doi:10.3969/j.issn.1000-436x.2013.02.006

融合面积估算和多目标优化的硬件任务划分算法

陈乃金[1,2]，江建慧[1]

(1. 同济大学 软件学院，上海 201804；2. 安徽工程大学 计算机与信息学院，安徽 芜湖 241000)

摘 要：针对可重构计算机系统配置次数（划分块数）的最小化问题，提出了一种融合面积估算和多目标优化的硬件任务划分算法。该算法每次划分均进行硬件资源面积的估算，并且通过充分考虑可重构资源的使用、一个数据流图所有划分块执行延迟总和、划分模块间边数等因素构造了新的探测函数 prior_assigned()，该函数能够计算每个就绪节点的优先权值，新算法通过该值能动态调整就绪列表任务节点的调度次序。实验结果表明，与现有的层划分、簇划分、增强静态列表、多目标时域划分、簇层次敏感等 5 种划分算法相比，该算法能获得最少的模块数，并且随着可重构处理单元面积的增大，除层划分算法之外，其执行延迟的均值也是最小的。

关键词：可重构计算；时域划分；最小化模块数；资源约束；探测函数；多目标优化

中图分类号：TP316 **文献标识码**：A **文章编号**：1000–436X(2013)02-0040-16

Hardware-task partitioning algorithm merged area estimation with multi-objective optimization

CHEN Nai-jin[1,2], JIANG Jian-hui[1]

(1.School of Software Engineering, Tongji University, Shanghai 201804, China;

2.College of Computer and Information Engineering, Anhui Polytechnic University, Wuhu 241000, China)

Abstract: In order to minimize the number of configuration (i.e. the number of partitioned modules) issue in reconfigurable computing systems, a hardware-task partitioning algorithm merged area estimation with multi-objective optimization was presented. It could estimate the area of hardware resource in each partitioning. Also, the detection function prior_assigned() was constructed with the guideline of making good use of reconfigurable resources, the execution delay sum of all partitioning modules of a data flow graph, and number of edges between modules, etc. The detection function could calculate priority values of nodes. The proposed algorithm could adjust dynamically the scheduling order of the ready list task-nodes by the priority values. Experimental results show that the proposed algorithm can get less modules than that of level-based partitioning, cluster based partitioning, enhanced static list, multi-objective temporal partitioning and level sensitive cluster based partitioning algorithms, and it can also obtain the least average execution delay with increase of the area of reconfigurable processing unit except level-based partitioning.

Key words: reconfigurable computing; temporal partitioning; minimum number of modules; resource restraint; detection function; multi-objective optimization

1 引言

近年来，随着 VLSI 技术的迅速发展，尤其是大规模高性能可编程器件的出现，促进了可重用芯片实现的多样化，从而在很大程度上影响了计算机软硬件计算平台的重新定位。可重构计算 (RC, reconfigurable computing)是一种融软件的编程灵活性和硬件的高效性于一体的计算方式。根据应用的不同需要对可重构硬件进行动态配置，根据配置信息来改变其内部可重构单元的功能和相互之间的

收稿日期：2011-11-08；修回日期：2012-05-11

基金项目：国家自然科学基金资助项目（60903033）；国家高技术研究发展计划（“863”计划）基金资助项目（2009AA011705）

Foundation Items: The National Natural Foundation of China (60903033); The National High Technology Research and Development Program of China (863 Program) (2009AA011705)

连接关系，并在辅助设备(包括外围控制硬件和软件)的协同下完成相应的计算任务，这种系统已经在很多领域得以广泛应用[1~4]。音、视频的编解码等计算密集型任务的关键循环占用了大量计算时间，如果将计算密集型任务经过软硬件划分，然后提取出其关键循环的数据流图(DFG, data flow graph)，并将其划分映射到可重构单元阵列(RCA, reconfigurable cell array)上执行，这将大大加快计算密集型任务的执行效率，但是 RCA 的资源有限，所以在动态可重构系统中，当要计算的硬件任务所需的资源大于硬件能够提供的资源时，就必须要对其进行划分，划分是映射的前提，并且可以直接形成粗粒度可重构处理单元(RPU, reconfigurable processing unit)流水化映射[5~8]的装载调度队列，其结果直接影响从计算密集型任务提取的核心循环在 RPU 上执行速度。

