中国科学: 信息科学 2011 年 第 41 卷 第 5 期: 541–551
www.scichina.com info.scichina.com
《中国科学》杂志社
SCIENCE CHINA PRESS
论 文

三值光学计算机中 MSD 加法器的理论和结构

金翊[①*], 沈云付[①], 彭俊杰[①], 徐拾义[①], 丁广太[①], 岳东剑[①], 尤海航[②]

① 上海大学计算机工程与科学学院, 上海 200072
② National Institute for Computational Sciences, Oak Ridge, Tennessee, TN37831, USA
* 通信作者. E-mail: yijin@shu.edu.cn

收稿日期: 2010–04–23; 接受日期: 2010–08–09
国家自然科学基金 (批准号: 61073049)、教育部博士点建设基金 (批准号: 20093108110016) 和上海市教委重点学科建设项目 (批准号: J50103) 资助项目

摘要 在三值光学计算机中采用 MSD 加法器, 将使这个光学计算机的 “三值” 和 “数据位数众多” 两大优势在数值计算中得以充分发挥. 以降值设计理论为基础, 三值光学计算机既可以随时构造出一个数千位的大加法器来满足计算超大数据的用户, 也可以常备一个数据位数适中的加法器来满足普通用户的需要. 引入流水计算技术和发明数据剪辑技术后, 常备加法器的效率提高到接近每个时钟输出一个计算结果. MSD 加法器的成功, 促进了三值光学计算机进行乘法运算、除法器和各类矩阵运算的研究.

关键词 三值光学计算机 MSD 数字 加法器 流水计算技术 数据剪辑

1 三值光学处理器的特点

随着大型电子计算机系统的复杂度不断提高, 其功耗大到难以接受的地步, 于是人们越来越关注各种新形式的计算机, 光学计算机成为人们关注的焦点之一. 光有不同于电的物理特性, 导致光学计算机有不同于电子计算机的特点: 速度可以更快、位数可以更多、使用更多的物理状态 (多值) 等, 理想的光学计算机应该兼有这些特点. 目前的光学计算机研究中, 有许多研究着力于 “高速度”, 同时也有研究着力于 “位数众多”.

三值光学计算机实验系统就是已经成功的巨位数计算机 [1,2], 依据 2008 年公开的降值设计原理 [3], 可以随时把三值光学处理器近千个数据位的任意区域构造成需要的三值 (包括二值) 逻辑运算器. 然而, 众多的数据位数不能容忍行波加法器中进位过程的延时, 光学部件又很难实现先行进位加法器中的进位树结构, 在三值光学计算机研究的初期, 作者就注意到这个难题, 于 2003 年提出了进位直达并行通道加法器原理和结构 [4], 但多种因素致使这种加法器没能付诸实施. 因此, 两年来三值光学计算机实验系统一直不便进行大规模算术运算. 2009 年底, 作者认真总结了三值光学计算机系统的各项研究和实验 [5,6], 建立了三值光学计算机的 MSD(modified signed-digit) 计数制的数值计算体系, 它包括 MSD 加法器、乘法例程、除法例程和矩阵乘法例程, 其中 MSD 加法器是基础和核心.

三值光学计算机用两个偏振方向正交的偏振态和无光态表示信息, 所以, 它的一个数据位上有三个取值, 除 0 和 1 之外, 还可以再取一个值, 如 −1. 它用液晶阵列控制光束的偏振方向, 配合偏振片

引用格式: 金翊, 沈云付, 彭俊杰, 等. 三值光学计算机中 MSD 加法器的理论和结构. 中国科学: 信息科学, 2011, 41: 541–551

来完成信息处理. 由于液晶阵列的像素非常多, 所以三值光学计算机的数据位数非常多 [2,7], 2007 年建成的实验系统中有 360 位, 目前在建的实验系统达到 1000 位, 而且很容易继续扩大.

三值光学计算机实验系统的光学处理器依据降值设计原理建造, 对这种处理器而言, 降值设计原理表述为: 在 3^9(=19683) 个三值逻辑处理器中, 任意一个都可以用 18 种运算基元组合而成, 而且用到基元总数不多于 6 个. 于是, 三值光学处理器中只要对 18 种运算基元准备好足够的数量, 就可以随时把光学处理器的任何部分构造成用户需要的某个三值逻辑运算器. 这称为三值光学处理器的重构性 [3].

由于逻辑运算的位和位之间没有关联, 因此可以把一个光学处理器的不同位置, 即数据位的不同片段, 构造成不同的逻辑运算器. 于是, 在一个操作指令下, 这个光学处理器的不同区域能完成不同的逻辑运算. 这是电子计算机所做不到的. 另一方面, 又可以把一个光学处理器看作是一个大光学处理器的一部分, 当这个光学处理器的数据位数不够多时, 可以启用大光学处理器的其他部分, 即给这个光学处理器拼接一个新的光学处理器, 二者形成一个较大的光学处理器, 而不像电子计算机形成 “多 CPU” 或 “多核”. 因此, 当用几个光学处理器拼接成一个大处理器时不存在 “核间通信” 和 “核间同步” 问题. 这是与电子计算机的又一个不同点. 同时, 光学处理器的这种直接拼接性, 更使得三值光学计算机可以有很多的数据位数, 理论上多到满足任何用户的需要.

