国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
一九八七年第二期　总第五十八期　No.2 1987 Sum. 58

数据流与控制流相结合的计算模型与体系结构

（摘　要）

论文作者：刘桂仲
指导教师：慈云桂　教授
（计算机系）

一、引　　言

数据流计算模型是国外六十年代末期提出来的，至今，数据流计算机还没有诞生第一代产品。人们普遍认为数据流计算机技术上的主要问题是开销过高。

造成数据流计算机开销过大的基本原因是并行的级别太低，或者说并行的粒度（granularity）太细。一个系统中的并行可以划分为多个级别，如图1所示。在数据流计算模型中，没有这种明显的并行层次，各级并行往往都化为指令级并行来实现。正是这种不适当地依赖指令级并行便付出了过高的代价。例如：对数据结构数组进行处理时需要化整为零，因而困难重重。如果注意函数一级的并行，设计出各种处理数组的函数，如向量加、减、乘、除函数，利用数据驱动这些函数的执行，虽然好象牺牲了元素之间的并行加工，但如果采用流水线加工方式，则代价微不足道，并行性也可开发。

作业级
进程级
函数级
指令级
图 1

数据流计算模型操作开销大的另一个原因是没有控制，异步操作。任何实际的计算系统中物理资源总是有限的，数据流计算系统也不例外，这就限制了一些指令的同时执行，于是，在实际的系统中必需按照一定的策略选择部分指令优先执行。为此，数据流系统通常在各个部件中分散设置各种队列和缓冲器暂存请求服务的对象。但是，立足全局，从宏观分析，在大批可同时执行的指令中，哪些能优先执行，哪些能优先完成操作

本文1987年8月收到

便与多种偶然因素有关，因而是不确定的。由此可见这种分散控制无异于集中的随机控制策略。而随机控制策略显然不是好的策略。

为了克服数据流计算机的问题必须：

(1) 提高并行级别；

(2) 在保持并行的条件下引入必要的控制，减少指令级并行开销。

二、数据流与控制流相结合的计算模型

2.1 平衡的数据流图与一般的数据流图之比较

我们首先定义数据流机模型中几个我们关心的部件。

设有数据流图G，G中的结点记为ai，i=1,2,3,…,n. 它们就是在数据流机器上执行的指令，图中token记为tj，j=1,2,3,…,m。

定义 2.1 PM为数据流计算机中的程序存储器的抽象描述。PM为图G中的全部结点的集合，或者说是程序中的全部指令的集合。PM={a1，a2,…,an}，ai为G中的结点。

定义 2.2 TM是数据流机中的数据存储器的抽象描述。TM为图G中全部token的集合，TM={t1,t2,t3,…,tm}，ti为G中的token。

我们把图中的每一条弧都赋以token的标识，当某一时刻某一弧线上不存在数据token时，认为其token的标识取值为空。

通常，数据流机不一定将程序与数据分开存放。从下述分析可以看出，我们这种处理有利于揭露问题的本质。

数据流机的点火规则记为F。对于一般的数据流机，点火规则为判定指令的操作数是否到齐，如果到齐了，指令可点火执行，否则不执行。

设缓冲器记为BU，它存放可以点火的指令。从BU中取指令送入处理器执行的规则称为优先规则，记为S。对一般数据流机而言，优先规则可视为随机选取规则，即由BU中随机选取一批指令。

平衡的数据流图是一种特殊的数据流图，在这种图中从它的始结点至其它一个结点如果有多条路径，则它们的长度相等。

如果G是平衡的数据流图，可以定义下述两个关系：

定义 2.3 PM上的关系“=”如下述：

$$ap, aq \in PM,\ ap = aq \iff Level(ap) = Level(aq),$$

其中Level (a) 为结点a的级。

显然，PM上的关系“=”满足自反性、对称性和传递性，故“=”是PM上的等价关系，由“=”，可得到PM上的一个划分{PM0，PM1,…,PMn}，它满足：

$$PM = PM0 \cup PM1 \cup \cdots \cup PMn,$$

$$PMi \cap PMj = \phi,\ 0 \leqslant i, j \leqslant n.$$

且有 1. 如果ap，aq∈PMi，则ap=aq；

2. 如果ap∈PMi，aq∈PMj，i≠j，则ap≠aq。

定义 2.4 PM上的关系“<”如下述：

ap，aq∈PM，ap<aq<⟺>Level (ap) <Level(aq)。

显然，PM上的关系“<”满足非自反、非对称和传递性，故“<”是PM上的拟序关系。

图G上的任意一条弧线对应TM中的一个数据token。设（ap，aq）是G上的一条弧，则有一数据token由ap产生并传送至aq，所以图中的弧线与TM中的token一一对应，故弧线数就是TM的基数。TM的基数的大小反映了数据空间的大小。但一个数据token并不是自始至终都是必须的，它们有其生存期。所谓生存期是指token产生与消失的间隔时间。

