并行算法的 FP 描述及其脉动化的判定

胡振江 孙永强

(上海交通大学计算机系, 200030)

DESCRIPTION OF PARALLEL ALGORITHMS BY FP AND DECIDABILITY OF ITS SYSTOLIC IMPLEMENTATION

Hu Zhenjiang and Sun Yongqiang

(*Department of Computer Science, Shanghai Jiao Tong University, 200030*)

Abstract This paper extends the expressive power of FP by introducing stream and recursive equations on stream for the description of parallel algorithms with cyclic data dependency and states. The decidability theorem of about whether the FP program has its systolic implementation is given. This research also shows that the systoliclizing method on graph can be easily used on FP description.

摘要 本文通过引入流及流上的递归方程，增强了 FP 表达并行算法的能力，有效地克服了用 FP 描述循环数据依赖及状态记忆的困难. 文中给出了并行算法的 FP 描述及其可脉动化的判定定理. 同时说明，许多在图上研究的脉动方法可以方便地应用到用 FP 描述脉动化的研究中去.

§1. 引 言

脉动阵列 (systolic array) 是 H. T. Kung 首先提出来的 [2], 它是一个具有同步性，规整性，时态局部性和高效并行性的计算网络. 脉动阵列在模式识别，图形处理，矩阵运算和符号运算方面有了广泛的应用 [2]. 现在人们热衷于从研究并行算法的行为描述系统地导出其脉动阵列的实现的方法 [11].

1990 年 5 月 18 日收到, 1990 年 8 月 15 日定稿. 作者 胡振江, 1990 年在上海交通大学获硕士学位，现为博士生，目前主要从事脉动阵列机函数式语言方面的研究工作. 孙永强, 教授，博士导师，主要从事计算理论，新型语言方面的研究工作.

系统地从并行算法的行为描述综合出脉动阵列的实现最关心的问题是描述语言的设计. 并行算法的描述语言大致可分三类：第一类是数学语言，它用代数方程或差分方程 (如 URE, ARE) 来描述[3,4]; 第二类是图形语言，如信号流图 SFG(Signal Flow Graph)[7]; 第三类是用计算机语言[5,8,12,13,14], 如过程式语言，逻辑式语言，函数式语言或算子演算. 用计算机语言作为描述语言便于在机器上实现. 过程式语言语义复杂，不利于推理和变换，但易于模拟；逻辑式语言适合于推理；算子的演算比较适合于系统的推理 (reasoning) 和证明；函数式语言适合于变换. 由于脉动阵列的综合是一个变换系统，又函数式语言在行为级，体系级和结构级具有统一的代数基础. 因此，我们选择函数式语言作为我们的描述语言.

M. Sheeran[8] 第一个将 FP 用于脉动阵列的描述和综合. 她充分挖掘高阶函数的几何意义并加入许多高阶算子来研究脉动描述及脉动方法. Y. C. Lin[13] 认为用 aFP 描述算法本身就是从并行的思想入手，人们根据高阶函数的组合及作用的模式 (顺序输入，并行输入或 skew 输入) 导出一维脉动阵列的实现. W. Luk[12] 介绍了从一般规范描述综合出 FP 程序的方法.

但以上一些方法存在的共同缺陷是所采用的是 ad hoc 方法：究竟设计出来的 FP 并行表达能力如何？怎样的 FP 描述是可以脉动化的？如何系统地进行脉动化？对于这些问题他们都没有回答. 另外，他们无法描述具有循环数据依赖的并行算法.

本文通过引入流及流上的递归方程，增强了 FP 表达并行算法的能力，有效地克服了用 FP 描述循环数据依赖及状态记忆的困难. 文中给出了并行算法的 FP 描述及其可脉动化的判定定理. 同时说明，许多在图上研究的脉动方法可以方便地应用到用 FP 描述脉动化的研究中去.

§ 2. 并行算法的 FP 描述

进程是并行算法中的一个重要概念. FP 要能描述并行算法的关键问题是如何用函数描述进程.