鉴于液晶阵列和偏振片制作工艺的成熟性、液晶阵列在像素数量上的优势和提升液晶响应速度的潜在可能性等情况, 目前选择液晶阵列来制作三值光学处理器. 在这个选择下, 18 种基本运算单元具有相似的结构 —— 两个偏振片夹一个液晶像素. 其差别在于偏振片的偏振方向和液晶的静态旋光性. 于是, 三值光学处理器具有图 1 所示的结构. 其中, 液晶阵列和偏振片 P_1、P_2 构成所有的基本运算单元; 重构控制器模块含有液晶静态旋光性控制电路和选择、组合基本运算单元成特定逻辑运算器的例程, 并对外提供汇编指令. 系统的位数管理程序用这些汇编指令来满足用户的要求, 使重构控制器模块发出重构光学处理器信号 e, 在光学处理器的指定位置构造出指定位数的指定逻辑运算器, 这个过程称为重构光学处理器.

输入光信号 (数据 a 和 b) 从两个不同的位置进入光学处理器, 其中一个 (如 b) 产生对光学处理器上同一数据位的液晶像素的控制信号 d, 另一数据 (a) 通过偏振片 P_1、P_2 和液晶阵列的组件时, 被改变光状态, 成为运算结果 —— 输出光信号 c. 由于目前光控液晶还不成熟, 图 1 所示的系统中使用电控液晶阵列来做光学处理器, 相应的用光电转换器来接收 b 光线, 并产生对液晶的控制电信号 d. 图中的解码器用于把输出光信号 c 转换成电信号, 以便和电子计算机交换数据.

三值光学处理器可重构和位数众多的特性, 使它更适合处理数据位之间没有关联的逻辑运算. 常规加法运算中数据位之间的进位关联性, 将使三值光学处理器难以发挥它的优点. 幸好到 20 世纪 60 年代, 数学家们已经为解决这个问题做好了理论准备. 那个年代, 数学家为解决电子计算机加法器中进位过程的串行问题, 完善了冗余表示计数法 [8,9]. 在这种计数法中, 加法运算没有进位过程, 这特别适合计算机使用. 但二进制的电子计算机只有两个值, 无法在硬件上实现 “冗余计数”, 所以, 这种计数法一直没在电子计算机中得到应用. 恰好, 三值光学计算机有三个取值, 可以用一个值做冗余, 进行冗余二进制计算. 因此, 二者的结合相得益彰.

2 MSD 计数法

常规计数法则中, n 进制有 n 个计数符号, 分别表示值 0 到 $n-1$, 且逢 n 进一. 如果规定 n 进制

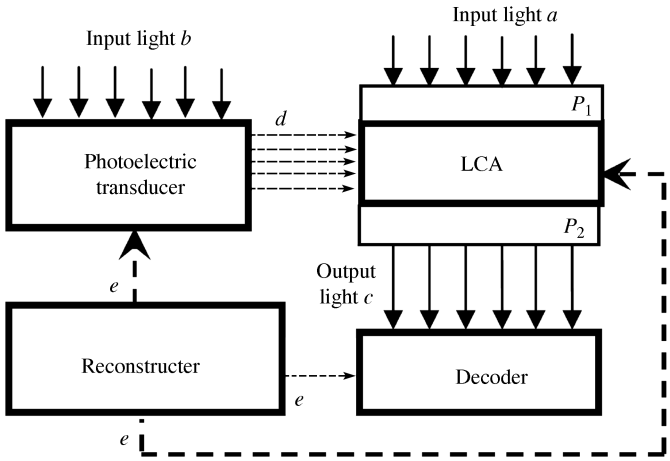

图 1　三值光学处理器结构示意

Figure 1　Block diagram of ternary optical processor

的每一位有 $n+r$ 个计数符号 (n, r 为正整数), 且仍然逢 n 进一, 则每一位可以表示 $n+r-1$ 个值, 称为冗余计数. 显然, 在冗余计数中一个数值可能有几个表示形式, 这对人类使用很不方便, 但在多值计算机中, 却有它的方便之处, 下文将充分利用这些便利.

在三值光学计算机, 只能采用 $n=2, r=1$ 的冗余计数法, 在这一类冗余计数法中, 使用上最便利的一种是 MSD 计数法.

