance and application value in the area of data parallel undoubtedly,which will play a more and more important role in embedded signal processing.

Key words:FFT;SIMD PE array;mapping language;MPP computer

0 引 言

在数字图像变换处理中,DFT(Discrete Fourier Transform)是重要的变换算法,被广泛地用于图像匹配、纹理分析、去噪滤波等图像处理领域。FFT 是采用蝶式交换的处理方式,实现了 DFT 变换的快速算法。针对 G 级像素帧的 FFT 处理,由于具有较高的实时性

收稿日期:2016-10-27 **修回日期**:2017-02-14 **网络出版时间**:2017-07-19
基金项目:陕西省科技统筹创新工程计划(2015KTTSGY04-05)
作者简介:杨 琳(1982-),女,硕士研究生,研究方向为电子与通信工程;钟 升,研究员,研究方向为计算机体系结构、信号处理、微工艺系统;张家田,教授,研究方向为通信工程与信号处理。
网络出版地址:http://kns.cnki.net/kcms/detail/61.1450.TP.20170719.1109.028.html

要求,使得基于 MPP 计算机处理阵列的数据并行实现方式成为研究热点[1-4]。在并行处理计算机上如何结合体系结构高效率地实现典型算法是并行计算的核心技术[5-10]。基于阵列的数据并行实现就是针对确定的并行算法,基于数据在阵列中的位置分布特性,拟定合适的并行计算处理步骤。在各个并行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、传送距离及处理方式也是确定、已知的。对各并行步中存储被操作数 PE 单元进行事先标识,并将标识码预置于相应 PE 的 PSR 寄存器[11-14],使得基于 PE 标识的数据并行操作,不但具有了 PE“自治”(automations)能力,而且省去了数据寻址时间。各并行步的数据处理是规范,符合算法的阵列实现特点,能够很好地满足 MPP 计算机处理阵列操作规则性强的要求。

1 FFT 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[1-4]。为了方便起见,首先给出 8 点的 FFT 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 N 点的 FFT 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 为变换系数或旋转因子。