进程的一般结构是：输入流 ⟶ 进程处理 (内部状态)⟶ 输出流

这里的输入流和输出流是一个与时间相关的量. 在某一时刻，输出和输入之间不构成函数对应关系，因为输出不仅与输入有关，而且和当前的内部状态有关. 但是，设起始状态为进程处理的不关心状态 (常用？或 – 表示)，则纵观历史，输出流和输入流间形成函数对应.

2.1 流的引入

流是一个无限的序列，我们用下述符号定义一个流： STREAM≡<S<1>, S<2>, ⋯, S<n>, ⋯ > *, 其中 (1) 每个 S<i> 的类型都是一样的， (2) S<i+1>=O(S<i>) 即 S<i> 的下一个节拍的值是 S<i+1>. 整个流是在一个全局时钟的控制下工作的.

例如： <1, 2, 3, ⋯ > * 表示一自然数流； <C, C, C, ⋯ > * 表示一常数 C 的流.

在流上引进如下的操作：

(1) D^i 操作

设一流 <X1, X2, ⋯ > *, 则 D^i: Xj=Xj+i (i>=0)

D 操作具有下述代数性质：

* D^i 。 D^j=D^(i+j),

* 设 f 是流上的函数，则 f 。 D^i=D^i 。 f.

注： f^i=f 。 f 。 ⋯ 。 f, 其中 f 的个数为 i.

(2) D−i 操作

设一流 STR=<X1, X2, ⋯ > *, 则 D−i: STR=<X1, −, −, ⋯, X2, −, −, ⋯ > *

它的直观意义是将 STR 流的流速放慢 i 倍，上述定义中，符号 − 为与 Xi 类型相同的不关心值， Xi 与 Xi+1 之间经 D−i 的作用后插入 i−1 个 −.

2.2 流上的 FP–FPS 及其语义描述

流上的 FPS 是在 BACKUS 的 FP[1] 的基础上加以扩充而得. 它加入了流对象以及流上的函数 D^i 和 D−i. FPS 不仅能表达序列上的递归行为，而且可表达时序上的递归行为 (见 2.3 节). 同时，它保持原 FP 上良好的代数性质.

为了描述 FPS 的语义，我们将流映射为 Backus'FP 上的无穷序列，用 FP 来描述 FPS 的操作语义. 设 SF: FPS− >FP 是一个语义函数，它将流上的 FPS 程序用序列上的 FP 来解释.

(1) SF 〖 f 〗 =αf, 如果 f 为原 FP 上的基本函数；

(2) SF 〖 f 。 g 〗 =SF 〖 f 〗 。 SF 〖 g 〗；

(3) SF 〖 f, g 〗 =zip 。 [SF 〖 f 〗， SF 〖 g 〗];

(4) SF 〖 p− >q; r 〗 =α(1− >2; 3) 。 zip 。 [〖 p 〗， 〖 q 〗， 〖 r 〗];

(5) SF 〖 D^i 〗 =sl^i;

(6) SF 〖 F: x 〗 =SF 〖 F 〗: 〖 x 〗；其中， 〖 x 〗 将 x 映射为 FP 上的无穷序列.

其中， zip:<<x11, x12, ⋯ >, <x21, x22, ⋯ >, ⋯, < xn1, xn2, ⋯ >>=<<x11, x21, ⋯, xn1>, <x12, x22, ⋯, xn2>, ⋯ >, sl: <x1, x2, ⋯ >=< −, x1, x2, ⋯ >

对于两个 FPS 程序 PROG1 和 PROG2, PROG1 和 PROG2 等价 (记为 PROG1=PROG2) 定义为： SF 〖 PROG1 〗 =SF 〖 PROG2 〗.