2.1　MSD 表达式

数值 A 的 MSD 表达式为

$$A=\sum_i a_i\times 2^i, \tag{1}$$

其中, a_i 的值域为 $\{\bar{1},0,1\}$, $\bar{1}$ 表示 -1; 2^i 说明 MSD 仍是二进制计数法. 由于 a_i 有 3 个可取值, 则对于一个数值可以有多个 MSD 表示. 例如, 对于数值 5 有

$$(5)_{10}=(101)_2=(101)_{\mathrm{MSD}}=(10\bar{1}\bar{1})_{\mathrm{MSD}}=(1\bar{1}0\bar{1}\bar{1})_{\mathrm{MSD}},$$

而数值 -5 的表示为

$$(-5)_{10}=(-101)_2=(\bar{1}0\bar{1})_{\mathrm{MSD}}=(\bar{1}011)_{\mathrm{MSD}}=(\bar{1}1011)_{\mathrm{MSD}}.$$

对比上两式, 在 MSD 表示中对一个数的表示式取反就得到对应负数的表示式, 这个特点具有一般性, 所以, MSD 不需要增设符号位.

2.2　MSD 加法

MSD 加法的运算过程与常规二进制数加法不同, 它依靠表 1 给出的 4 个变换 (T, W, T′, W′) 来完成 [10,11]. 这 4 个变换都是三值逻辑运算, 运算过程可分为下列 5 个步骤:

第 1 步: 对两个输入数 a 和 b 的对应位分别进行 T 变换和 W 变换, 位数少的数前面补 0. T 变换的结果称为 u, W 变换的结果称为 w.

表 1　MSD 加法使用的 $\mathbf{T}(\mathbf{T}_2)$, W, $\mathbf{T}'$ 和 $\mathbf{W}'$ 变换

Table 1　T, W, T′ and W′ transformations used in MSD addition

$T(T_2)$ transform				W transform				T′ transform				W′ transform			
$b(w')$	$a(t')$			b	a			w	t			w	t		
	$\bar{1}$	0	1		$\bar{1}$	0	1		$\bar{1}$	0	1		$\bar{1}$	0	1
$\bar{1}$	$\bar{1}$	$\bar{1}$	0	$\bar{1}$	0	1	0	$\bar{1}$	$\bar{1}$	0	0	$\bar{1}$	0	$\bar{1}$	0
0	$\bar{1}$	0	1	0	1	0	$\bar{1}$	0	0	0	0	0	$\bar{1}$	0	1
1	0	1	1	1	0	$\bar{1}$	0	1	0	0	1	1	0	1	0

第 2 步: 给 u 后面补 0, 称为中间结果 t.

第 3 步: 对两个中间结果 t 和 w 的对应位分别进行 T′ 变换和 W′ 变换, 位数少的数前面补 0. T′ 变换的结果称为 u', W′ 变换的结果称为 w'.

第 4 步: 给 u' 后面补 0, 称为中间结果 t'.

第 5 步: 对两个中间结果 t' 和 w' 的对应位再次进行 T 变换, 位数少的数前面补 0, 变换结果就是最终结果 c. 下文及表 1 中称这次 T 变换为 T_2, 以便于书写中区分这两次 T 变换.

例如 (以下数字如不特别说明均为 MSD 数字): 设 a=110100=$(52)_D$, b=$1\bar{1}100\bar{1}$ =$(23)_D$, 下标 D 表示十进制数. 则

$$c = a + b = 110100 + 1\bar{1}100\bar{1}.$$

第 1 步: 对 a, b 进行 T 变换有: $u = 10110\bar{1}$; 对 a, b 进行 W 变换有: $w = 00\bar{1}\bar{1}01$.

第 2 步: 给 u 后边补一个 0, 有: $t = 1011\bar{0}1\bar{0}0$.

第 3 步: 对 t, w 进行 T′ 变换有: $u' = 0000000$; 对 t, w 进行 W′ 变换有: $w' = 1010\bar{1}\bar{1}1$.

第 4 步: 给 u' 后边补一个 0, 有: $t' = 00000000$.

第 5 步: 对 t', w' 进行 T_2 变换有: $c = 01010\bar{1}\bar{1}1$.

将 c 转换成十进制表示为 $(75)_D$, 与十进制加法 $c = a + b = (52)_D + (23)_D = (75)_D$ 一致.

b 还有一个与常规二进制形式一样的 MSD 表示: $b = 1\bar{1}100\bar{1} = 10111 = (23)_D$. 用这个形式的 b 进行 MSD 加法的中间值分别是

$$u = 110111, w = \bar{1}000\bar{1}\bar{1}, t = 1101110; u' = 0000000, w' = 1001101, t' = 00000000.$$

其最终结果是 $c = 1001101$ 转换成十进制数仍然是 $(75)_D$. 可见常规二进制也是 MSD 数的一种形式, 用其进行加法, 结果与其他 MSD 表示形式的加法结果相一致.

3　三值光学计算机 MSD 加法器

理论上, 三值光学计算机的 3 个物理状态, 与 MSD 数的 3 个符号 ($\bar{1}$, 0 和 1) 之间取任意对应关系都是等价的, 本文为叙述方便, 取一个偏振态表示 $\bar{1}$, 无光态表示 0, 另一个偏振态表示 1. 于是, MSD 数在三值光学计算机中得到自然的表达, 二值计算机做不到这一点.