一般的数据流图，不能给指令分级，因而不存在“<”序关系；再加之优先规则S是随机选取，因而指令执行序列无法预料，数据的生存期也就没有实际意义。故对一般的数据流图来说，如果它们不是流水线式的执行，它们所需要的数据存储空间就是TM的基数。

对平衡的数据流图，如果按“<”序执行，则情况大不相同。这时，数据是由某一级的结点产生，而为后级结点吸收，则弧线（ap，aq）上的token，其生存期可简单表示为（Level(ap)，Level(aq)）。

设有两个token的生存期分别为（i1，j1），（i2，j2）。所谓它们的生存期重叠是指它们满足下述条件：i2<j1且i1<j2。否则它们的生存期是不重叠的。

生存期不重叠的两个token，在时间上不同时存在，它们可以共用同一存储区域。因而，对按序执行的平衡数据流图，它的数据空间是可以压缩的。

定理 2.1 设G是可以按“<”序执行的数据流图，aij为第i级第j条指令，则G所需的数据空间大小为$MG=\max[\sum_j deg(a1j), \sum_j deg(a2j,\cdots,\sum_j deg(anj)]$，其中deg(aij)为结点aij的入度，n为G的最大级。

证明：（略）

一个非平衡的无圈图，可以变换为平衡的数据流图，因而所需的数据空间也不过是一级指令所需的数据空间。显然，MG≪|TM|，即：有序执行的平衡的数据流图所需的数据空间远远小于相应的一般的数据流图。

动态数据流图设置相联存储器，也缩小了数据空间。但这种方法不过是把大的数据逻辑空间映射到小的物理空间上。由于逻辑空间并未缩小，所以不得不采用相联访问的存储器，因而开销仍然不小。而有序的数据流图则是缩小数据逻辑空间，因而更为有效。

在按序执行的数据流图的模型中可以设置一个指针Point，它指向当前可执行的指令集PMi。当PMi中的指令执行完毕，即PMi=ϕ时，Point+1，指向下一级。这实际上是把点火规则修改为：

$$\forall aij\in PM,\ i=point \Longleftrightarrow aij\ 可点火。$$

称这一点火规则为按序点火规则。这种修改不仅仅是简化执行，省去了判定点火的操作，更重要的是无需将数据直接传送至指令。这样确实可以将数据空间与程序空间相分离，且因为数据空间可压缩，使我们可以利用小容量高速度的寄存器存放数据token，

因而使各种开销减小。

2.2 带控制标志的数据流图

2.2.1 建立有序模型的两个问题

一般的数据流图因为含圈不能表示为有序模型。这是必须克服的首要问题。另一问题是如果简单地按序执行指令，还会形成过多的冗余操作。因为2.1中的点火规则与数据到齐的点火规则并不等价，前者是后者的必要条件。数据流模型中只执行分支指令的一支，另一支因数据不到达，不会执行。但若按序执行则两支都要执行。因此，必需设法消除这些冗余操作。

2.2.2 改进的数据流图 DFGC

为改进数据流图，本文提出一种带控制标志的数据流图（Data FLOw Graph with Controlling Tag）简记为DFGC。这是一种与数据流图类似的有向图，但图中的标记 token 都增加了控制标志C。C可取真、假值。点火规则修改为当所需要的控制标志到达时，则该操作点火。图2是计算 if a＞b then a－b else b－a 的 DFGC。图中有一种独特的操作符，称为 Attach，它的作用仅仅是修改输入 token 的控制标志，即用＞的输出 token 的控制标志取代另一输入 token 的控制标志，形成 c′.a 和 c′.b，图中的运算操作要求两个输入控制标志为真。如果输入 token 的控制标志为假，该操作的作用仅仅是吸收输入 token。图中 Merge 是按照控制条件选择一个输入 token 作为输出。