2.3 并行算法的 FPS 描述

FPS 在并行算法的表达上与 FP 相比，具有以下几个特点：

(1) FPS 能方便地描述并行性，如 [f1, f2, ⋯, fn] 描述了 f1, f2, ⋯, fn 可同时进行 (FP 中亦可); 同时 FPS 又能方便地描述流水性，如 f1 。 D 。 f2 描述了 f1 与 f2 间的流水性.

(2) FPS 能方便地描述不同时刻数据之间的依赖关系. 原 FP 描写的是静态作用在某数据上的递归行为. 但是 FPS 描写的递归行为则有两方面涵义：一是横向的递归描述，这与原 FP 描述方式类似；二是纵向的递归描述，即描述不同时刻数据上的递归关系.

[例 1] 矩阵相乘的 FPS 描述

设 A=(A1, A2, ⋯, An), B=(B1, B2, ⋯, Bn) 是两个矩阵， A 矩阵中 Ai 表示第 i 列，B 矩阵中 Bi 表示第 i 行。其乘积 C=A*B=ΣAi*Bi. 将 A, B 看作两个流 (A1, A2, ⋯)*, (B1, B2, ⋯)*, 则矩阵相乘的 FPS 描述为：

MM=α(α+) 。 α trans 。 trans 。 [D 。 MM, α(α*) 。 α distr 。 distl]

上式中 MM=E(D 。 MM) 的递归式描述了累计和的过程.

MM=E(D 。 MM) 表达了流上的递归行为. 为了便于以后的讨论，我们给出流递归方程的一般定义：

Cf=E(f, D^i 。 Cf) (i≥1)

定义中，当 f 为常函数时，高阶流递归方程变为一阶流递归方程：

C=E(D^i 。 C) (i≥1)

由定义可见，流递归方程与原 FP 的一般递归方程的不同点在于：流递归方程中被递归定义的函数名在定义体中都是以和 D^i 复合的情形出现或可变换成这种情形.

(3) FPS 描述组合反馈及状态记忆更加直接. M. Sheeran[8] 为了描述这种行为，引进了 μ 算子及其复杂的代数运算. μ 算子的定义为：设一流 <X1, X2, ···> *,

如果 f: <<X1, ?>, <X2, S1>, ···> *=<<01, S1>, <02, S2>, ···> *

则 μ f: <X1, X2, ···> *=<01, 02, ···> *.

μ 算子对应的 FPS 描述为：

Cf=2 。 f 。 [id, D 。 Cf];

μf=1 。 f 。 [id, D 。 Cf];

(4) FPS 解决了循环数据依赖的描述问题。双向数据依赖结构甚至复杂的循环数据依赖关系用 FP 描写是非常棘手的，但采用流递归方程后，便可迎刃而解。如图 2–1 的结构，图中的数字及其对应的箭头线表示相应函数第几个输入 (或输出), 例如，f 函数有三个输入分别对应于三个输入箭头。我们将 f, g 函数的双向依赖连线分别定义为高阶函数 C[f, g] 和 D[f, g], 其定义如下：

C[f, g]=1 。 g 。 [D[f, g], 2 。 2, 3]

D[f, g]=2 。 f 。 [1, 1 。 2, D 。 C[f, g]]

采用代入法得：

C[f, g]=1 。 g 。 [2 。 f 。 [1, 1 。 2, D 。 C[f, g]], 2 。 2, 3]

D[f, g]=2 。 f 。 [1, 1 。 2, D 。 1 。 g 。 [D[f, g], 2 。 2, 3]]

则图 2–1 的双向结构描述为：

F[f, g]=[1 。 f 。 [1, 1 。 2, D 。 C[f, g]], 2 。 g 。 [D[f, g], 2 。 2, 3]]

进一步发展，高阶函数 β 定义了图 2–2 的结构，其 FPS 描述为：

βf=eq 。 [length 。 2, ~2]– >F[f, f]; F[f, β f] 。 [1, [hd, tl] 。 2, 3]

注： 除特别说明，下面讨论的 FP 就是指 FPS.