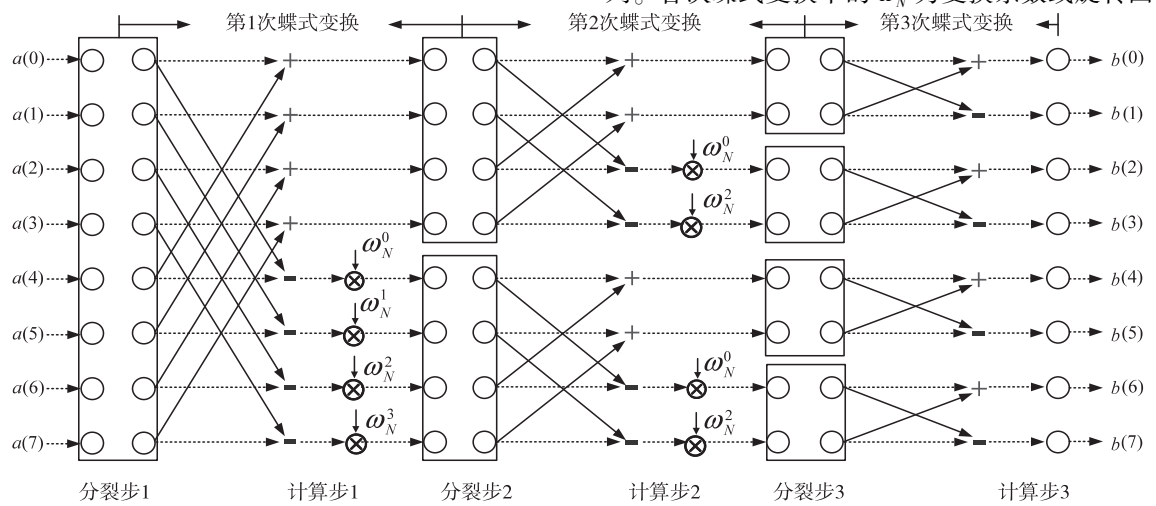


图 1 8 点的 FFT 蝶式计算架构

从图 1 可直观看出,各次蝶式变换中的数据交互和相应的计算处理是规则的,采用不同基色(带填充色的点和无填充色)的点来标识,在各次蝶式变换中的数据交互总是发生在以相同间隔分布的两类基色点间,且无填充色的点只进行“+”运算,填充色点进行“-”和与变换系数的乘法运算。两类代表不同操作数据类型的填充色原点的分布以及它们间的间隔距离,在每次蝶式解算中均呈现显著规律,数据并行操作便是基于这些规律来实现的。为便于描述,基于不同的运算操作来标识不同的数据类型,参照提升小波变换的命名方式,简称为数据分裂步,将具体的数据传送和计算操作称为数据计算步。这样一来,8 点的蝶式变换就划分为分裂 1、分裂 2 和分裂 3 以及计算步 1、计算步 2 和计算步 3。式(1)~(9)采用向量操作方式,给出对应于分裂步和数据计算步的并行计算公式。

分裂步 1:

$$\boldsymbol{A}_0 := \{a(0),a(1),a(2),a(3)\};\boldsymbol{B}_0 := \{a(4),a(5),a(6),a(7)\} \tag{1}$$

计算步 1:

$$\boldsymbol{A}_1 := \{s_1(0),s_1(1),s_1(2),s_1(3)\} = \boldsymbol{A}_0 + \boldsymbol{B}_0 \tag{2}$$

$$\boldsymbol{B}_1 := \{s_1(4),s_1(5),s_1(6),s_1(7)\} = \boldsymbol{\omega}_0(\boldsymbol{A}_0 - \boldsymbol{B}_0),\boldsymbol{\omega}_0 = \{\omega_N^0,\omega_N^1,\omega_N^2,\omega_N^3\} \tag{3}$$

分裂步 2:

$$\boldsymbol{A}_1' := \{s_1(0),s_1(1),s_1(4),s_1(5)\};\boldsymbol{B}_1' := \{s_1(2),s_1(3),s_1(6),s_1(7)\} \tag{4}$$

计算步 2:

$$\boldsymbol{A}_2 = \{s_2(0),s_2(1),s_2(4),s_2(5)\} = \boldsymbol{A}_1' + \boldsymbol{B}_1' \tag{5}$$

$$\boldsymbol{B}_2 = \{s_2(2),s_2(3),s_2(6),s_2(7)\} = \boldsymbol{\omega}_1(\boldsymbol{A}_1' - \boldsymbol{B}_1'),\boldsymbol{\omega}_1 = \{\omega_N^0,\omega_N^2,\omega_N^0,\omega_N^2\} \tag{6}$$

分裂步 3:

$$\boldsymbol{A}_2' := \{s_2(0),s_2(2),s_2(4),s_2(6)\};\boldsymbol{B}_2' := \{s_2(1),s_2(3),s_2(5),s_2(7)\} \tag{7}$$

计算步 3:

$$\boldsymbol{A}_3 = \{s_3(0),s_3(2),s_3(4),s_3(6)\} = \boldsymbol{A}_2' + \boldsymbol{B}_2' \tag{8}$$

$$\boldsymbol{B}_3 = \{s_3(1),s_3(3),s_3(5),s_3(7)\} = \boldsymbol{\omega}_2(\boldsymbol{A}_2' + \boldsymbol{B}_2') \tag{9}$$

其中,$\boldsymbol{A}_0$、$\boldsymbol{B}_0$;$\boldsymbol{A}_1$、$\boldsymbol{B}_1$;$\boldsymbol{A}_1'$、$\boldsymbol{B}_1'$;$\boldsymbol{A}_2$、$\boldsymbol{B}_2$;$\boldsymbol{A}_2'$、$\boldsymbol{B}_2'$;$\boldsymbol{A}_3$、$\boldsymbol{B}_3$ 均为向量数据,向量中的各个分量是由各个分裂步确定的,各个计算步的执行是按向量计算方式进行的,也就是“+”,“-”与“·”是两个向量中对应分量间的运算。换句话说,各个计算步中的数据并行计算,是基于