2　相关研究

传统的 DFG 时域划分算法根据电路的抽象层次可大致分为网表级时域划分和行为级时域划分 2 类算法。网表级时域划分算法针对的是电路。基于网络流的网表级时域划分算法将电路定义为一个网络流，它可表示为由门和寄存器所组成的节点集、由门和寄存器之间的连接所组成的边集，以及由门和寄存器的面积所组成的面积等 3 个集合所构成的三元组。而一个计算密集型任务或程序的关键循环可以用 DFG 表示，图中每个节点表示一个操作符（或运算符），对这样的图所进行的时域划分通常称为行为级时域划分。

本文研究行为级时域划分方法。文献[9]首次直接讨论了面向可重构计算机系统的硬件任务划分问题，并提出了 2 个经典的单目标时域划分算法，即层划分(LBP, level based partitioning)和簇划分(CBP, cluster based partitioning)，LBP 算法在某一硬件面积约束基础上，对计算任务节点进行按层划分。优点是试图获得较大的并行度，其目的是试图获得一个任务 DFG 的所有划分块执行延迟总和的较小化，缺点一是划分块间的通信成本较大(表现为一个任务 DFG 划分块间的非原始 I/O 边数较多)；二是不能根据实际划分情况动态调整就绪列表队列，结果产生了大量硬件碎片；三是把第 0 层的输入作为第一个划分块，这样做有可能会使任务 DFG 执行延迟的增大。CBP 算法试图尽可能地将联系紧密的操作放到一个划分块中，其目的是想获得划分后 DFG 块间的较小非原始 I/O 边数。缺点一是任务 DFG 的每个划分块内部计算任务节点的并行度较小；二是不能根据实际划分情况动态调整就绪列表队列；三是把第 0 层的输入作为第一个划分块不太合理，理由同 LBP 算法。

簇层次敏感的划分(LSCBP, level sensitive cluster based partitioning)算法对 CBP 算法进行了改进[10]，提高了硬件资源的利用率，效果良好，但是该算法缺点一是划分块间的边数仍然较大；二是当一个划分块间运算节点的输出边数超过 1 时，直接按实际边数进行统计，这样做可能导致该运算结果被多次存储；三是把第 0 层的输入化为第一个划分块不太合理，理由同 LBP 算法。

基于流网络的多路任务划分(NFMP, network flow-based multiway-task partitioning)算法是追求划分块间边数较小化的单目标时域划分算法，NFMP 算法首先运用了最大流最小割理论计算获得待划分任务 DFG 可行的最小割集初始划分[11]，然后在此基础上，通过划分子图构造、广度优先搜索和最短路径等相融合来获得划分块间通信成本的较小化(表现为一个任务 DFG 划分块间的非原始 I/O 边数较少)，效果较好。但是该算法缺点一是在某一硬件资源面积约束下，对待划分任务 DFG 所需的划分块总数考虑不足；二是没有考虑每个划分块内运算任务节点的并行度。

现有的典型多目标时域划分算法主要有多目标时域划分(MOTP, multi objective temporal partitioning)[12]、增强的静态列表(ESL, enhanced static list)[13]等。

MOTP 算法试图尽可能地考虑划分块数、划分块内运算节点的并行度和划分块间的通信量等 3 个指标来进行任务 DFG 的时域划分。优点是获得了较小的划分块数。缺点一是在每次划分时，没有进行硬件资源面积的估算，可能产生硬件碎片；二是统计划分块间输出边的数量不太合理，理由同 LSCBP 算法。

