文章编号：1001-2486（2002）02-0036-08

一种应用于数据通道综合的定向搜索流水调度算法*

欧 钢
（国防科技大学 ATR 重点实验室，湖南 长沙 410073）

摘 要：流水调度是专用数字信号处理器高层综合中一个困难而急待解决的问题。给出了一种定向搜索流水调度算法，目标是使全面考虑了运算单元、寄存器和互连的硬件代价最小化。它作为一种利用启发信息的迭代算法，一方面克服了确定性算法爬峰能力差、易于陷于局部极值的缺点，另一方面启发信息的利用加快了搜索过程。典型设计实例显示算法性能达到或超过了目前流水调度文献报道的最好性能。

关键词：高层综合；流水时序调度；数字信号处理
中图分类号：TP391 文献标识码：A

A Directed Searching Pipeline Scheduling Algorithm for Datapath Synthesis

OU Gang
（ATR National Lab，National Univ. of Defense Technology，Changsha 410073，China）

Abstract：In the high－level synthesis of application－specific DSP，pipeline scheduling is a difficult and urgent problem. Concerning the global optimization of functional units，registers and interconnections，we present a novel algorithm called directed searching pipeline scheduling，which is an iterative algorithm utilizing heuristic information. The heuristic information speeds up the searching process，and the weakness of deterministic algorithms，which are vulnerable to trip in local optimal，poor at hill－climbing，is overcome in this algorithm. Typical design examples show that the performance of this transformation－based algorithm has reached or surpassed the best performance of pipeline scheduling algorithms reported so far.

Key words：high－level synthesis；scheduling；digital signal processing

1 定义

时序调度是数据通道综合中最重要的环节，它是数据流图到时间的映射。在满足顺序约束的前提下，时序调度把数据流图中的数据操作节点安排到恰当的控制步中，一方面使一个控制步可以同时执行多个数据操作，满足性能要求，另一方面合理控制各个控制步中的并行度，减少硬件开销。数字信号处理算法一般为无限循环，每一个循环对应着一个数据输入周期。流水调度把处理算法的一个周期分成几个长度相等的执行段，相邻周期的不同执行段并行执行，这种时间重叠技术是提高数据速率的有效方法。但是因为数据操作之间的相关关系不仅存在于周期内部和周期之间，而且还可能是递归的，因此流水调度比一般的时序调度要更加复杂和困难。一些时序调度算法虽然能够扩展为流水调度，但却不能处理递归的数据流图；有些设计系统为了挖掘周期之间数据操作的并行性[1,2]，首先运用重定时变换和流水变换来优化数据流图，然后再采用一般的非流水时序调度算法完成时序的映射。但是从变换搜索的角度来看，流水调度的优化可以归结为各数据操作节点在执行段内部和执行段之间的移动，已隐含完成了流水变换和重定时变换，它们融合成为了一个优化过程。因此在面向数字信号处理应用时流水调度应能得到更好的设计结果。

流水调度算法的输入包括三个部分：信号处理算法、性能要求和运算单元库，信号处理算法用数据流图表示，性能要求由参数数据重载周期（*DII*）和最大控制步（*TMAX*）设定。

* 收稿日期：2001-10-15
作者简介：欧钢（1969—），男，副教授。

定义 1（数据流图）：数据流图（DFG）是一个赋权的有向图，由四元组（J，$OPTYPE$，E，D）描述，其中：

·$J=\{INPUT,OUTPUT\}\cup I$，$INPUT$、$OUTPUT$ 分别是输入和输出节点，I 是数据操作节点集合；

·$OPTYPE$（i）表示数据操作的类型，$\forall i\in I$；

·$E\subseteq J\times J$，是代表节点之间数据相关关系的赋权有向弧的集合；

·D（e）$\in N$，是边的权重，等于延迟器的个数，$\forall e=(i\rightarrow j)\in E$，$N$ 是自然数集合（下同），它表示数据操作 i 所生成数据经过 D（e）个周期后被数据操作 j 利用，成为它的操作数。

定义 2（运算单元）：运算单元类型由一个五元组（K，$COST$，$DELAY$，$INITIME$，$OPSET$）描述，其中 K 是运算单元类型的集合，$\forall k\in K$：