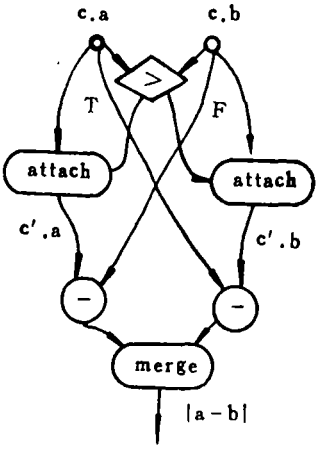

图 2

DFGC 的突出特点是显含了控制标志，其作用是控制各种操作的执行。就是说在 DFGC 中，是由 token 中的控制标志的到达来触发各种操作的点火。这是把原来数据流图中的 token 分成了两部分，分别完成原来 token 具有的两种职能，即传送数值和触发点火。这样做的结果带来了极大的灵活性，使得无法控制的数据流图，变得易于控制。

当一个 token 传送至一个 actor 时，即判断该 token 的控制标志是否为真，若是的，便继续等待其他输入 token；若不是的，则不进行操作。只是把一个数据域为空，控制标志为假的输出 token 放置在输出弧线上。这时，actor的作用相当于数据流图的sink。只有在输入 token 到齐，且它们的控制标志都为真时，才进行规定的操作。

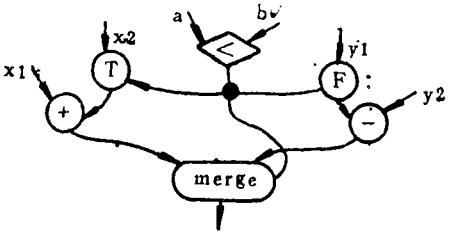

图 3

一般的数据流图在执行时，可能有一部分操作不能点火。例如在图 3 中，假定＜

结果为真，⊖的操作数 y2 可能会到达，而y1 因 F 门的控制信号为假，没有输出，其结果⊖不能点火。但这不影响这个图的执行结果。不过⊖的一个操作数 y2 因此便不能被吸收。如果后续数据再传送到这组操作，便会引起错误。对动态数据流机，y2 是暂时存放在相联存储器中，由于⊖不能点火，y2 将永远停留在相联存储器中。虽然不会引起错误，但不能回收存储器。如果这种多余数据增加，就会使相联存储器效率下降，甚至造成系统死锁。这种问题是实际的数据流机必然遇到的，而往往又为理论研究所忽略。我们把这种一部分操作不点火的情况称为不完全点火。此外，一般数据流图中点火规则对 merge 例外，它仅要求控制信号与相应的一个操作数到达。这一例外使实现复杂。它也是不完全点火的另一含意。在 DFGC 图中，点火的含意就是激活 (enable)，它与函数操作相区别。因此，其点火规则是完全的，即任何操作都全部点火，对 merge 操作要求其两个输入 token 都到达方能点火。但一个操作点火后不一定执行所规定的函数操作。这样即消除了不完全点火的问题，也没有引入冗余操作。

DFGC是有向图，可以含圈。我们希望构造的是它的一个子集，称为良结构的DFGC.

定义 2.5 良结构的 DFGC 是不含圈的 DFGC。

以下我们仅讨论良结构的 DFGC。如不加说明，DFGC 就是指良结构的 DFGC。

在良结构的 DFGC中，不允许含圈，也就不能实现循环结构。为了弥补这一不足，我们引入一种新的 actor—apply，称为作用。它的第一个输入是与一个特定的 DFGC 相关联的 token，它的 d 域是该 DFGC 的唯一标识符。

在引入 apply 操作以后，DFGC 的表达能力大为增强。可以实现函数调用，也可实现递归。虽然不能实现循环，但有了 apply 足以用递归表示循环迭代。这样，DFGC 的计算能力就不小于 DFG。

2.4 基于 DFGC 的同步与异步相结合的模型 SDS

本节以 DFGC 为基础建立一种同步与异步相结合的数据流系统模型(Synchronous Dataflow System) SDS，用做下一步高级数据流语言和体系结构设计的共同基础。

2.4.2 高层次模型——函数相关图

函数相关图是 SDS 中的高层次模型，目的是为了描述函数间的相关关系，以便开发函数一级的并行。函数相关图是一种特殊的 DFGC。图中的结点是复合函数。除此而外，与 DFGC 没有区别。

复合函数是将 DFGC中的基本操作经有限次运用复合、条件等四种构造进行组合，然后，用这些复合函数的名称替换相应的 DFGC子图，即构成了函数相关图。上述过程称为复合函数划分。