由 2.2 小节的论述知: MSD 加法通过 $T(T_2)$, W, T′ 和 W′ 4 个三值逻辑运算器来完成. 又由第 1 节知, 这些逻辑运算器可以随时在三值光学处理器的任意位置 (任意数据位上) 重构而成, 所以, 在三值光学计算机中完成 MSD 加法有 3 个途径.

途径 1: 顺序把光学处理器的全部位数构造成 T、W、T′、W′ 和 T_2 变换器, 每构造成一个变换器, 对数据进行相应的变换. 其优点是加法器的位数最大, 达到光学处理器的数据位数, 缺点是对光学处理器要重构 5 次, 费时较多.

途径 2: 把光学处理器的数据位分成 z_1 和 z_2 两部分, 把 z_1 构造成 T, 同时把 z_2 构造成 W, 对输入数据同时进行 T 和 W 变换; 再把 z_1 构造成 T′, 同时把 z_2 构造成 W′, 对上次变换的结果数据同时进行 T′ 和 W′ 变换; 然后再把 z_1 构造成 T_2 变换器, 再对上次变换的结果进行 T 变换, 得到计算结果. 特点是加法器的位数是光学处理器的一半, 同时对光学处理器的重构次数减为 3 次.

途径 3: 把光学处理器的数据位分成 5 个部分, 同时把每个部分构造成一个变换器, 输入数据同时通过 T 和 W, 其输出再同时通过 T′ 和 W′, 它们的输出又通过 T_2, 得到最后结果. 其优点是只需要重构一次光学处理器, 但数据位数最多达到光学处理器的 1/5.

用途径 3 做加法, 只需要在开始时构造一次光学处理器, 因此, 三值光学计算机不仅可以随时为用户构造需要的加法器, 还可以事先构造好一个 MSD 加法器备用, 当普通用户做加法时, 就直接使用这个常备的加法器, 从而在工作开始后省掉重构光学处理器的操作时间. 但这个常备加法器构造成多少位才适合大多数用户使用是一个难以确定的问题, 为此, 我们发明了数据剪辑技术来解决这个问题, 这项技术将在本文第 5 节首次公开. 另外, 把流水计算方式引入 MSD 加法器后, 它的工作效率提高到宏观上接近一个时钟完成一个加法的水平. 这项技术将在本文第 4 节介绍. 在这两项技术的支撑下, 三值光学计算机采用途径 3 来构造加法器, 其工作步骤描述如下:

第 1 步: 将光学处理器分成 5 个部分, 同时将其分别用于构造 T、W、T′、W′ 和 T_2. T 和 W 的长度为 q 位, T′ 和 W′ 的长度为 $q+1$ 位, T_2 的长度为 $q+2$ 位. 对于常备的 MSD 加法器, 这一步可以在系统初始化时完成, 在日后的长期工作中省去.

第 2 步: 把输入数据 a, b 同时送入 T 和 W 得到中间结果 u 和 w.

第 3 步: 在 w 前补 0, 在 u 后补 0 生成 t.

第 4 步: t 和 w 同时送入 T′ 和 W′, 得到中间结果 u' 和 w'.

第 5 步: 在 w' 前补 0, 在 u' 后补 0 生成 t'.

第 6 步: t' 和 w' 送入 T_2, 得到计算结果 c.

4 MSD 加法器的流水计算方式

在现代电子计算机中, 指令的执行过程几乎都采用流水线方式, 乘法器中也常常采用流水计算方式. 但是, 二进制加法进位过程的相关性, 使电子计算机的加法计算过程无法采用流水线方式, 总是一组数据独占加法器. 然而, MSD 加法没有进位过程, 三值光学处理器采用的 MSD 加法器可以分解成 5 个独立的逻辑运算部件, 而且逻辑部件的使用顺序不能改变, 这为在加法运算过程中引入流水计算方式创造了条件. 对于三值光学计算机的 MSD 加法器而言, 流水计算的具体做法为:

若待运算的数据很多, 在第 1 组数据完成 T 和 W 变换, 进入 T′ 和 W′ 时, 第 2 组数据进入 T 和 W; 第 1 组数据进入 T_2 时, 第 2 组数据进入 T′ 和 W′, 第 3 组数据进入 T 和 W. 以此类推, 直至所有数据处理完毕. 宏观上从第 1 组数据产生输出开始, 实现每一个时钟有一组数据产生结果, 直至全部数据处理完毕.

5 加法数据的剪辑技术

三值光学计算机以数据位众多为特长, 适合处理超长数据和大量数据, 它可以随时为特定用户构造专用的加法器. 但每个用户要处理的数据长度和数据量是不同的, 若为每一个用户都去重构光学处理器, 又会花费很多时间和资源. 针对这种情况, 三值光学计算机采取的应对策略是: 常备一个规模适中的 MSD 加法器为大多数用户服务, 必要时再为特殊用户随时构造专用的 MSD 加法器.