·$COST$（k）是运算单元的代价因子，它一般正比于运算单元所占的芯片面积；

·$DELAY$（k）是运算单元的延迟时间，等于数据输入至运算结果输出之间的时间间隔；

·$INITIME$（k）是运算单元的初始化时间，等于运算单元相邻两次运算之间的最短时间间隔；

·$OPSET$（k）是运算单元能够执行的数据操作类型的集合。

如果运算单元是流水部件，那么 $INITIME(k)<DELAY(k)$ 如果是非流水部件，那么 $INITIME(k)=DELAY(k)$。$OPTYPE(i)\in OPSET(k)$ 表示数据操作 i 可以在类型 k 运算单元上执行。

我们假设在流水调度之前已经为各数据操作节点选定了运算单元类型，$FU(i)=k$ 表示数据操作 i 由类型为 k 运算单元完成。规定时序调度中输入节点总是安排在第一个控制步，输入节点和输出节点的延时为零。

定义 3（性能约束的流水调度）：给定数据流图 DFG，流水调度表述为一个映射 S：$I\rightarrow T$，其中 $T=\{t\mid 1\leqslant t\leqslant TMAX\wedge t\in N\}$ 是控制步集合，流水调度为第一次执行中的每一个数据操作确定开始执行的时刻，时间单位是控制步，并且满足：

·合法性，数据在被利用之前必须先产生；

·性能约束；

·最优化，使实现时序调度所需的硬件资源最少，硬件资源包括运算单元、寄存器、总线等。

数据流图的时序调度结果是定时数据流图（SDFG）或归一化定时数据流图（NSDFG）。NSDFG 是在更一般的意义上表示流水调度，不再局限于数据流图所给出的具体计算结构。通过对 NSDFG 的研究我们证明了：（1）从任意一个合法的初始流水调度出发，通过一系列节点或复合节点的合法上移变换或下移变换，可以搜索到设计空间中的任何一个流水调度；（2）如果 NSDFG 中没有临界计算环路，那么仅节点上移变换、下移变换就构成了完备的变换集[4]。

与逐步构造法相比，变换搜索法可以在代价函数中方便地表示数据通道的性能与运算单元、寄存器和互连等硬件代价之间的折衷关系，并且可以避免贪心而易于陷于局部极值的缺点。

2 初始流水调度

如果数据流图中存在环路，那么数据初始化周期 DII 有下限，下限为

$$DII_{\min}=\max_{Loop\in DFG}\left[\frac{LENGTH(Loop)}{ORDER(Loop)}\right]=\max_{Loop\in DFG}\left[\frac{\sum_{i\in Loop}EXE_i}{\sum_{(i\xrightarrow{d_{ij}}j)\in Loop}d_{ij}}\right] \tag{1}$$

如果 DII 小于下限，那么就不可能存在合法的流水调度；反之，如果 DII 不小于 $DII_{\min}$，那么一定存在合法的流水调度，使所有的顺序约束关系都得到满足，即：

$$S_{i2}\geqslant S_{i1}+EXE_{i1}-d_{12}\cdot DII,\ \forall e=(i_1\xrightarrow{d_{12}}i_2)\in E \tag{2}$$

下述算法步骤可以构造合法的流水调度：

（1）所有的节点都安排在第一控制步。

（2）按一定的顺序扫描所有的数据相关约束，如果都满足，则输出结果，算法结束。

如果发现顺序冲突，假设 $i_1 \xrightarrow{d_{12}} i_2$，有：

$$S_{i2} < S_{i1} + EXE_{i1} - d_{12} \cdot DII \tag{3}$$

则执行第3步。

(3) 化解冲突，把 i_2 节点移至使（2）式取等号的位置

$$S_{i2} = S_{i1} + EXE_{i1} - d_{12} \cdot DII \tag{4}$$

然后转步骤（2）。

构造算法一定会结束，证明见文献[3]。

3 代价函数

代价函数反映实现流水调度所需硬件资源的多少。它由三部分组成：运算单元代价分量，寄存器代价分量与互连代价分量。

$$COST(S) = \sum_{k \in K} COST(k) \times M_k + C_R \times Reg + C_B \times Bus \tag{5}$$

式中：$COST(k)$、C_R、C_B 分别为运算单元类型 k 的代价因子、寄存器的代价因子和数据总线的代价因子；M_k：类型 k 运算单元的数量；Reg：寄存器的数量；Bus：总线的数量。代价因子 $COST(k)$、C_R 与 C_B 是预先已知或设定的。时间并行必定要求空间展开，同时执行的数据操作不能共享运算单元，活跃期交叠的数据变量不能占据相同的物理寄存器，发生在相同时刻的数据传输必须由不同的总线来完成。