现已证明对没有副作用的函数语言，如果所有的函数都是严格函数，则这种语言具有所谓引用透明性，便可进行上述的代换，而不影响语义。

引入控制标志使我们可以比较有效地解决无定义值用于触发后续函数点火的问题。这样，就可以将原来的指令级并行的数据流图重新加以组合，构造成较大的执行单位——复合函数，以便开发高一级的并行。

2.4.3 低层次同步流水线 DFGC模型

在 SDS 的低层次，即在一个复合函数内部的指令级我们建立了同步的DFGC模型，

这是2.1中提出的有序数据流图的具体实现。该模型又分为流水线式和非流水线式的两种。本节介绍流水线式的模型。

一个复合函数的DFGC图，如果其中不含函数调用操作，则它的DFGC图可以变换为平衡的DFGC图。由于平衡的DFGC可以按序执行，因此，我们可以采用按序点火的规则，相当于同时点火一排指令，这是同步的意义。

我们可以设想一个二维处理器阵列，在这个阵列中处理器通过网络共享公共存储器。然后可以把平衡的DFGC映射到二维处理器阵列上。假定存储器与互联网络的延时趋于0；任何操作的延时都是相同的单位时间；全部处理器由一个公共的控制器控制。它自上而下，按序启动每排处理器，这就形成了整个部件在控制器管理下同步运行。这是同步运行的含意。当数据源源不断从上端流入时，各排处理器分别加工不同的数据，每一拍都有结果数据源源从下端流出。由此，形成了宏流水线。它既开发了并行，又保持了小的开销。这是流水线的意义。

2.4.4 低层次同步非流水线DFGC模型

对于不适合于同步流水线DFGC描述与执行的函数，我们采用同步非流水线DFGC模型。所谓同步非流水线DFGC模型就是按序逐级驱动点火的良结构的DFGC图，它与同步流水线DFGC模型的区别是允许DFGC是非平衡的，也允许函数调用，结点之间允许传名、传值或仅仅传控制。

这种同步模型从以下几方面有效地减少了操作开销：

1. 在DFGC图中，点火规则已修改为控制标志C到齐，指令可执行。同步模型采用按序点火规则。因为分支指令的两支都有控制标志送达，都可点火，又由于它们的控制标志不同，仅有其中的一支执行指令操作，另一支仅仅是将C值传送至下一级，并不执行指令，因而消除了冗余操作。

2. 点火规则是完全的，消除了不完全点火引起的一些实际问题。

3. 无需操作间的回答信号。

4. 数据传送仅仅在两级之间进行，故可引入中间工作寄存器，按名访问，甚至可按寄存器地址访问，无需在处理器和存储器之间传送中间数据，也不必采用相联匹配寄存器。

5. 指令中只需含操作码，操作数寄存器地址与目标寄存器地址，因而指令长度可压缩到与von Neumann机器相当。

6. 有了控制标志可消除无效操作。

数据可按名访问，故可反复使用。

三、函数语言VAL及SDS模型的抽象机

3.1 VAL语言与SDS模型的抽象机

VAL语言是美国MIT的Dennis等人基于数据流计算模型而设计的函数语言，它与SDS模型的要求基本一致，两者大体上是相适应的。SDS模型中可以恰当地表达无定义值，这为VAL语言中引入的多种错误值和例外处理提供了基础。两者尚存在不

相适应之处。主要问题是：SDS 是两层次模型，高层次是函数相关图，图中的结点是复合函数，这是一种比 VAL 中的操作大，比模块小的程序单位，即：一个模块并不对应一个复合函数，而是对应一个函数相关图，其中有多个复合函数。VAL 语言中的某些语句，如 forall，为开发并行必须变换为多个复合函数，以便并行执行。VAL 语言的最初文本中[7]，引入了循环迭代结构，这与 SDS 模型的要求相矛盾，为此取消了 VAL中的迭代，而容许递归。

我们设计的 VAL 语言的编译程序对程序的每个模块进行独立的编译，最后再链接为可运行的目标代码。编译程序的任务是把每个源程序模块变换为函数相关图，再把每个复合函数变换为 DFGC 图。一般说来，编译程序首先将源程序变换为原子操作构成的 DFGC图，再根据其结构，划分为不同的复合函数。在复合函数确定之后，再将其映象为同步流水线 DFGC 图或同步非流水线 DFGC 图。