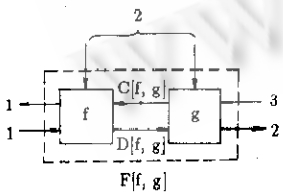


图 2–1 f 和 g 的双向依赖结构

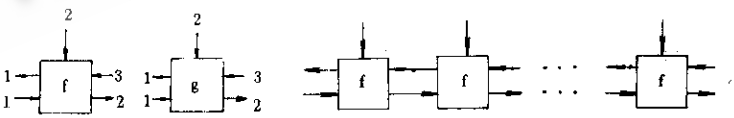


图 2–2 若干 f 的双向依赖结构

§ 3. 可脉动 FP 程序的判定

FP 描写的并行算法很多，究竟怎样的 FP 描述存在它的脉动化表达，为此，我们首先给出脉动 FP 表达式的定义，为了寻求判定方法，我们将 FP 程序转化为容易处理的中间形式——函数作用图，利用图上的现有的结论来进行判定.

为了便于讨论，我们将递归方程进行分类：

1) 数值上可展开的递归方程：这类递归方程的特点是方程展开的规模与输入参数的具体值有关，例如.

fac=eq 。 [id, ~0]– >~1; * 。 [fac 。 – 。 [id, ~1], id]

这类递归方程不适合用固定规模的硬件来实现，其内在的流水性和并行性只适合于在一个处理单元上运算. 因此，这类递归方程定义的函数常被看作某个单元函数 (cell function).

2) 序列上可展开的递归方程：这类递归方程的特点是展开规模取决于输入参数的模式。例如，内积的定义如下：

IP 。 [[x], [y]]=* 。 [x, y]

IP 。 [x|xx, y|yy]=+ 。 [* 。 [x, y], IP 。 [xx, yy]]

IP 对于 <<x1, ···, xn>, <y1, ···, yn>> 的输入模式其硬件实现与具体的 xi, yi 的值无关。这类递归方程最适合于导出满足此运算的硬件体系结构，而且，这一体系结构往往是规整的，这是由原来算法的递归定义决定的。

3) 流上的递归方程：这在上节已讨论过，这类递归方程适合于描写算法的时序要求.

3.1 可脉动的 FP 程序的定义

设 SYS 是可脉动的 FP 程序集，则 SYS 的递归定义为：

1) 如果 f 是单元函数或基本操作函数 (如 +, – 等), 则 D^i 。 f∈SYS; (i>=1)

2) 如果 f∈SYS, l1, l2 是走线函数 (如 apndl, concat 等，但不包括 distl, distr)
则 f 。 l1, l2 。 f∈SYS;

3) 如果 f, g∈SYS, 则 f 。 g, [f, g]∈SYS;

4) 对于流递归方程 Sf=E(f, D^i 。 Sf), 如果将 Sf 理解为一单元函数时，
E(f, D^i 。 Sf)∈SYS, 那么 Sf∈SYS.

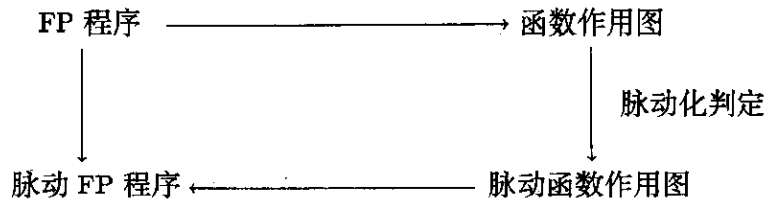
根据上面的定义，可导出高阶函数和序列上可展开的递归方程的对应的脉动表达式。例如，如果 f∈SYS, 则 /f, αf∈SYS.

一个 FP 程序 prog 在一定的输入模式 in 下可脉动化定义为：存在 prog_sys∈SYS, i, j∈N, 使得： SF 〖 D^j 。 prog 。 D–i: in 〗 =SF 〖 prog_sys 。 D–i: in 〗.