虽然只有在资源分配完成之后才能得到寄存器和互连代价的准确值，而资源分配又与数据通道的结构风格密切相关，但是时序调度中我们仍然可以得到它们较准确的估计。M_k、Reg、Bus 分别估计为在定时数据流图中由类型 k 运算单元完成的数据操作、活跃数据变量以及数据传输这三者在时间维上分布的最大并行度。

3.1 运算单元代价分量

定义符号 $PARALLEL_{OP}(k,t)$ 表示由类型 k 运算单元实现的、在控制步 t 开始执行或正在执行的数据操作的集合：

$$PARALLELop(k,t) = \{i \mid i \in I \wedge FU(i) = k \wedge t \in MS_i\} \tag{6}$$

式中 $1 \leqslant t \leqslant DII$，$MS_i$ 为数据操作节点 i 独占运算单元的控制步集合，那么

$$M_k = \max_{t=1}^{DII} | PARALLEL_{OP}(k,t) | \tag{7}$$

3.2 寄存器代价分量

3.2.1 变量的活跃期分析

运算单元完成数据操作 i 生成变量 VAR_i，直至 i 所有的立即后继数据操作开始执行，VAR_i 的活跃期方才结束。记 $LT_i = (c_i, d_i)$ 为 VAR_i 的活跃期，其中

$$c_i = A_j + EXE_i$$

$$d_i = \max_{j \in Suc(i)} (A_j + stage_{ij} \times DII) \tag{8}$$

式中 A_i 为数据操作节点 i 在定时数据流图中的位置，$stage_{ij}$ 是定时数据流图中计算路径 $i \rightarrow j$ 的级数。那么，变量 VAR_i 活跃期的长度为

$$| LT_i | = d_i - c$$

如果 $| LT_i | > DII$，令

$$\Gamma_i = \begin{cases} \left[\frac{| LT_i |}{DII}\right] + 1 ; & \text{如果 } (| LT_i | \mod DII) \neq 0 \\ \left[\frac{| LT_i |}{DII}\right] ; & \text{如果 } (| LT_i | \mod DII) = 0 \end{cases} \tag{9}$$

那么 VAR_i 的存储必须需要 Γ_i 个寄存器，因为在相邻 Γ_i 次迭代中数据操作 i 生成的变量的活跃期是重

叠的。当 $\Gamma_i > 1$ 时，必然要把 VAR_i 的活跃期分成 Γ_i 段，每段的长度小于或等于 DII，每段对应一个寄存器，并且认为是不同的变量 VAR_{id}，$1 \leqslant d \leqslant \Gamma_i$。分段的不同并不影响寄存器代价的计算，假设分段规则如下：第一段，变量 VAR_{i1} 的活跃期为 $LT_{i1} =$ （c_i，DII）；第二段、第三段、……、第（$\Gamma_i - 1$）段，变量 VAR_{i2}、VAR_{i3}、…、$VAR_{i(\Gamma_i - 1)}$ 的活跃期相同，为（0，DII）；第 Γ_i 段的活跃期为（0，d_i）。

3.2.2 寄存器代价分量的计算

定义符号 $PARALLEL_{VAR}$（t）表示在控制步 t 活跃的变量集合：

$$PARALLEL_{VAR}(t) = \{VAR_i \mid c_i \leqslant t \leqslant d_i\} \tag{10}$$

那么寄存器的数量可以估计为：

$$Reg = \max_{t=1}^{DII} \mid PARALLEL_{VAR}(t) \mid \tag{11}$$

3.3 互连代价分量

数据通道中的数据传输基本分为两种：存储运算结果和取操作数，如果采用线性拓扑的数据通道结构模型，那么只有取操作数才需要全局总线，所以在计算互连代价的时候我们仅考虑取操作数的数据传输。设 DT 表示数据传输，det（DT）表示由 DT 提供操作数的数据操作，$INITIME$ 是执行 det（DT）的运算单元的数据初始化周期，定义发生在控制步 t 的数据传输集合为 $PARALLEL_{DT}(t)$：

$$PARALLEL_{DT}(t) = \{DT \mid t \leqslant A_{det(DT)} \leqslant t + INITIME - 1\} \tag{12}$$