3.2 复合函数级并行性的开发

3.2.1 函数相关图的动态属性

函数相关图类似动态的宏数据流图。所谓动态是指容许在函数相关图的一条弧上同时出现多个 token。直观地说，一个复合函数可以有多个实例同时运行。这是开发并行的手段。

实现动态函数有两种方法：一是当需要时复制一份新的复合函数；二是将复合函数的描述分为静态与动态两部分，当需要时仅创建动态部分，共享静态部分。比较起来第一种方法简单，但全部复制开销大。为降低开销我们采用第二种方法。

3.2.2 函数的动态派生

SDS 模型除了能并行执行不相关的复合函数外，也容许动态派生一个函数的多个实例，并行运行。这一方面是为了提高并行度，另一方面也是为了实现函数递归调用。函数的动态派生是通过 applyF，beginF—returnF 三个复合函数协同完成的。

3.2.3 forall 的动态展开

根据 forall 结构的语义，它是对一组数组的元素进行同样的加工，数组的元素之间不相关，因而可并行执行。

在 SDS 模型中，对 forall 有两种处理方式：其一是并行化，动态展开 forall 结构，生成与数组元素相同数量的实例，并行执行，这些实例中的复合函数的程序是同步非流水线 DFGC 图；其二是向量化，把一个数组分成适当大小的几段，每一段生成一forall 的实例，在这些实例中的复合函数的程序是同步流水线 DFGC 图。

采用两种方式的理由是为了使并行度的粒度大小适当，开销减小。例如以下 forall 结构：

```
forall i [1，128]
  construct A[i]+B[i]
endall
```

对 A，B 的每组数组元素来说，只是简单地做一次加法。如果对上述结构做并行化处理，即展开为 128 个复合函数，每个复合函数的操作仅仅是一次加法，这无异于指令级并行。为此，宜采用向量化方法。例如，将 A，B 划分为 8 段，每段 16 个元素，由一

个复合函数处理。编译时生成一个复合函数，动态生成8个复合函数。每个复合函数流水线式地加工16组元素。这样既开发了并行，又减小开销。

四、SDS-1系统的体系结构

4.1 SDS-1系统的层次结构

SDS-1系统是层次结构，这与SDS模型的层次结构相对应。它实质上是树型结构，如图4所示。根结点中有全局控制器和存储器，它们与多个局部控制器互相联接，再

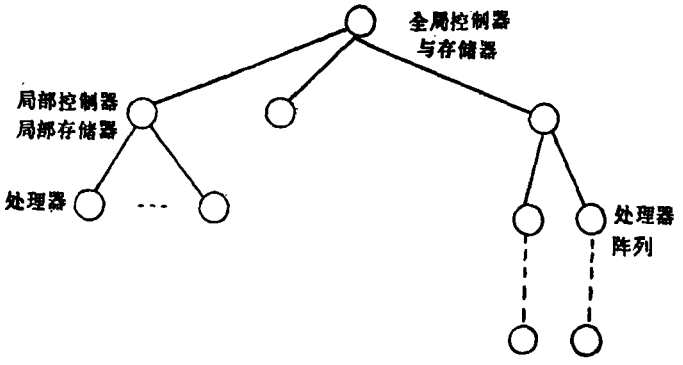

图 4

与多个处理器互联。复合函数与数据都存放在根结点，然后发送至局部存储器，再在局部控制器管理下发送至多个处理器并行执行。其结果送回局部存储器，再送至全局存储器，处理器之间不能直接通讯，需经局部存储器互连，各个处理部件（由局部控制器、局部存储器和多个处理器组成）之间亦不能直接通讯。

采用层次结构除了与程序的多级结构相适应之外，就硬件结构而言，对互连网络和存储体系设计有着明显优越性。

(1) 简化了互连网络，降低成本。

假定系统有N个处理部件，其中二分之一是向量部件，其它是通用部件。其中通用部件中又有N个处理器，每个向量部件有N×N个处理器，它们按流水线式排列，仅仅一头一尾的处理器与局部存储器互联。假定存储器的模块也有N个。则系统两个层次的互连网络都是N×N。设N=16，系统处理器数量共有$16\times8+(16\times16)\times8=2176$，系统规模已十分可观，但对互连网络的要求仅仅是17个相互无关的16×16的网络。即使采用Crossbar也是可行的。反之，若采用单一层次，则要求设计一个$2176\times M$，$M\gg16$的互连网络，从目前的技术条件看，这是不现实的。