ESL 算法考虑了任务 DFG 块之间的通信代价和模块关键路径等指标。其不足是对并行因素考虑不够，且没有考虑任务 DFG 划分中可能产生的硬件碎片问题。

本文在综合考虑了一个计算任务 DFG 划分块数最小化、所有划分块执行延迟总和、所有划分块之间的 I/O 边数的较小化等 3 个方面的因素，提出

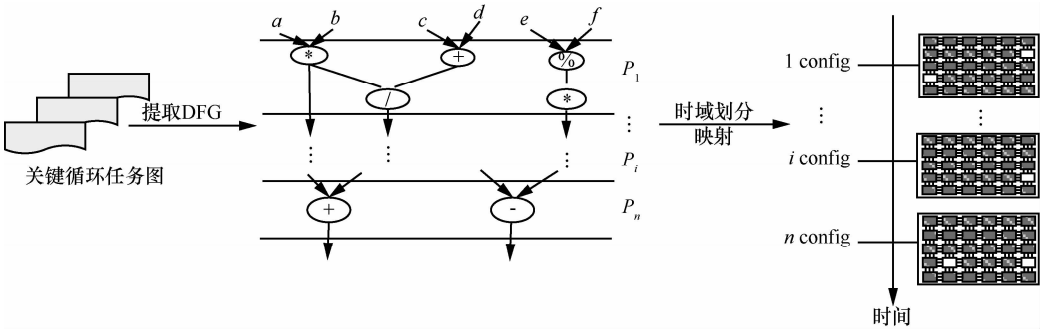


图 1　可重构系统划分映射模型

了一种融合面积估算和多目标优化(AEMO, area estimation with multi-objective optimization)的硬件任务划分算法。

3　问题定义

本文的研究有 3 个前提条件[14]：1) 待划分的任务由 DFG 表示，一个计算密集型任务已经转换为一个 DFG；2) 划分块数等于 RPU 中的 RCA 重复使用的次数，任一划分均可按划分形状直接映射到一块 RCA 上去，并且一个划分对应一块 RCA，如图 1 所示；3) 划分算法只考虑硬件实现的计算延迟，而不考虑互连延迟。

定义 1[14]　一个计算任务或程序的 DFG 是一个有向无环图，可以表示为 $G=(V, E, W, D)$，顶点集 $V=\{v_i|v_i$ 是有序运算符，$1\leqslant i\leqslant n\}$，$|V|=n$ 表示运算符的个数；边集 $E=\{e_{ij}|e_{ij}=<v_i, v_j>, 1\leqslant i, j\leqslant n\}$，$e_{ij}$ 表示从 v_i 到 v_j 的有向边，v_i 是 v_j 的直接前驱节点，v_j 是 v_i 的直接后继节点，其表示了 v_i 和 v_j 2 个运算符的先后依赖关系，v_j 的执行依赖于 v_i，$|E|=m$ 为边的数量；权集 $W=\{w_i|$表示 v_i 所占的硬件资源面积，$1\leqslant i\leqslant n\}$；$D$ 代表延迟集，$d_i\in D$ 代表第 i 个运算节点的执行延迟。

一般而言，RPU 中的 RCA 的面积为一个定值（表示为 A_{RPU}），任意一个任务 DFG 很难被全部映射上去，这就需要对其进行划分。

定义 2　一个 DFG 的划分问题可以描述如下。

输入：$G=(V, E, W, D)$；A_{RPU}。

输出：G 的一个划分 $P=\{P_1, P_2, \cdots, P_M\}$。

约束条件：1) $\bigcap_{i=1}^{M} P_i=\Phi$；2) $\bigcup_{i=1}^{M} P_i=V$；3) $\forall P_i\in P$，$1\leqslant i\leqslant M$，$\sum_{i=1}^{n} w_j\leqslant A_{\mathrm{RPU}}$，$w_j$ 为划分 P_i 中节点 j 的权值；4) P 中所有划分块之间不存在非法依赖关系；5)RPU 支持流水线处理。