为发挥常备 MSD 加法器数据位数多和流水计算方式两个优势, 要求待处理的数据要有适当的大小和较多的个数. 因此, 把一个大数据截断成几个长度适合常备加法器的数据段, 而把几个小数据拼接成一个长度适合常备加法器的数据链, 是支持常备加法器高效率工作的前提. 鉴于 MSD 加法中数据位之间无进位相关性, 对数据的截断和拼接 —— 数据剪辑, 必定可行. 但 T 变换和 T' 变换的结果都要在尾部附加一个 0, 这要求数据剪辑时要有一点技巧, 下面分别讨论之.

5.1 数据的截断

设一对大数据 a 和 b 有 $n+1$ 位, 计算 $c=a+b$:

$$a = a_n a_{n-1}\cdots a_k a_j a_i a_h \cdots a_1 a_0,$$
$$b = b_n b_{n-1}\cdots b_k b_j b_i b_h \cdots b_1 b_0.$$

直接对 a 和 b 采用 MSD 加法, T 和 W 变换后有

$$t = t_n t_{n-1}\cdots t_k t_j t_i t_h \cdots t_1 t_0 0,$$
$$w = 0 w_n w_{n-1}\cdots w_k w_j w_i w_h \cdots w_1 w_0.$$

T′ 和 W′ 变换后有

$$t' = t'_{n+1} t'_n t'_{n-1}\cdots t'_k t'_j t'_i t'_h \cdots t'_1 t'_0 0,$$
$$w' = 0 w'_{n+1} w'_n w'_{n-1}\cdots w'_k w'_j w'_i w'_h \cdots w'_1 w'_0.$$

T_2 变换有

$$c = c_{n+2} c_{n+1} c_n c_{n-1}\cdots c_k c_j c_i c_h \cdots c_1 c_0. \tag{2}$$

现在把 a 和 b 从 i 和 j 位之间截断成两组数据段为

$$a^1 = a_n a_{n-1}\cdots a_k a_j; \quad a^2 = a_i a_h \cdots a_1 a_0;$$
$$b^1 = b_n b_{n-1}\cdots b_k b_j; \quad b^2 = b_i b_h \cdots b_1 b_0;$$

对二者同时进行 T 和 W 变换, 有

$$t^1 = t_n t_{n-1}\cdots t_k t_j 0; \quad t^2 = t_i t_h \cdots t_1 t_0 0;$$
$$w^1 = 0 w_n w_{n-1}\cdots w_k w_j; w^2 = 0 w_i w_h \cdots w_1 w_0;$$

T′ 和 W′ 变换后有

$$t'^1 = t'_{n+1} t'_n t'_{n-1}\cdots t'_k t'^*_j 0; \quad t'^2 = t'_{i+1} t'_i t'_h \cdots t'_1 t'_0 0;$$

$$w^{'1}=0w'_{n+1}w'_nw'_{n-1}\cdots w'_kw^{'*}_j;\quad w^{'2}=0w'_{i+1}w'_iw'_h\cdots w'_1w'_0;$$

T_2 变换有

$$c^1=c_{n+2}c_{n+1}c_nc_{n-1}\cdots c^*_kc^*_j\quad c^2=c_{i+2}c_{i+1}c_ic_h\cdots c_1c_0. \tag{3}$$

对比 (2), (3) 两式可以看到, 在 c_0 到 c_{n+2} 的序列里它们有两个差异: 第 1 个是 (3) 式中多了 c_{i+2},c_{i+1} 两个数, 第 2 个是 $c^*_kc^*_j$ 和 c_kc_j 的值不一样. 在把 c^1 和 c^2 拼接成 c 时, 必须修正这两个差异, 修正第 1 个差异很容易, 剪去 $c_{i+2}c_{i+1}$ 即可. 修正第 2 个差异, 则需要在 a^1 和 b^1 的最后分别重复 a^2 和 b^2 的头两个数字, 然后在 c^1 的尾部剪去 $c^*_kc^*_j$ 即可. 这个技术可命名为 “断点重复, 接口剪齐”. 说明如下:

取

$$a^1=a_na_{n-1}\cdots a_ka_ja_ia_h;\quad a^2=a_ia_h\cdots a_1a_0;$$
$$b^1=b_nb_{n-1}\cdots b_kb_jb_ib_h;\quad b^2=b_ib_h\cdots b_1b_0;$$

则

$$\begin{aligned}
&t^1=t_nt_{n-1}\cdots t_kt_jt_it_h0;\quad t^2=t_it_h\cdots t_1t_00;\\
&w^1=0w_nw_{n-1}\cdots w_kw_jw_iw_h;\quad w^2=0w_iw_h\cdots w_1w_0;\\
&t^{'1}=t'_{n+1}t'_nt'_{n-1}\cdots t'_kt'_jt'_it^{'*}_h0;\quad t^{'2}=t'_{i+1}t'_it'_h\cdots t'_1t'_00;\\
&w^{'1}=0w'_{n+1}w'_nw'_{n-1}\cdots w'_kw'_jw'_iw^{'*}_h;\quad w^{'2}=0w'_{i+1}w'_iw'_h\cdots w'_1w'_0;\\
&c^1=c_{n+2}c_{n+1}c_nc_{n-1}\cdots c_kc_jc^*_ic^*_h;\quad c^2=c_{i+2}c_{i+1}c_ic_h\cdots c_1c_0.
\end{aligned}\tag{4}$$

由 (4) 式可见, 剪去 $c^*_ic^*_h$ 和 $c_{i+2}c_{i+1}$, 然后将 c^2 拼接在 c^1 后面就得到 c. 因此, 可以用 “断点重复, 接口剪齐” 技术把一个大数据截断成几个长度适当的数据段排入流水计算队列.