SDS-1系统设置了8个处理部件，其中四个是向量部件，每个向量部件有4×4个处理器，每个通用部件含四个处理器，全系统共80个处理器。这样一个系统，只要求设计4×4互连网络。因此采用Crossbar是可行的。但若是单层结构，其复杂度为80×80，采用Crossbar已不可能，只能牺牲性能，降低网络的复杂度。

应当提及的是向量部件的流水线结构使得该部件中与互连网络直接联接的处理器数

量由 N×N 下降为N。所以，流水线也是简化结构的有效方法。

(2) 存储体系简单易行。

在SDS-1 系统中有三级存储器，图 4 中 反 映 了 两级，实际上还有一级，即通用寄存器。复合函数将数据取至处理部件，中间结果不送回全局存储器。这样一来可有效减少访问全局存储器的频度，一方面减少访问冲突，加速了处理器运行，另一方面，全局存储器空间占用少，也减小了存储器管理的开销。

一般多级系统中，多级存储器的最棘手的问题之一是所谓一致性问题。即当修改某一局部存储器时，可能需要修改全局存储器，甚至其它局部存储器，这一问题在本系统中已自然消除。因为SDS 模型不存在变量，没有赋值，任何值都不得修改，故一致性问题迎刃而解了。

SDS-1 系统的体系结构如图 5 所示，由五个主要部分 组 成。

(1) 全局控制器 GCU。

GCU 中有一个有效函数队列 EFQ(Enabled Function Queue)。满足点火条 件 且 控制标志为真，因而可执行的函数（称有效函数），它们的函数标题指针 被 送 入 EFQ 排队，GCU 按照 FIFO 的策略取出队列排头送往处理部件。

(2) 通用函数处理部件 GFU (General-purpose Funntion Processing Unit) 。

GFU 可以按照同步非流水线的 DFGC 模型加工一个复合函数。其输入/输出数据可以是标量，也可以是数组元素，故它是通用的函数处理部件。

(3) 向量处理部件 VPU (Vector Piplined Unit) 。

它按照同步流水线 DFGC模型加工数组。

(4) 程序存储器 PM。

用来存放复合函数的程序。

(5) 数组存储器 AM。

它用来实现多叉树的数据结构，完成各种数组操作。两种存储器均由多模块组成，它们与处理部件之间经网络相连接。

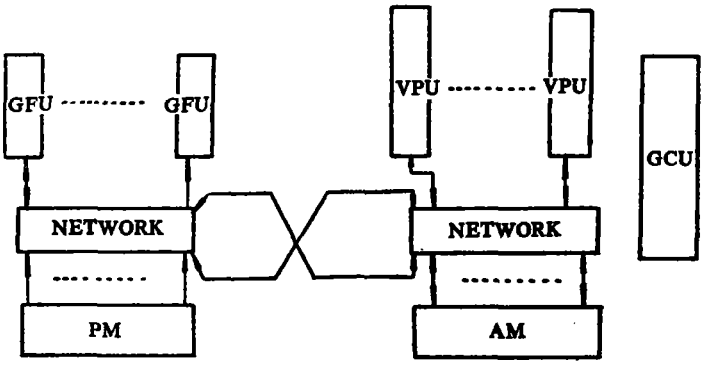

图 5 SDS-1 体系结构

五、模 型 机 结 构

5.1 研制模型机的目的

我们制作模型机有两个主要目的。其一是通过在模型机上运行各种典型程序，实测运行速度与处理器部件数量之间的关系，研究系统实际并行度和开销的大小，从而论证SDS-1 体系结构的优劣；其二是研制函数语言 VAL 的编译器，特别是研究划分复合函数的有效算法和优化算法。在此基础上进一步研究函数语言的特点，从而为设计新的更适当的函数语言打下基础。可见模型机不仅仅是研究硬件结构的手段，也是软件研究的环境和基础，这一点是其它方法不能比拟的。