分裂步所给定的向量,并依所给向量中各个分量的序号进行的条件计算。对于 N 点的蝶式变换,采用相同的办法来处理,思路是一致的,只不过数据量较大而已。表 1 给出了 N 点蝶式变换的数据并行计算公式。

表 1　N 点 FFT 蝶式变换对应的分裂步与计算步的数据并行计算公式

步骤	计算公式
分裂 1	$\boldsymbol{A}_0 = \{a(k)\}_{k=0,1,\cdots,(N/2)-1}; \boldsymbol{B}_0 = \{a(k+N/2)\}_{k=0,1,\cdots,(N/2)-1}$
计算步 1	$\begin{cases}\boldsymbol{A}_1 = \{S_1(k)\}_{k=0,1,\cdots,(N/2)-1} = \boldsymbol{A}_0 + \boldsymbol{B}_0 \\ \boldsymbol{B}_1 = \{S_1(k+(N/2))\}_{k=0,1,\cdots,(N/2)-1} = \omega_0(\boldsymbol{A}_0 - \boldsymbol{B}_0)\end{cases}, \omega_0 = \{\omega_N^k(k+N/2)\}_{k=0,1,\cdots,(N/2)-1}$
分裂 2	$\boldsymbol{A}_1' = \{\{S_1(k)\},\{S_1(k+N/2)\}\}_{k=0,1,\cdots,(N/4)-1}; \boldsymbol{B}_1' = \{\{S_1(k+N/4)\},\{S_1(k+3N/4)\}\}_{k=0,1,\cdots,(N/4)-1}$
计算步 2	$\begin{cases}\boldsymbol{A}_2 = \{\{S_2(k)\},\{S_2(k+N/2)\}\}_{k=0,1,\cdots,(N/4)-1} = \boldsymbol{A}_1' + \boldsymbol{B}_1' \\ \boldsymbol{B}_2 = \{\{S_2(k+N/4)\},\{S_2(k+3N/4)\}\}_{k=0,1,\cdots,(N/4)-1} = \omega_1(\boldsymbol{A}_1' - \boldsymbol{B}_1')\end{cases}$ $\omega_1 = \{\omega_N^{2k}(k+N/4), \omega_N^{2k}(k+3N/4)\}_{k=0,1,\cdots,(N/4)-1}$
……	……
分裂 $l(l=1,2,\cdots,\log_2 N)$	$\begin{cases}\boldsymbol{A}_{l-1}' = \{\{S_{l-1}[k+H(N/2^l)]\}_k\}_H \\ \boldsymbol{B}_{l-1}' = \{\{S_{l-1}[k+(H+1)(N/2^l)]\}_k\}_H\end{cases}, l=1,2,\cdots,L(=\log_2 N), k=0,1,\cdots,(N/2^l)-1, H=0,2,\cdots,2^l-2$
计算步 l	$\begin{cases}\boldsymbol{A}_l = \{\{S_l[k+H(N/2^l)]\}_k\}_H = \boldsymbol{A}_{l-1}' + \boldsymbol{B}_{l-1}' \\ \boldsymbol{B}_l = \{\{S_l[k+(H+1)(N/2^l)]\}_k\}_H = \omega_{l-1}\cdot(\boldsymbol{A}_{l-1}' - \boldsymbol{B}_{l-1}')\end{cases}, \omega_{l-1} = \{\omega^{2^{l-1}k}[k+(H+1)(N/2^l)]_k\}_H$

至此,FFT 蝶式变换的数据并行计算,就由 L 个分裂步和 L 个计算步给出。下面将主要讨论如何在 SIMD PE 阵列上实现 FFT 蝶式变换。基于 SIMD PE 阵列的蝶式变换数据并行实现,就是在 SIMD PE 阵列中实现 L 个分裂步和 L 个计算步。