上述定义的直觉意义是： prog 与其脉动表达式 prog_sys 的行为在对输入进行 D–i(调时操作) 后， prog_sys 的输出结果比 prog 的输出结果延迟若干节拍.

3.2 可脉动的 FP 程序判定：

我们可按如下所示的方法进行判定：



我们首先在 FP 函数和图之间建立对应关系，从而利用图上的判定定理来研究 FP 上的可脉动化的判定问题.

一个 FP 程序在输入参数的模式确定之后，可通过高阶函数的展开和序列上递归方程的展开得到一个仅含单元函数，复合算子 o, 构造算子 [ ], 条件算子 – > 的流递归方程. 条件算子可用下式消除： p– >q; r=sel 。 [p, q, r], 其中 sel: <x, y, z>=y (如果 x=true), sel: <x, y, z>=z(如果 x=false).

FP 函数经过上面的变换后，化为图的算法如下：

1) 如果 f 函数为基本操作函数 (如 +, –, Sel 等) 或为数值上可展开的递归方程，则 f 对应一个结点 (图 3–2 ①); 另外，为了消除如 distl, distr 等函数在图中所表示的广播式，引进 \ 算子 [8], 其涵义如图 3–1 所示.

则 distl=\[id, 2] 。 apndl,

distr=distl 。 [2, reverse 。 1]

利用上面两个等式可消去 distl, distr. 此时，设函数 [id, 2] 对应一结点.

2) 如果 f=D^i, 则对应图 3–2 ②;

3) 如果 f, g 对应两个结点，则 f 。 g, [f, g] 分别与图 3–2 ③④对应;

4) 对于流递归方程 Sf=E(f, D^i 。 Sf), 当在 E 中 Sf 理解为走线函数时， E(f, D^i 。 SF) 对应图 E, 则 Sf 对应于图 3–2 ⑤.

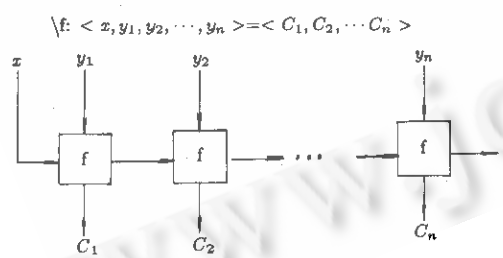


图 3–1　\ 算子的直观解释

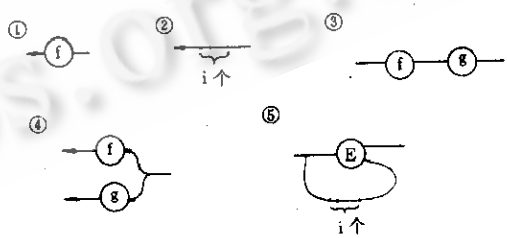


图 3–2　FP 函数与函数作用图的对应

对于矩阵相乘算法的 FP 描述当输入模式为 <<x1, x2, x3>, <y1, y2, y3>> 时，对应的函数作用图如图 3–3 所示.

引理 3–1: FP 程序对应的函数作用图是一个具有局部连接的确定性的图，图中不存在这样的环路，它的弧线上总的延迟 D 的和为零. (证略)

引理 3–2: 如果 prog∈SYS, 则 prog 对应的函数作用图中，每一弧线上至少有一个延迟 D. (证略)

引理 3–3: 如果图中不存在这样的环路，它的弧线上总的延迟 D 的和为零，则此图是可脉动化的．即图中每个弧线上至少有一个延迟 [9].