SDS-1 的结构分为两个层次，开发两级并行。考虑到实际条件的限制，模型机仅模拟 SDS-1 的高层次，开发复合函数一级的并行。

5.2 模型机物理结构

模型机由一台主微型机与八台单板机构成。它们经一组单总线互相共享主存，互相通讯，形成紧密耦合系统。其总体结构如图 6 。

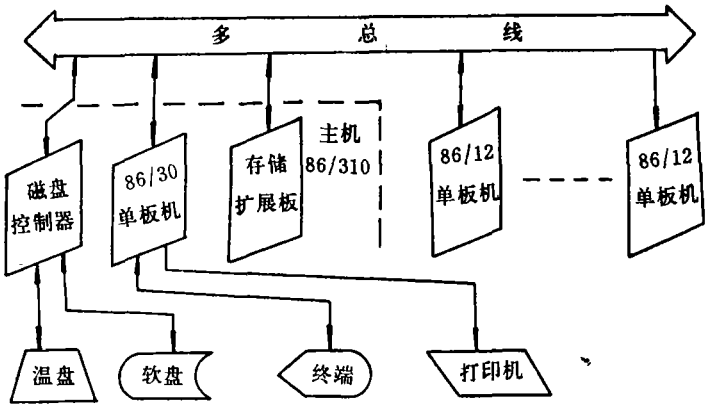

图 6 模型机总体结构

主微机（简称主机）管理外设，与外界通讯，并执行 SDS-1 的 GCU 部件和主存储器（包括 PM，AM）的功能，8 台单板机模拟 SDS-1 的 8 个处理部件，它们实际执行复合函数的各种运算。目前单板机仅完成 SDS-1 的通用函数部件的功能。向量部件采用流水线式的工作方式，对互连网络要求高。目前的单总线通频带太低，不能满足要求，故首先在模型机上设置通用函数处理部件。

主机选用的是 Intel 86/310 机，是 16 位微机，它由 8 块插件组成。一块是主机板 86/30，其中有 CPU，浮点部件，128K RAM，32K ROM，串、并行接口电路，主频 5M，对外数据总线宽 16 位，比一般 IBM PC宽一倍，因而数据传输快；另一块插件是 512K RAM 存储器扩充板；第三块是磁盘控制器板。这三块板都接在总线上，8 块单板机是 Intel 86/12，也是 16 位微机，它们全都接在总线上。各单板机本身有32K RAM 的双口存储器，都分配有总线上的地址，因而其它单板机都可通过总线访问。即 512K

存储器和各单板机存储器是共享的，由此构成紧密耦合的多机系统。各单板机通过共享存储器交换数据，实现相互通讯。这是整个系统实现并行的基础。

六、SDS-1 系统性能评价

本文用两种方法分析与评价 SDS-1 系统的性能。(1)利用应用程序分析 SDS-1 系统的性能；(2)利用模型机评价 SDS-1 系统。

6.1 应用程序分析

我们参照 Diety 和 Szewerenko[19]的方法进行分析。为简化计算，根据数据流机的特点，做了修正。

以下考查计算误差函数：

ERROR(1:128) = (square_root(x(1:128) - a) * *2 + (y(1:128) - b) * *2)/n

我们对不同的数据流机模型考查计算 ERROR 的执行过程，其结果统计在表 6.1 中。

表 6.1

		静态数据流机	动态数据流机	SDS-1
1	程序中指令数	6656	40	60
2	执行指令数	18176	22528	15765
3	取指令数	18176	640	525
4	存取数据数	54528	529	545
5	全局通信	72704	1169	1087
6	局部取指令	0	22528	15765
7	局部存取数据	0	67584	47295
			(局部相联访问)	(寄存器)
8	局部通信	0	90112	63060
9	系统并行度	128	64	64
10	最大并行度	6.5	6.8	23

(1) 指令的有效性

SDS-1 指令更为有效，省去了一些不必要的操作，在数据流机中需重复产生一些多次使用的数值，SDS-1 则不必。所以 SDS-1 执行的指令数分别是静态数据流与动态数据流的 0.86 与 0.7 (表 6.1，2)。

(2) 信息的流量

静态数据流机形成 72704 个操作包在全系统流动（未计算回答包），动态数据流机则有 1169 个全局信息包，而 SDS-1 仅有 1087 个操作包在系统内流动。SDS-1 全局通信是静态数据流机的 1.5%（表 6.1，5)。