2　基于 SIMD PE 阵列的 FFT 蝶式变换的数据并行实现方法

为简化起见,对 8 个通道以信号长度均为 8 的信号并行进行 FFT 蝶式变换为典型示例,揭示 FFT 变换是如何在特定处理元阵列中实现数据级并行处理的。假定处理元阵列的大小与待处理数据的规模一致,即也为 8×8,且数据编号与阵列位置一致,即 Signal[i][j] 是放在 PE 阵列的处理元 PE[i][j] 中。阵列中各 PE 单元之间的互连关系采用 mesh 结构,如图 2 所示,PE 间仅具备相邻通信能力。

对多个通道进行 FFT 蝶式变换,可看成是对一幅二维信号图进行 FFT 变换。在 SIMD PE 阵列上,就是对存放像素值的所有 PE,按行方向对各 PE 中的像素值进行数据并行计算,计算后的结果依然存入相应的 PE 单元中。至此,一幅经 FFT 变换后的谱上的图像就存放于 PE 阵列中。以下先以在 8×8 的 PE 阵列中对所有的行同步执行 8 点 FFT 蝶式变换为例,具体说明 FFT 变换在 SIMD PE 阵列中的数据并行实现方法,对所有列的处理同理。在此基础上给出 N 点 FFT 蝶式变换的 SIMD PE 阵列实现方法。

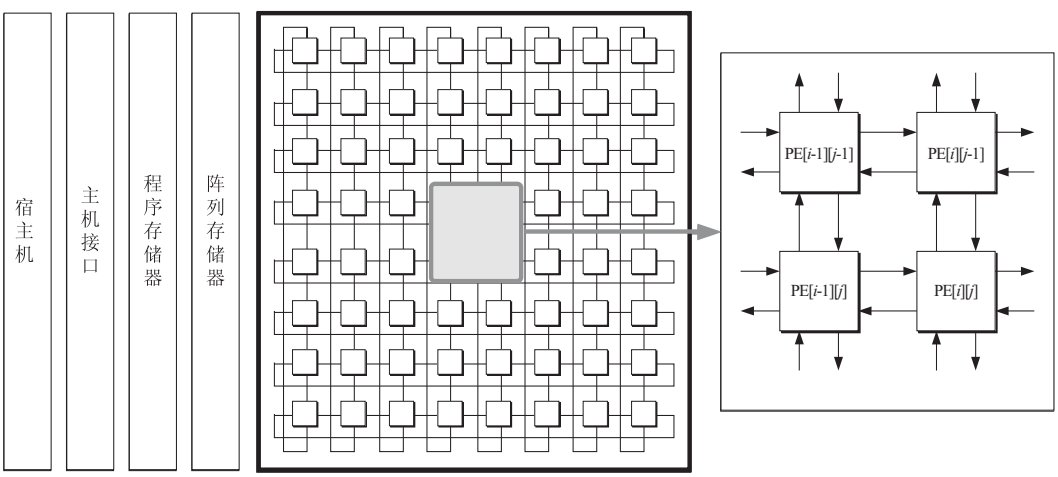


图 2　SIMD PE 阵列体系结构(LS-MPP)

在 SIMD PE 阵列中,指令执行将结合 PE 单元具体的坐标**而有数据**相应的条件操作。由于各个计算步中各个分量的序号是与像素值无关的,是依据表 1 所给各个分裂步事先确定的,或者说,可以在执行计算操