那么

$$Bus = \max_{t=1}^{DII} \mid PARALLEL_{Dr}(t) \mid = \max_{t=1}^{DII} \left(\sum_{k \in K} \mid PARALLEL_{OP}(k,t) \mid \times IN_k \right) \tag{13}$$

式中 IN_k 代表运算单元类型 k 的输入端口的个数。

4 定向搜索算法

在构造了初始流水调度之后，加上合法、完备的变换集，具有爬峰能力的定向搜索算法就可以快速求得流水调度的最优解或近似最优解。定向搜索算法包括上移迭代和下移迭代，上移迭代和下移迭代交替进行，每一次迭代包含一系列节点移动变换，不同于模拟退火等算法，这种变换不是随机的，而是确定性的。算法具有以下特点：

（1）在每一次迭代中节点的移动方向是固定的，在上移迭代中节点只能向上移动，在下移迭代中节点只能向下移动，一个节点移动变换，仅仅影响两个控制步中并行度的分布。如果 NSDFG 有临界计算环路，算法应先探测临界计算环路和环路上的节点，在上移迭代中，当环路的所有输入边存在松弛节点时，对环路实施复合节点上移，在下移迭代中，当环路的所有输出边存在松弛节点时，对环路实施复合节点下移，环路的复合节点移动会影响多个控制步中并行度的分布。

（2）移动节点的顺序是由优先级函数决定的，优先级高的节点先移动，优先级低的节点后移动。节点 i 的优先级函数反映了移动节点 i 对并行度的影响，如果节点移动能够增大并行度分布的均匀性，那么优先级函数值就大，反之则小。因为并行度在时间维的均匀分布能够提高硬件资源的利用率，利于降低硬件的开销。

优先级函数 $PRIORITY(i)$ 由数据操作分量 $P_{OP}(i)$、数据变量分量 $P_{VAR}(i)$ 和数据传输分量 $P_{DT}(i)$ 组成：

$$PRIORITY(i) = P_{OP}(i) + P_{VAR}(i) + P_{DT}(i) \tag{14}$$

$PARALLEL_{OP}(k,t)$、$PARALLEL'_{OP}(k,t)$ 分别是在节点移动之前和节点移动之后在控制步 t 中属于类型 k 的数据操作的并行集合，那么定义

$$P_{OP}(k) = C_k \times \sum_{t1=1}^{DII} \sum_{\substack{t2=1 \\ t2 \neq t1}}^{DII} (DIFF(t1,t2) \times \mathrm{MAX}(\mid PARALLEL(k,t1) \mid, \mid PARALLEL(k,t2) \mid)) \tag{15}$$

为类型 k 数据操作的优先级函数分量，式中 C_k 是运算单元类型 k 的代价因子，$DIFF$（$t1$，$t2$）反映类型 k 数据操作在控制步 $t1$、$t2$ 分布的均匀性的变化：

$$DIFF(t1,t2) = \| PARALLEL_{OP}(k,t1)| - |PARALLEL_{OP}(k,t2)\| - \| PARALLEL_{OP}(k,t)| - |PARALLEL_{OP}(k,t2)\| \tag{16}$$

如果均匀性增强，那么$DIFF(t1,t2)$为正，否则为负。整个节点移动的优先级函数中数据操作分量为

$$P_{OP}(i) = \sum_{k\in K} P_{OP}(k) \tag{17}$$

与数据操作分量定义相类似，优先级函数的数据变量分量定义为

$$P_{VAR}(i) = C_R \times \sum_{t1=1}^{DII} \sum_{\substack{t2=1 \\ t1\neq t2}}^{DII} (DIFF \times MAX(|PARALLEL_{VAR}(t1)|,|PARALLEL_{VAR}(t2)|)) \tag{18}$$

C_R 是寄存器的代价因子，$DIFF$ 反映变换前后活跃数据变量个数在控制步 $t1$、$t2$ 中的差异的变化：

$$DIFF = \| PARALLEL_{VAR}(t1)| - |PARALLEL_{VAR}(t2)\| - \| PARALLEL_{VAR}(t1)| - |PARALLEL_{VAR}(t2)\| \tag{19}$$

式中 $PARALLEL'_{VAR}(t)$代表变换后控制步 t 中活跃数据变量的集合。

优先级函数的数据传输分量定义为

$$P_{DT}(i) = C_{DT} \times \sum_{t1=1}^{DII} \sum_{\substack{t2=1 \\ t1\neq t2}}^{DII} (DIFF \times \mathrm{MAX}(|PARALLEL_{DT}(t1)|,|PARALLEL_{DT}(t2)|)) \tag{20}$$