SDS-1 的局部通信是动态数据流机的 70%（表 6.1，7)，而且SDS-1 主要是寄存器访问。

(3) 存储器存取指令与数据量

静态数据流机访问全局存储器 72704 次； SDS-1 数据流机访问全局存储 器 1087 次（表 6.1，5)。

动态数据流机访问局部存储器和匹配寄存器 90112 次；SDS-1 仅取指令访问局部存储器，共 15765 次。占动态数据流机的 17%（表 6.1，6，8)。

就以上三项而言 SDS-1 都占有显著的优势，它有效的解决了数据流系统中 常 见 的两个瓶颈问题：

(1) 存储器

SDS-1 对全局存储器的访问已大幅度下降；就局部访问而言，大量的是 寄 存 器 访问。

(2) 网络

SDS-1 信息传送也大幅度下降，因而，对网络要求降低，便于实现。

6.3 SDS-1 系统的模型机评价

6.3.1 测试程序

根据 SDS-1的应用领域，选择了三个典型程序，在模型机上运行，统计了有关的参数，测试了有关性能，这三个程序是矩阵乘法、平均误差和排序。

6.3.2 实测结果与评价

三个测试程序在模型机上实际运行，获得了一些数据（表 6.2，详见全文）。由 这些数据可给出程序运行时间与处理部件关系的曲 线，见 图 7 。

表 6.2 平均效果

部件数	1	1	3	4
系统运行时间	224190	133870	123290	101880
GCU 运行时间	490	480	490	460
GCU/SDS	0.2%	0.36%	0.4%	0.45%
tn/t1	1.0	0.597	0.55	0.45

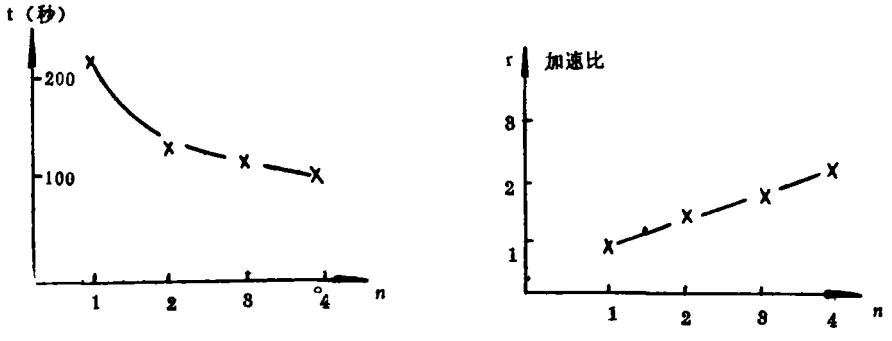

图 7

说明：1. GCU/SDS 为 GCU 运行时间与 SDS 系统运行时间之比；

2. tn/t1 为 n 个处理部件与一个处理部件时系统运行时间之比。

由这些数据和曲线可说明系统的性能是比较好的。程序2中只有四个复合函数，运行时系统动态生成了64个可并行执行的函数，有效地发掘了程序中的并行性。

SDS-1的GCU运行时间不到系统件运行时间的1.0%，这说明利用复合函数以及动态派生函数所引入的系统开销是微不足道的，却能有效减少指令级开销。这说明开发函数级并行是合理的。

七、结 束 语

1. 本文对数据流机进行了较深入的分析，明确了开销过大是数据流机的主要问题。而形成开销过大的原因是单纯依赖指令级并行和异步操作，没有控制。

2. 针对数据流模型的问题本文提出了有序的数据流与控制流相结合的模型DFGC作为高级语言与体系结构研究的基础。本文对DFGC的语义给出了较严格的说明，并就模型做了论证，在阐明了它的优越性的基础上提出了实用化的SDS模型。该模型开发两级并行，并将数据流与控制流相结合，动态与静态机制相结合，流水线与非流水线操作相结合，异步与同步操作相结合，使之融为一体形成一有机系统，充分地发挥各自的特长。

3. SDS模型保持了函数语义，因而支持高级函数语言。

4. 根据SDS模型设计了SDS-1系统的体系结构。

5. 构造了SDS-1系统的模型机和VAL语言的编译器。

6. 运用两种方法对SDS-1系统进行全面分析与评价，特别是利用模型机实测一些程序运行的结果，并由此得出结论：

(1) DFGC模型改进了数据流图；

(2) SDS模型可以在保持低开销的前提下有效地开发函数语言中的并行成分，由此而设计的SDS-1系统结构是可行的；

(3) 由于SDS-1系统结构简单，又是同构的模块结构，因而适合VLSI技术。

参 考 文 献

(略)

Computation Model and Its Architecture of Combining Dataflow and Control Flow

Liu Guizhong

Abstract

Dataflow computation can exploit high parallelism, but its overhead is great. In this paper, the improvement of dataflow graph is discussed, a model of combining dataflow and control flow is presented, and a practical architecture SDS-1 according to DFGC is described. The SDS-1 can develop two levels of parallelism, compound function level and instruction level. Finally, the performance of SDS-1 is analysed.