作前就可以确定。因此,不采用求解坐标位置的办法,而是利用各个 PE 单元内的 Program Status Register (PSR),即程序状态寄存器,并利用 PSR 中相应的状态位来提前标识各个分裂步的结果,或者说执行分裂步的工作。在 8×8 的 PE 阵列中的各个 PE 的 PSR 状态位中相应 bit 位将按如下原则设置:对于列坐标 $j=0$,1,2,3 的 PE 单元,其 PSR 寄存器的 b_0 位置为 0,其他 PE 单元内 PSR 寄存器的 b_0 位设置为 1;对于列坐标 $j=0,1,4,5$ 的 PE 单元,其 PSR 寄存器的 b_1 位设置为 0,其他 PE 单元内 PSR 寄存器的 b_1 位设置为 1;对于列坐标 $j=0,2,4,6$ 的 PE 单元,其 PSR 寄存器的 b_2 位设置为 0,其他 PE 单元内 PSR 寄存器的 b_2 位设置为 1。这样一来就可通过判别 PSR 的状态位来确定各个计算步中相应向量的各个分裂,或执行指定运算操作的各个 PE[i,j]单元,该方法使得后续的阵列操作简洁、规则,而且没有了寻址时间开销。此外,FFT 变换系数依据表 1 中的计算公式确定并作为立即数被提前预置于对应位置的 PE 内的立即数寄存器中。在 SIMD PE 阵列中,各个计算步的并行实现,就是基于各个 PSR 设置,按表 1 给定的计算公式,在阵列中执行相应的条件传送与条件计算操作。各个计算步的并行执行是以状态位为条件,采用映射语言 M(Mapping language/Middle language)[12] 的条件传送语句与条件算术语句等描述的。

//在数据加载阶段,初始数据 A0 与 B0 已加载至各 PE 单元的 DATA_registeR0 寄存器中,变换系数加载于相应 PE 单元的立即数寄存器 LITERAL_register_1 与 LITERAL_registe_2 中,DATA_registeR1

//在计算阶段计算步 1 的并行实现

1 If DATA_registeR1=(b_0 = =0)? DATA_registeR0[j +4]: NOP;//PE[j]←PE[j +4];

2 If DATA_registeR1=(b_0 = =1)? DATA_registeR0[j -4]: NOP;//PE[j]←PE[j -4];

3 If DATA_registeR0=(b_0 = =0)?{DATA_registeR0+DATA_registeR1}:NOP;

4 If DATA_registeR0=(b_0 = =1)?{DATA_registeR1-DATA_registeR0}:NOP;

5 If DATA_registeR0=(b_0 = =1)?{DATA_registeR0×LITERAL_register_1}:NOP;

//在计算阶段计算步 2 的并行实现

1 If DATA_registeR1=(b_0 = =0)? DATA_registeR0[j +2]: NOP;//PE[j]←PE[j +2];

2 If DATA_registeR1=(b_0 = =1)? DATA_registeR0[j -2]: NOP;//PE[j]←PE[j -2];

3 If DATA_registeR0=(b_0 = =0)?{DATA_registeR0+DATA_registeR1}:NOP;

4 If DATA_registeR0=(b_0 = =1)?{DATA_registeR1-DATA_registeR0}:NOP;

5 If DATA_registeR0=(b_0 = =1)?{DATA_registeR0×LITERAL_register_2}:NOP;

//在计算阶段计算步 3 的并行实现

1 If DATA_registeR1=(b_0 = =0)? DATA_registeR0[j +1]: NOP;//PE[j]←PE[j +1];

2 If DATA_registeR1=(b_0 = =1)? DATA_registeR0[j -1]: NOP;//PE[j]←PE[j -1];

3 If DATA_registeR0=(b_0 = =0)?{DATA_registeR0+DATA_registeR1}:NOP;

4 If DATA_registeR0=(b_0 = =1)?{DATA_registeR1-DATA_registeR0}:NOP;

对于 8 点的 FFT 蝶式变换,可按照上述方式处理,仅是数据传递的步距和执行数据的规模不同而已。对于 N 点的 FFT 蝶式变换,处理方法与上述方法类似,PE 单元内 $\log_2 N-1$ 个系数寄存器 LITERAL_register_1 至 LITERAL_register_ m , $m=1,2,\cdots,\log_2 N-1$ 的设置依据列坐标 $j=\{\{k+(H+1)*N/2^m\}_k\}_H$, $k=0,1,\cdots,(N/2^m)-1$; $H=0,2,\cdots,2^m-2$,其中的变换系数为 $\omega_N^{2^{m-1}k}$,PE 单元的 PSR 设置与 8 点 FFT 的设置是类似的。