目标：1) 划分块数的最小化；2) 所有划分块执行延迟总和的较小化；3) 尽可能减少划分块间 I/O 边数。

定义 3　若一个任务 DFG 存在一个划分，划分块之间严格按时间顺序执行，它的所有划分块之间如有一个产生了非法依赖关系，则该划分就称为不合理的划分。

设 v 是一个计算任务或程序的 DFG 的有序运算符中的任一个运算节点，则 v 被划入一个划分块的前提是 v 的所有的前驱节点均已经被分配到相应的划分块中。一个合理的划分要求划分块之间均不产生非法依赖关系。因 DFG 是一个有向无环图，如果一个节点的前驱节点被分配到其后继模块中，就会引发非法依赖关系现象。图 2 给出了这种情况的一个例子，为了简化问题描述，假设一个任务或程序的 DFG 中各个节点的权值相同，这样，本文可以将面积的单位等价于节点的个数。若 $A_{\mathrm{RPU}}=2$，获得 P_1、P_2、P_3 3 个划分，设 P_3 总是最后执行，但如果划分块 P_1 和 P_2 的执行次序是从 $P_1\rightarrow P_2$，因 P_1 划分中运算节点 v_4 前驱 v_2 位于 P_2 划分，所以划分块 P_1 和 P_2 之间产生了非法依赖关系。

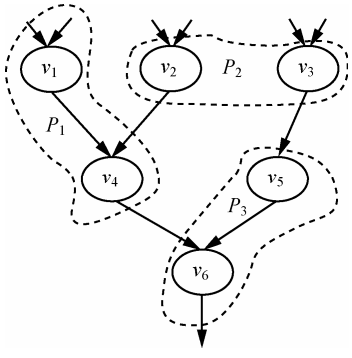


图 2　一个不合理划分的例子

4　AEMO 算法

4.1　AEMO 算法设计

设待划分 DFG 的运算符节点列表 queuelist={v_1, v_2, v_3,…, v_n}，从列表队列节点里挑选出优先级高的运算节点构成新的就绪列表 readylist={v_1', v_2', v_3',…,v_n'}，每次划分时依次从 readylist 中挑选节点。AEMO 算法采用以下 5 个策略来进行任务划分。

策略 1　可重构划分面积逻辑容量估算和按权值调度划分同时进行，获得最小的划分块数。

下面给出 2 种实现方案。

1) **方案 1**　在一个 A_{RPU} 的约束下，采用的策略是从就绪队列(由入度为 0 的点组成)中选取运算节点权值最小(本文约定节点的权值越小，则该点优先级就越高)的点作为起点，按深度优先搜索(DFS, depth first search)进行预划分，获得了一个合理划分，并计算硬件碎片面积 area1=A_{RPU}-area_filled1 (其中 area_filled1 表示所有划入当前块节点的面积累加和)；然后再以同样的起点按广度优先搜索(BFS, breadth first search)进行预划分，获得了一个合理划分，并计算硬件碎片面积 area2=A_{RPU}-area_filled2；将 area1 和 area2 进行大小比较，如果 area1 小，则本次划分按 DFS 方式进行，否则按 BFS 方式进行，但是这样做可能有一个缺陷，若 area1= area2，且 DFS 和 BFS 一遇到不满足要求的节点就停止，则这样会导致划分方式的不明确。