式中 $DIFF$ 反映变换前后活跃数据传输个数在控制步 $t1$、$t2$ 中的差异的变化：

$$[DIFF = \| PARALLEL_{DT}(t1)| - |PARALLEL_{DT}(t2)\| - \| PARALLEL_{DT}(t1)| - |PARALLEL_{DT}(t2)\|] \tag{21}$$

式中 $PARALLEL'_{DT}(t)$代表变换后控制步 t 中数据传输的并行集合。

(3) 不管变换前后解的质量如何变化，变换总是被接受。

为简化计，算法不区分被移动节点是否处于环路、或者是否所有输入边存在松弛节点，而是在上移变换中节点 i 上移一个控制步后，算法逐一地解决因节点 i 上移所引起的顺序冲突，如果所有的顺序冲突以及在解决冲突中新产生的顺序冲突都被解决，而同时输入节点 $INPUT$ 并没有因此受到影响，仍处于控制步 1，那么节点 i 的移动是成功的，否则判定为失败，因为节点 i 这时已处于控制步 $ASAP_i$；同理，在下移变换中，节点 i 下移一个控制步后，算法将逐一解决由此所引起的顺序冲突，如果所有的顺序冲突以及在解决冲突中新产生的顺序冲突都被解决，而同时输出节点 $OUTPUT$ 并没有因此突破参数 $TMAX$ 的限制，那么节点 i 的移动是成功的，否则是失败的，因为节点 i 已处于控制步 $ALAP_i$。所以在下面的论述中，移动节点 i 的含义是指由节点 i 移动所触发的一系列节点移动。

5 实验结果

本文给出 3 个流水调度设计实例：16 点 FIR 滤波器、五阶椭圆滤波器和 LMS 自适应滤波器，并与几种调度算法的结果进行比较，这些调度算法包括：Fold[3]、Sehwa[5]、PLS[6]、MARS[7]、FDLS[8]、模拟进化算法[9]。这其中 Sehwa、FDLS 算法不能处理包含递归结构的数据流图，当数据流图中含有计算环路时，算法退化为一般的非流水的调度，PLS、Fold、MARS 算法是资源约束的流水调度算法，但资源约束仅考虑了运算单元。

在这些设计实例中都采用了相同的一个假想运算单元库：加法器的代价因子为 1，一个加法操作可以在一个控制步中完成；一般乘法器的的代价因子为 4，一个乘法操作需要两个控制步来完成；流水乘法器的代价因子为 5，它的数据初始化时间为一个控制步，延时为两个控制步。此外数据总线和寄存器的代价因子都等于 1。

5.1 16 点 FIR 滤波器

这是一个不含环路的标准测试设计，内有 15 个加法操作和 8 个乘法操作。表 1、表 2 分别是采用非流水乘法器和流水乘法器定向搜索算法的实验结果，对于不同的 DII 值，定向搜索算法的结果都达

到了运算单元的理论下界。

表 1 非流水乘法器定向搜索流水调度算法

Tab.1 The directed-search pipeline scheduling algorithm with nonpipelined multipliers

DII	乘法器下限	加法器下限	乘法器	加法器	寄存器	数据总线	代价函数值
1	16	15	16	15	56	62	197
2	8	8	8	8	35	32	107
3	6	5	6	5	29	22	80
4	4	4	4	4	27	16	63
5	4	3	4	3	24	14	57
6	3	3	3	3	22	12	49
7	3	3	3	3	20	10	45
8	2	2	2	2	21	8	39
9	2	2	2	2	20	8	38
10	2	2	2	2	20	8	38
11	2	2	2	2	20	6	36
12	2	2	2	2	19	6	35
13	2	2	2	2	18	6	34
14	2	2	2	2	18	6	34
15	2	1	2	1	19	6	34
16	1	1	1	1	21	4	30

表 2 流水乘法器定向搜索流水调度算法

Tab.2 The directed-search pipeline scheduling algorithm with pipelined multipliers