对应于 IDFT 的 IFFT 蝶式处理算法,由于各处理阶段中的待处理数据在阵列中的分布位置,传送距离及处理方式与 FFT 基本一致,唯一不同的仅仅是各计算步中的变换系数而已,所以仅需将 FFT 中各计算步的变换系数更换为其倒数即可。例如,将 ω^k 更换为 ω^{-k},并存储于相同的 PE 单元中。其他设置及处理均与 FFT 一致,在此不再赘述。

3 性能分析与仿真验证

在运算量确定的情况下,PE 间的通信开销成为衡量性能的主要指标。以下针对上述实现方法,对计算量和通信量进行讨论。对于 FFT 变换,计算量主要取决于算法实现中的乘、加次数和判别运算的次数。通信量主要取决于蝶式运算和位序倒置变换中的数据交互次数,表 2 给出了相关的统计值。

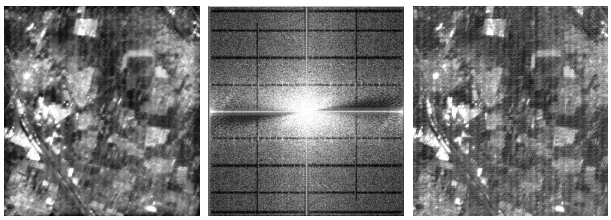
表 2 FFT 的计算量及通信次数

运算类型	运算量			通信量
	加法	乘法	位判决	
蝶式运算	$2\log_2 N$	$2\log_2 N-1$	$5\log_2 N$	$2N(1/2+1/4+\cdots+1/N)$
位序运算	0		$2\log_2 N$	$2N(t_1+t_2+\cdots+t_{n/2})$

在 FFT 变换运算中,涉及大量的复数加法和乘法运算。1 次复数加法由 2 次实数加法构成,1 次复数乘法由 4 次实数乘法和 2 次实数加法构成。以 $N=64\times64$ 点的 FFT 计算为例,在单处理器环境下,完成蝶式变换,共需 64×(64×6)次复数加法和 64×(32×5)次复数乘法;完成位序倒置,共需 64×56 次数据交换操作。

为简化起见,约定实数的加法和乘法运算、位判别运算以及 PE 单元间的相邻通信均可在一个指令周期内完成,并且访存时间也为 2 个指令执行周期。则对于单处理器而言,按上述约定,大约需要 372 736 个指令周期。对于文中方案,由表 2 可简单地计算出共需 634 个指令周期。其加速比约为 587.92,很可观。文中采用的仿真软件为 Parallaxis-Ⅲ,完成 PE 阵列的规模和互联结构的定义,其中 PE 间的数据交互是使用系统函数 MOVE 模拟实现。

如图 3 所示,通过仿真验证了实现方法的正确性。



(a) 空域图像　(b) FFT 处理后的频域图像　(c) IFFT 处理后的空域图像

图 3　2D-FFT 变换仿真图像

4　结束语

针对 G 级像帧实时处理的需要,研究了 FFT 在 MPP 计算机处理 SIMD PE 阵列上的数据并行计算实现方法。该方法的并行度不受算法限制,而且省去了寻址计算量和寻址时间,各个并行操作步骤规则、简洁且通用性强。该方法亦可应用于其他算法的并行实现[12-14]。

从硬件上讲,并行度仅受 PE 阵列规模的限制。由于 SIMD PE 阵列具有较强的可剪裁性,其大小是容易随应用的并行性要求而变化的;特别是随着芯片集成度的提高,阵列规模急速上升,容易满足大规模数据的处理需求。

参考文献:

[1] Tong C, Swartztrauber P N. Ordered fast Fourier transform on an massively parallel hyper cube multiprocessor[J]. Journal of Parallel and Distribute Computing, 1991, 12(1): 50-59.