例 1　假设算术表达式中允许包含 2 种括号：圆括号和方括号，其嵌套的顺序合理，则一个算术表达式可表示为：y=[[[(a×b×k−(g+h))+(c×d×l−m)]%[((a×b)+(e×f+(i+j)))+n]+o]+[[[(a×b+k−(g+h))+(c×d×l−m)]+[((a×b)+(e×f+(i+j)))+n]+p]×[[[(a×b×k−(g+h))+(c×d×l−m)]%[((a×b)+(e×f+(i+j)))+n]]+[[(a×b+k−(g+h))+(c×d×l−m)]+[((a×b)+(e×f+(i+j)))+n]]]]]+[[[(a×b+k−(g+h))+(c×d×l−m)]+[((a×b)+(e×f+(i+j)))+n]+p]+[[[(a×b×k−(g+h))+(c×d×l−m)]%[((a×b)+(e×f+(i+j)))+n]]+[[(a×b+k−(g+h))+(c×d×l−m)]+[((a×b)+(e×f+(i+j)))+n]]]−q]，该算术表达式的 DFG 如图 3(a)所示，其层次为 9，原始输入边有 17 条，原始输出边有 1 条，非原始输入输出边数为 29 条，有 23 个运算任务节点，其集合 V={v_i | 1≤i≤23}，划分前 P=Φ，划分后 P={P_1,P_2,…,P_M}，设 A_{RPU}=65，则按 DFS (P_1={v_1，v_6}，如图 3(b)所示)或者 BFS (P_1={v_1,v_2}，如图 3(c)所示)进行预划分获得的硬件碎片均为 area1= area2=65−54=11，虽然 v_3、v_4 均有划入的可能，但是却不能划入。为此引入第 2 种方案。

2) **方案 2**　在一个 A_{RPU} 的约束下，从就绪队列中按权值选取优先级最高的点作为起点，按 DFS 进行划分，每划分一次，就更新该点后继节点的入度，如果入度为 0 就直接预划入，如果入度不为 0，考察该点的其他没有划入当前块的前驱能否划入，如果能，则一并预划入，直到没有节点可以填入 area_filled1 为止，这样就获得了一个 DFS 的合理划分，计算 area1=A_{RPU}-area_filled1，若 area1<value (value 为一设定的阈值，其范围为(0，10]，本文设为 10)，则本次划分按 DFS 进行，然后计算没有划入就绪点的探测函数值，再按 A_{RPU} 和权值大小贪婪划入可以划入的点(详细的实现细节见后面的例子说明)，否则按 BFS 进行权值层贪婪划分，即使某个节点的后继入度更新为 0，也不直接划入当前块。这样做的目的是尽可能保证每次划分均能最大化利用 A_{RPU}，同时增大一个划分块内的运算节点的并行度。

由上可知，策略 1 侧重于要求每次划分尽可能填满每一块 RCA，从而使任务 DFG 总的划分块数达到最小，而且按 BFS 进行权值贪婪划分时，考虑了划分块内运算节点的并行度。

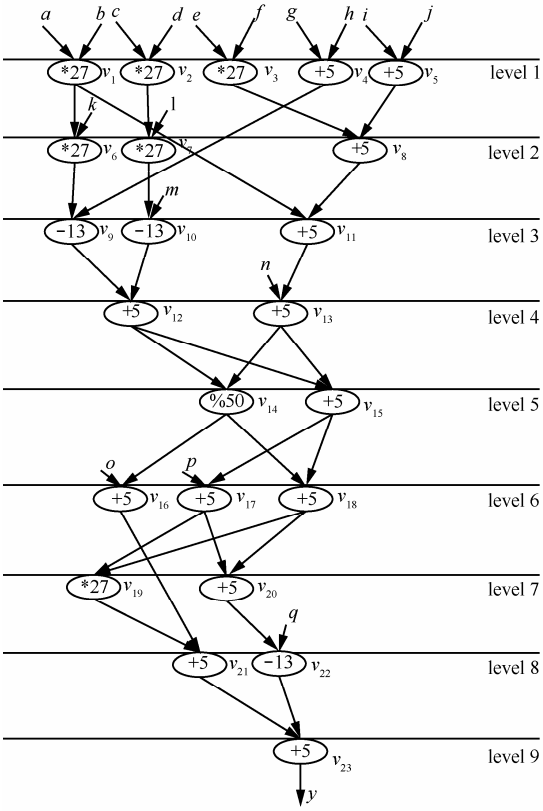
策略 2　执行延迟大优先(EDTLF, execution delay time long first)和可重构运算阵列面积的充分利用相融合。