DII	乘法器下限	加法器下限	乘法器	加法器	寄存器	数据总线	代价函数值
1	8	15	8	15	48	46	149
2	4	8	4	8	32	24	84
3	3	5	3	5	26	16	62
4	2	4	2	4	30	12	56
5	2	3	2	3	22	10	45
6	2	3	2	3	20	8	41
7	2	3	2	3	19	8	40
8	1	2	1	2	22	8	35
9	1	2	1	2	21	6	34
10	1	2	1	2	19	6	32
11	1	2	1	2	19	6	32
12	1	2	1	2	19	4	30
13	1	2	1	2	18	4	29
14	1	2	1	2	18	4	29
15	1	1	1	1	18	4	28
16	1	1	1	1	18	4	28

关于 16 点 FIR 滤波器，许多文献采用了与我们不同的假设：加法器完成一个加法操作的时间为半个控制步，乘法器完成一个乘法操作的时间为一个控制步。表 3 是在 *DII* = 3 时，Sehwa、FDLS、模拟进化算法和 Theda.Fold 算法的结果。相对应地，比较表 1 中 *DII* = 6 时的调度（带下划线的部分），这时硬件代价仅为 3 个乘法器、3 个加法器、22 个寄存器和 12 条数据总线，明显优于表 3 中 4 种算法的结果。MARS 算法 16 点 FIR 滤波器的流水调度也达到了运算单元资源的下界，但它的资源约束中不含寄存器和数据总线。

表 3 16 点 FIR 滤波器不同算法的调度结果

Tab.3 The scheduling results of different algorithms for the 16 – tap FIR filter

算法	*DII*	乘法器	加法器	寄存器	数据总线
Sehwa[5]	3	6	3	24	20
FDLS[8]	3	6	3	– –	– –
模拟进化算法[9]	3	5	3	23	20
Fold[3]	3	5	3	– –	– –

5.2 5 阶椭圆滤波器

信号流图中包含许多计算环路，有 26 个加法操作和 8 个乘法操作。表 4、表 5 是定向搜索算法和模拟进化算法的调度结果。由于计算环路的存在，模拟进化算法这时退化为一般的非流水时序调度。由表中可以看到，在不同的条件下，定向搜索算法的结果都要明显优于模拟进化算法，这主要是因为流水调度中所隐含的重定时变换对计算结构所作的优化。

表4 五阶椭圆滤波器调度结果的比较（非流水乘法器）

Tab.4 A comparison of scheduling results for the 5th – order elliptic filter (nonpipelined multipliers)

	定向搜索算法				模拟进化算法[9]			
DII	乘法器	加法器	寄存器	数据总线	乘法器	加法器	寄存器	数据总线
17	2	2	10	8	3	3	11	11
18	2	2	10	6	2	2	10	9
19	2	2	10	6	2	2	10	8
20	1	2	10	6	2	2	10	8
21	1	2	10	6	1	2	10	8

表5 五阶椭圆滤波器调度结果的比较

Tab.5 A comparison of scheduling results for the 5th – order elliptic filter (pipelined multipliers)

	定向搜索算法				模拟进化算法[9]			
DII	流水乘法器	加法器	寄存器	数据总线	流水乘法器	加法器	寄存器	数据总线
16	1	3	10	6	– –	– –	– –	– –
17	1	2	11	6	2	3	10	10
18	1	2	10	6	1	3	10	8
19	1	2	10	4	1	2	10	7
20	1	2	10	4	– –	– –	– –	– –
21	1	2	10	4	– –	– –	– –	– –

5.3 LMS 自适应滤波器

这个测试设计来自 MARS[7]，是含有五个系数的 LMS 自适应滤波器，包含 11 个乘法操作和 10 个加法操作。表 6 和表 7 是分别采用非流水乘法器和流水乘法器时定向搜索算法和 MARS 算法得到的时序调度结果，由表中带下划线的部分可以看出，即使不考虑定向搜索算法对硬件资源代价全面下降所做的努力，它得出的流水调度所用的运算单元也少于或等于 MARS 得出的流水调度所用的运算单元数。

表6 采用非流水乘法器时 LMS 自适应滤波器调度结果的比较

Tab.6 A comparison of scheduling results for the LMS adaptive filter between the directed – search pipeline scheduling algorithm and MARS with nonpipelined multipliers