[2] Swartztrauber P N. Multiprocessor FFTs[J]. Parallel Computing, 1987, 5(1-2): 197-210.

[3] Jamieson L H, Muller L P, Siegal H. FFT algorithm for SIMD parallel processing system[J]. Journal of Parallel and Distributed Computing, 1986, 3(1): 48-71.

[4] Berland G D, Wilson D. A fast Fourier transform algorithm for a global, highly parallel processor[J]. IEEE Transactions on Audio and Electroacoustics, 1969, 17(2): 125-127.

[5] Khailany B, Dally W J, Kapasi U J, et al. Imagine: media processing with streams[J]. IEEE Micro, 2001, 21(2): 35-46.

[6] Rixner S. Stream processor architecture[M]. Boston, MA: Kluwer Academic Publishers, 2002.

[7] Kapasi U J. The imagine stream processor[C]//Proceedings of IEEE 2002 international conference on computer design. [s. l.]: IEEE, 2002: 282-288.

[8] Serebrin B. A stream processor development platform[C]// Proceedings of IEEE 2002 international conference on computer design. [s. l.]: IEEE, 2002: 303-308.

[9] Crowley P. Network processor design issues and practices [M]. [s. l.]: Morgan Kaufmann Publishers, 2003.

[10] 陈国良. 并行算法的设计与分析[M]. 北京:高等教育出版社, 2002: 380-409.

[11] 沈绪榜,刘泽响,王　茹. 计算机体系结构的统一模型[J]. 计算机学报, 2007, 30(5): 729-736.

[12] 钟　升,沈绪榜,郑江滨,等. 提升小波变换的数据并行计算方法研究[J]. 计算机学报, 2011, 34(7): 1323-1331.

[13] 钟　升. 基于 SIMD PE 阵列的 DCT 数据并行实现方法研究[J]. 电子学报, 2009, 37(7): 1546-1553.

[14] 钟　升. 基于小波变换的图像 bit 纠错数据并行实现研究[J]. 系统仿真学报, 2008, 20(8): 2137-2141.

(上接第 90 页)

interface architecture for video processing applications[J]. IEEE Transactions on Circuits & Systems for Video Technology, 2001, 11(11): 1160-1170.

[8] Zhu J, Hou L, Wu W, et al. High performance synchronous DRAMs controller in H.264 HDTV decoder[C]//7th international conference on solid-state and integrated circuits technology. [s. l.]: IEEE, 2004: 1621-1624.

[9] Wang T H, Chiu C T. Low power design of high performance memory access architecture for HDTV decoder[C]//2007 IEEE international conference on multimedia and expo. [s. l.]: IEEE, 2007: 699-702.

[10] 刘贤梅,任　重. H.265 视频编码器在 TMS320C6678 上的优化实现[J]. 计算机技术与发展, 2015, 25(3): 171-174.

[11] 王尊亮,李学俊. 分布式视频编码技术研究进展[J]. 计算机工程与设计, 2010, 31(3): 550-554.

[12] Kim J, Kyung C M. A lossless embedded compression using significant bit truncation for HD video coding[J]. IEEE Transactions on Circuits & Systems for Video Technology, 2010, 20(6): 848-860.

[13] Lei J, Zou X, Wu Z, et al. Research of an image map encoding algorithm on frame buffer[C]//7th international conference on ASIC. [s. l.]: IEEE, 2007: 894-897.

[14] Yng T B, Lee B G, Yoo H. A low complexity and lossless frame memory compression for display devices[J]. IEEE Transactions on Consumer Electronics, 2008, 54(3): 1453-1458.

[15] 陶思平. 三维视频编码的关键技术研究[D]. 合肥:中国科学技术大学, 2009.

[16] Dikbas S, Zhai F. Lossless image compression using adjustable fractional line-buffer[J]. Signal Processing Image Communication, 2010, 25(5): 345-351.