当有多个节点处于就绪状态时，运用“尽可能早”原则，在保证 DFG 获取合理划分的前提下，优先选择执行延迟大和占用硬件面积大的运算节点。在延迟相同的前提下，占用硬件面积越大的任务越优先，而在占用硬件面积一样的前提下，延迟越大的任务越优先。在遇到当前点不能划入当前块时，还要动态考察该点之后还有无可以划入当前块的点，如有将其贪婪划入，这样做的目的是优先响应执行延迟大节点的同时，又充分利用了可重构运算阵列面积。

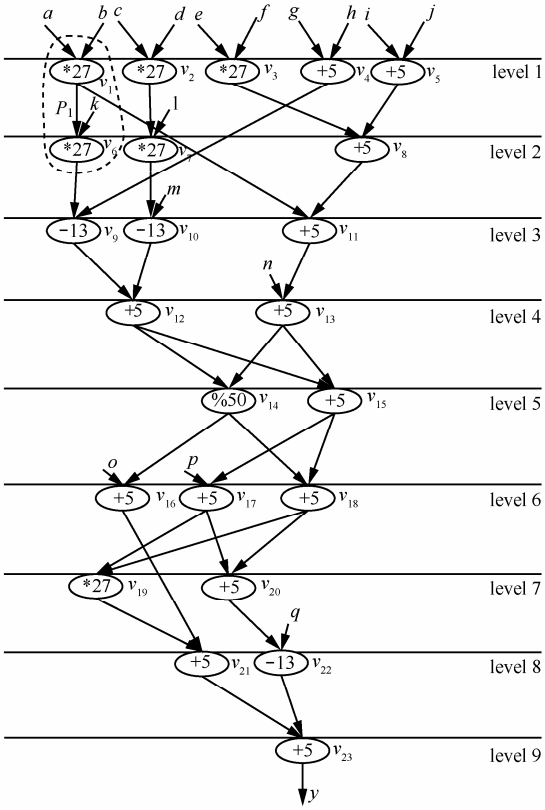
由上可知，策略 2 侧重于同时考虑了硬件碎片利用、硬件划分的吞吐量、执行延迟大节点的优先响应等因素，其目的是想同时获得任务 DFG 划分块数和所有划分块总的执行延迟的较小化。

策略 3　优先选择优先级高的节点和当前层任务节点。

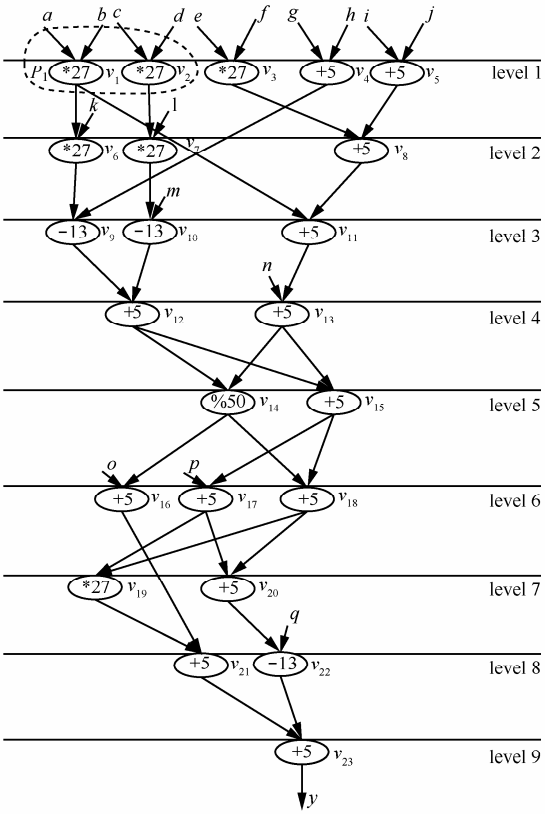
在满足资源面积和前后依赖约束的前提下，优先选择优先级大的当前层任务节点，当前层的下一层节点滞后选择；在 2 个任务节点优先级相同的情况下，优先选择当前层就绪的节点，若不存在这样