	定向搜索算法				MARS[7]	
DII	乘法器	加法器	寄存器	数据总线	乘法器	加法器
8	5	4	15	12	– –	– –
9	4	2	13	12	5	4
10	4	2	13	8	4	2
11	3	2	13	8	– –	– –
12	3	2	13	6	– –	– –
13	3	1	13	6	– –	– –
14	3	1	13	6	– –	– –
15	2	1	13	6	2	1
16	2	1	13	4	– –	– –
17	2	1	13	4	– –	– –
18	2	1	13	4	– –	– –

表7 采用流水乘法器时 LMS 自适应滤波器调度结果的比较

Tab.7 A comparison of scheduling results for the LMS adaptive filter between the directed – serach pipeline scheduling algorithm and MARS with pipelined multipliers

	定向搜索算法				MARS[7]	
DII	流水乘法器	加法器	寄存器	数据总线	流水乘法器	加法器
8	4	4	14	10	– –	– –
9	2	2	13	8	4	4
10	2	2	13	6	4	2
11	2	2	13	4	3	2
12	2	2	13	4	2	2
13	2	1	13	4	– –	– –
14	1	2	13	4	3	1
15	1	1	12	4	2	1
16	1	1	12	4	1	1
17	1	1	12	4	– –	– –
18	1	1	12	4	– –	– –

6 结束语

定向搜索算法不是选择使代价函数最小的节点移动，而是选择优先级最高的节点进行移动，虽然单个的这种节点移动不一定导致代价函数的下降，但是依靠这种节点移动的序列却可能最终找到最优或接近最优的解。定向搜索算法是一种确定性的算法，与列表调度、力矢量调度等启发式逐步构造的算法不同，它是一种利用启发信息的迭代搜索算法，一方面启发信息的利用大大加快了搜索过程，另

一方面又克服了一般确定性算法易陷入局部极值、不具爬峰能力或计算时间长的弱点。定向搜索算法表现出优异的性能，而且具有以下特点：

（1）收敛速度快。针对不同的数据流图或改变一些参数，我们做了 600 多组实验。结果是：435 组在第一次的迭代中，算法就找到了最小代价函数，有 163 组在第二次迭代中，算法找到了最小代价函数，分别有 27 组、19 组、9 组和 3 组在第三次迭代、第四次迭代、第五次和第七次迭代中才找到最小代价函数。因此，在绝大多数情况下定向搜索算法只需要一两次迭代搜索就能够完成流水调度的优化，算法收敛的速度是相当快的。

（2）能同时找到硬件代价相同的多个调度。如果有多个最优调度可以供后续设计流程选择，这有助于提高最终数据通道的设计质量。定向搜索算法在节点移动变换的搜索过程中，如果发现得到的流水调度的代价函数值与当前搜索到的最小代价函数值相同或相近，则把它保存起来。

（3）同样适用于一般的非流水时序调度。当 DII 与 $TMAX$ 相等的时候，移去数据流图中 $D(e)>0$ 的有向弧，那么定向搜索算法同样可以解决非流水时序调度的问题。

参考文献：

[1] Haroun B S，Elmasry M I. Architecture synthesis for DSP silicon compliers［J］.IEEE Trans.on CAD，April 1989，431－447.

[2] Potkonjak M，Rabaey J. Optimizing resource utilization with transformations［J］. IEEE Trans.on CAD，March 1994，277－292.

[3] Lee T，Wu A C. A transformation－based method for loop folding［J］. IEEE Trans.on CAD，April 1994，439－450.

[4] 欧钢．应用于流水时序调度的归一化定时数据流图理论［J］．国防科技大学学报，2001，23（1）.

[5] Park N，Parker A C. Sehwa：A software package for synthesis of pipelines from behavioral specifications［J］.IEEE Trans.on CAD，Mar.1988，356－370.

[6] Hwang C. PLS：A scheduler for pipeline synthesis［J］. IEEE Trans.on CAD，Sept.1993，1279－1286.

[7] Wang C. High－level DSP synthesis with concurrent transformation，scheduling and allocation［J］. IEEE Trans.on CAD，March 1995，274－295.

[8] Paulin P G. Force－directed scheduling in digital signal processing［J］. IEEE Trans.on CAD，June 1989，661－679.

[9] Ly T A，Applying simulated evolutoin to high level synthesis［J］. IEEE Trans.on CAD，March 1993，389－409.