5.2 数据的拼接

当用户有很多位数少于 MSD 加法器位数 1/2 的小数据时, 把几个小数据拼接成一个略少于 MSD 加法器位数的数据链, 会提高加法器数据位的利用率, 从而节约计算时间. 技术细节如下:

设两对小数据为

$$a^3=a^3_na^3_{n-1}\cdots a^3_1a^3_0;\quad a^4=a^4_ma^4_{m-1}\cdots a^4_1a^4_0;$$
$$b^3=b^3_nb^3_{n-1}\cdots b^3_1b^3_0;\quad b^4=b^4_mb^4_{m-1}\cdots b^4_1b^4_0;$$

分别进行 T 和 W 变换后有

$$t^3=t^3_nt^3_{n-1}\cdots t^3_1t^3_00;\quad t^4=t^4_mt^4_{m-1}\cdots t^4_1t^4_00;$$
$$w^3=0w^3_nw^3_{n-1}\cdots w^3_1w^3_0;\quad w^4=0w^4_mw^4_{m-1}\cdots w^4_1w^4_0;$$

T' 和 W' 变换后有

$$t^{'3}=t^{'3}_{n+1}t^{3'}_nt^{3'}_{n-1}\cdots t^{3'}_1t^{3'}_00;\quad t^{4'}=t^{4'}_{m+1}t^{4'}_mt^{4'}_{m-1}\cdots t^{4'}_1t^{4'}_00;$$

$$w^{3'}=0w^{3'}_{n+1}w^{3'}_{n}w^{1'}_{n-1}\cdots w^{3'}_{1}w^{3'}_{0};\quad w^{4'}=0w^{4'}_{m+1}w^{4'}_{m}w^{4'}_{m-1}\cdots w^{4'}_{1}w^{4'}_{0};$$

T_2 变换有

$$c^3=c^3_{n+2}c^3_{n+1}c^3_nc^3_{n-1}\cdots c^3_1c^3_0;\quad c^4=c^4_{m+2}c^4_{m+1}c^4_mc^4_{m-1}\cdots c^4_1c^4_0. \tag{5}$$

假如在第 2 组数据的前面增加两个 0, 再和第 1 组数据拼接成大数据有

$$a=a^3_na^3_{n-1}\cdots a^3_1a^3_000a^4_ma^4_{m-1}\cdots a^4_1a^4_0;$$
$$b=b^3_nb^3_{n-1}\cdots b^3_1b^3_000b^4_mb^4_{m-1}\cdots b^4_1b^4_0;$$

T 和 W 变换后有

$$t=t^3_nt^3_{n-1}\cdots t^3_1t^3_000t^4_mt^4_{m-1}\cdots t^4_1t^4_00;$$
$$w=0w^3_nw^3_{n-1}\cdots w^3_1w^3_000w^4_mw^4_{m-1}\cdots w^4_1w^4_0.$$

t 和 w 中部的两个 0 来源于 T 和 W 都有对于输入 (0,0), 输出 0 的特性. t 中的前一个 0 正好起到要给未拼接前的 t^3 尾部补 0 的作用, w 中的后一个 0 起到给 w^4 前部补 0 的作用.

T′ 和 W′ 变换后有

$$t'=t^{3'}_{n+1}t^{3'}_nt^{3'}_{n-1}\cdots t^{3'}_1t^{3'}_00t^{4'}_{m+1}t^{4'}_mt^{4'}_{m-1}\cdots t^{4'}_1t^{4'}_00;$$
$$w'=0w^{3'}_{n+1}w^{3'}_nw^{3'}_{n-1}\cdots w^{3'}_1w^{3'}_00w^{4'}_{m+1}w^{4'}_mw^{4'}_{m-1}\cdots w^{4'}_1w^{4'}_0.$$

t' 和 w' 中部的 0 来源于 T′ 和 W′ 都有对于输入 (0, 0), 输出 0 的特性. t' 中的 0 正好起到给未拼接前的 $t^{3'}$ 尾部补 0 的作用, w' 中的 0 正好给未拼接前的 $w^{4'}$ 前部补 0 的作用.

T_2 变换后有

$$c=c^3_{n+2}c^3_{n+1}c^3_nc^3_{n-1}\cdots c^3_1c^3_0c^4_{m+2}c^4_{m+1}c^4_mc^4_{m-1}\cdots c^4_1c^4_0. \tag{6}$$

对比 (5) 和 (6) 式知, 只要在 (6) 式给出的 c 之后部, 剪下长度为 a^4 位数加 2 的段, 就得到 c^4, 剩余部分就是 c^3. 这项技术不妨称为 “双零拼接, 加二剪开”.