对于矩阵相乘函数作用图经脉动化后如图 3–4 所示．图上的脉动过程请参见 [9]. 最后，从脉动式图返回 FP 表达式是很直接的，因为脉动后的图和原来的图的差别仅在弧线上多了若干的 D, 所以我们只须在原 FP 程序的适当位置加入 D 函数，就可得到它的脉动表达式．至此，我们得出下面的定理．

定理 3–4: 如果 FPS 的程序中含有的递归方程仅为以上讨论的三种类型，则它是可脉动化的．（证略）

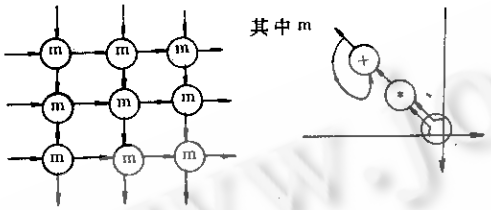


图 3–3　矩阵相乘算法的函数作用图

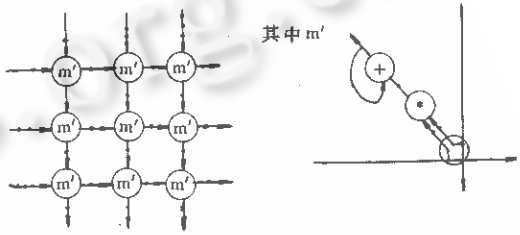


图 3–4　矩阵相乘脉动化后的函数作用图

§ 4. 结论

本文通过引入流及流上的递归方程，增强了 FP 表达并行算法的能力，有效地克服了用 FP 描述循环数据依赖及状态记忆的困难．文中给出了并行算法的 FP 描述及其可脉动化的判定定理．同时说明，许多在图上研究的脉动方法可以方便地应用到用 FP 描述脉动化的研究中去．

但是，图上的变换往往比较复杂，而 FP 程序具有良好的代数性质，在 FP 程序上进行保持正确性的代数变换比在图形上进行变换更为有效．限于篇幅，我们将在以后的文章中讨论．

参考文献

[1] J. Backus, "Can Programming be Liberated from the Von Neumann Style? A Functional Style and Its Algebra of Programs", CACM, No. 3, 1978.

[2] H. T. Kung, "Let's Design Algorithms for VLSI Systems", Tech. Report, Carnegie-Mellon Univ. TR–CMU–CS–79–151.

[3] U. Weiser, A. L. Davis, "Mathematical Representation for VLSI Arrays", Tech. Report, Utah Univ. TR–UU–CS–80–111.

[4] D. I. Moldovan, J. A. B. Forts, "Partitioning and Mapping Algorithms into Fixed Size Systolic Arrays", IEEE Trans. on Computer, Vol. c–35, No. 1, 1986.

[5] S. Y. Kung, K. S. Arum, "Wavefront Array Processor: Language, Architecture, and Application", IEEE Trans. on Computer, Vol. c–31, No. 11, 1982.

[6] P. Quiton, "Automatic Synthesis of Systolic Arrays from Uniform Recurrent Equations", Proc. IEEE,

1984.

[7] C. E. Leiserson, J. B. Saxe, "Optimizing Synchronous System", Tech. Report, Carnegie–Mellon Univ. TR–CMU–CS–82–101.

[8] M. Sheeran, "μFP– An Algebraic VLSI Design Language", Ph.D Thesis, Oxfors Univ. TM–PRG–39, Nov. 1983.

[9] S. Y. Kung, "On Supercomputing with Systolic/Wavefront Array Processors" Proc. IEEE Vol. 72, July, 1984.

[10] J. D. Ullman, "Computational Aspects of VLSI", Computer Science Press, 1983.

[11] J. A. B. Forts, K. S. Fu, "Systematic Approaches to the Design of Algorithmically Specified Systolic Arrays", Purdue Univ. TR–EE84–39.

[12] W. Luk, G. Jones, "The Derivation of Regular Synchronous Circuits" IEEE Int. Conf. on Systolic Array, 1988.

[13] Y. C. Lin, F. C. Lin, "The Use of A FP to Design Regular Array Algorithms" Proc. 1988 Int. Conf. On Comp. Lang., Oct. 1988.

[14] G. Milne, "CIRCAL: A Calculus for Circuit Description", Univ. of Edinburgh, 1982, Tech. Report, CSR–122–82.