数据剪辑技术由三值光学计算机监控系统软件的一个模块完成, 这个监控模块将根据用户输入的数据对数据进行自动剪辑, 并排入加法器的输入队列, 然后对计算结果进行相应的反向剪辑. 整个过程对用户透明.

6 模拟实验

鉴于可实现 MSD 加法的三值光学计算机千位实验系统还在建设中, 近日, 用电子计算机对 MSD 加法、数据剪辑技术和流水计算技术进行了模拟实验, 实验结果完全证实了上述理论. 模拟实验的要点如下.

6.1 实验规划

① 确定 MSD 加法器: 从适合论文规模和证明本文结论两方面考虑, 用软件模拟一个 15 位的 MSD 加法器. 为此, 软件中包含模拟 15 位 T 和 15 位 W 变换的程序模块、16 位 T′ 和 16 位 W′ 变换的模块以及 17 位 T_2 变换的模块, 这 5 个变换模块构成 MSD 加法器模拟器; 软件中定义了一个 79 单元

表 2 实验用例

Table 2 Test case

	1	2	3	4	5	6	7
a	1011 0101	101 1100 1001 1110	110	101	1010	10 1111 0111 0101 1011	1 1001
b	0100 1101	111 0110 0000 1111	101	011	1110	01 1000 0111 0110 0111	1 1100
c	01 0000 0010	0 1101 01$\bar{1}$1 $\bar{1}$010 111$\bar{1}$	0 110$\bar{1}$	0 1000	01 1000	0100 100$\bar{1}$ 1110 1100 0010	011 1$\bar{1}$1$\bar{1}$

的一维整型数组 $D[79]$, 用来存放各变换模块的结果, 其中, T 的结果位于 $D[0]$ 到 $D[14]$ 单元, W 的结果位于 $D[15]$ 到 $D[29]$, T′ 的结果位于 $D[30]$ 到 $D[45]$, W′ 的结果位于 $D[46]$ 到 $D[61]$, T_2 的结果, 也是最终结果 c, 位于 $D[62]$ 到 $D[78]$.

② 准备实验用例: 两个 8 位数相加, 两个 15 位数相加, 两个 3 位数相加, 两个 3 位数相加, 两个 4 位数相加, 两个 18 位数相加, 两个 5 位数相加. 具体值见表 2 中 a 和 b 两行.

6.2 模拟软件的策略要点

模拟软件用 Dev C++ 编写, 数据输入输出采用文件方式, 按 6.1 小节第 ① 条的规划定义数组 D, 并定义一个数组 weishu [10] 用于记录各数据的编辑状况. weishu[i]=0 表示对应数据未经编辑; 0 < weishu[i] < 15 表示对应数据与相邻数据链接成一个大数据, weishu[i] 值是对应数据的位数; weishu[i] =15 表示对应数据是被切分的大数据的第一个片段, 最低的 15 位数; weishu[i]=16 表示对应数据是被切分的大数据的一个中间片段; weishu[i]=17 表示对应数据是被切分的大数据的最后一个片段, 最高的若干位. 在流水计算过程中, i 值从 0 开始, 逐次加 1.

输入数据的编辑策略有

① 当前数据的位数小于 16, 但加上后边数据的位数后超过 13 位, 且 weishu[$i-1$] 小于 15 时: 数据送入加法器, 并给 weishu[i] 单元赋 0.

② 当前数据的位数加上后边数据的位数后不足 14 位, 且 weishu[$i-1$] 为 0 或 17 时: 当前数据放入临时数据链的最低部分, 当前数据的位数值写入 weishu[i] 单元.

③ 当前数据的位数加上临时数据链的位数后仍不足 14 位, 且 weishu[$i-1$] 不为 0 也不为 17 时: 当前数据后加两个 0, 然后连接到临时数据链中已有数据的前面, 当前数据的位数值写入 weishu[i] 单元.

④ 当前数据的位数加上临时数据链的位数后超过 13 位, 且 weishu[$i-1$] 小于 15 但不为 0 时: 将 weishu[$i-1$] 改为 0, 并把临时数据链送入加法器. 然后对当前数执行策略 ① 或 ② 或 ④.

⑤ 当前数据的位数超过 15 位, 且 weishu[$i-1$]=0 时: 在第 15 位和第 16 位之间插入第 14 位和第 15 位的值, 低 15 位数据作为一个数据段送入加法器, 剩余部分作为下一个当前数据, 在 weishu[i] 单元写入 15.

⑥ 当前数据的位数超过 15 位, 且 weishu[$i-1$]=15 或 16 时: 在第 15 位和第 16 位之间插入第 14 位和第 15 位的值, 低 15 位数据作为一个数据段送入加法器, 剩余部分作为下一个当前数据, 在 weishu[i] 单元写入 16.

⑦ 当前数据的位数不足 15 位, 且 weishu[$i-1$]=15 或 16 时: 当前数据段送入加法器, 并在 weishu[i] 单元写入 17.

计算结果的编辑策略有

① weishu[i]=0 时: c 是计算结果, 将其输出.

② weishu[i] 不是 0 且小于 15 时: 在当前 c 的后部剪下 weishu[i] 值加 2 长度的数据, 剪下的数据是计算结果, 输出之; $i = i + 1$ 并对 c 的剩余部分重复本策略, 直到 c 被分配完, 此时的 weishu[i]=0.

③ weishu[i]=15: 当前 c 剪去最高两位, 剩余部分为计算结果的最低 15 位.

④ weishu[i]=16: 当前 c 剪去最低两位和最高两位, 剩余部分接续到计算结果的前边形成新的计算结果.

⑤ weishu[i]=17, 当前 c 剪去最低两位, 剩余部分接续到计算结果的最前边, 这时的计算结果是原大数据的最终结果.

6.3 模拟实验结果

模拟实验的结果见表 2 的 c 行, 计算结果完全正确, 数据剪辑技术成功.

详细分析每一步计算后的 D[79] 值, 可以确定流水计算技术也正确. 鉴于这个分析需要较多篇幅, 本文略去.

7 结论

三值光学计算机采用 MSD 加法器获得了加法运算过程无进位的好处, 使它的众多数据位之优势得以充分发挥. 流水计算技术的引入和数据剪辑技术的发明, 极大地提高了常备 MSD 加法器的效率. 本文描述的 MSD 加法器标志着三值光学计算机研究进入了数值计算领域, 是三值光学计算机步入成熟的又一个里程碑. 它为开发乘法例程、除法例程和矩阵运算的各种例程打下了基础.

参考文献

1 Jin Y, He H C, Lu Y T. Ternary optical computer principle. Sci China Ser F-Inf Sci, 2003, 46: 145–150

2 Jin Y. Management strategy of data bits in ternary optical computer. J Shanghai Univ (Nat Sci), 2007, 13: 519–523

3 Yan J Y, Jin Y, Zuo K Z. Decrease-radix design principle for carrying/borrowing free multi-valued and application in ternary optical computer. Sci China Ser F-Inf Sci, 2008, 51: 1415–1426

4 Jin Y, He H C, Ai L R. Lane of parallel through carry in ternary optical adder. Sci China Ser F-Inf Sci, 2005, 48: 107–116

5 Wang X C, Peng J J, Jin Y, et al. Vector-matrix multiplication based on ternary optical computer. In: the 2nd International Conference on High Performance Computing and Application (HPCA2009). LNCS, 5938. Shanghai, 2010. 426–432

6 Teng L, Peng J J, Jin Y, et al. A cellular automata calculation model based on ternary optical computer. In: the 2nd International Conference on High Performance Computing and Application, HPCA2009. LNCS, 5938. Shanghai, 2010. 377–383

7 Zhan X Q, Peng J J, Jin Y, et al. Static allocation strategy of data-bits of terry optical computer. J Shanghai Univ(Nat Sci), 2009, 15: 528–533

8 Avizienis A. Signed-digit number representations for fast parallel arithmetic. IRE Trans Electron Comp, 1961, EC: 389–400

9 Ha B, Li Y. Parallel modifed signed-digit arithmetic using an optoelectronic shared content-addressable-memory processor. Appl Opt, 1994, 33: 3647–3662

10 Casasent D, Woodford P. Symbolic substitution modifed signed-digit optical adder. Appl Opt, 1994, 33: 1498–1506

11 Li G Q, Qian F, Ruan H, et al. Compact parallel optical modifed-signed-digit arithmetic-logic array processor with electron-trapping device. Appl Opt, 1999, 38: 5038–5045

Principles and construction of MSD adder in ternary optical computer

JIN Yi[1*], SHEN YunFu[1], PENG JunJie[1], XU ShiYi[1], DING GuangTai[1], YUE DongJian[1] & YOU HaiHang[2]

1 *School of Computer Engineering & Science, Shanghai University, Shanghai* 200072*, China;*
2 *National Institute for Computational Sciences, Oak Ridge, Tennessee,* TN37831*, USA*
*E-mail: yijin@shu.edu.cn

Abstract The two remarkable features of ternary values and a massive unit with thousands bits of parallel computation will make the ternary optical computers (TOC) with modified signed-digit (MSD) adder more powerful and efficient than ever before for numerical calculations. Based on the decrease-radix design presented previously, a TOC can satisfy either a user requiring huge capacity for data calculations or one with a moderate amount of data, if it is equipped with a prepared adder. Furthermore, with the application of pipelined operations and the proposed data editing technique, the efficiency of the prepared adder can be greatly improved, so that each calculated result can be obtained almost within one clock cycle. We are hopeful that by employing an MSD adder, users will be able to enter a new dimension with the creation of a new multiplier, new divider, as well as new matrix operator in a TOC in the near future.

Keywords ternary optical computer (TOC), modified signed-digit (MSD), adder, pipeline computing, data editing technique