(a)一个表达式的DFG



(b)按深度划分的DFG



(c)按广度划分的DFG

图 3　解释方案 1 不合理的例子

的点，则按先来先服务(FIFO, first in first out)原则执行；对于同时就绪的任务节点，则选择优先级高的任务节点。

由上可知，策略 3 侧重于考虑任务 DFG 划分块内运算节点的并行度。

策略 4　尽量减少划分块之间的 I/O 边数。

实现策略 4 的方法有：1)每次划分首先按 DFS 的方式选择当前优先级最大的节点作为起始点划入当前块，同时更新该点直接后继的入度，如后继的入度为 0，在满足 A_{RPU} 的前提下，直接预划入当前块；如后继的入度不为 0，还要考察该点没有划入当前块的前驱能否划入，如果能，则一并预划入；2)每划入一个点，就动态计算正在划分的模块与就绪节点之间的边数，边数越大就说明该就绪节点与当前划分块联系越紧密，尽可能将其划入将有助于减少划分块之间的 I/O 边数。

由上可知，策略 4 在策略 1~策略 3 的基础上，侧重于减少划分块之间 I/O 边数。

策略 5　其他方面因素的考虑。

1) 层次大优先级高的节点应优先响应

对于不同层的任务节点要考虑其层号，这样做的目的一方面是在同等条件下，节点的层号越小越优先，另一方面是为了照顾优先级高且层次较大任务节点被优先响应，以便尽可能提高每个划分块内部运算任务节点之间的并行度。

2) 优先划入出度较大的运算节点

同等条件下，优先划入出度较大的运算节点，目的是搜索范围的扩大化使更多的节点成为就绪点，从而有可能使每次划分得以进一步优化。

3) 防止划分块间运算任务节点非法依赖关系的产生

任务 DFG 的时域划分要求入度为 0 的点或者某点的所有前驱已经被划入当前块或上一块时才能划入该点，这应在程序中用条件语句加以约束，这样做的目的是防止划分块间非法依赖关系的产生（如图 2 所示）。

由上可知，策略 5 中的 1)侧重于考虑任务 DFG 划分块内运算节点的并行度；策略 5 中的 2)侧重于优化每次划分；策略 5 中的 3)侧重于防止任务 DFG 每个划分块之间产生非法依赖。

根据以上策略，AEMO 为就绪列表中的任务构造了新的探测函数(本文约定函数值越小优先级越大)，并且根据不同参数指标的贡献大小而将其设为分母或分子。

为了便于实验比较，本文用重构硬件资源面积作为约束条件，各类运算所占的重构硬件资源数(单位：CLB)的确定参照了 XC4000E 系列 8bit FPGA(如表 1 所示)。

表 1　　**各类运算的时延和占用资源量**

运算类型	时延(单位：时钟周期)	占用硬件资源(单位:CLB)
加法	1	5
乘法	2	27
取模	4	50
减法	1	13
逻辑比较	1	17
异或	1	5
逻辑左移	1	5

探测函数构造的基本思想如下。

首先，由表 1 可知，运算任务节点执行延迟大，其所占的硬件资源数也较大，故为了保证其优先响应，应采用第 i 个运算节点 v_i 所占的硬件资源面积 w_i 和其执行延迟 d_i 的累加和作为探测函数的分母的一部分($1 \leqslant i \leqslant n$)。

其次，在任务 DFG 的划分过程中，为了尽可能地划入与正在划分模块紧密联系的就绪节点，需要动态计算当前正在划分的模块与就绪节点之间有向边的数量，其目的是尽可能地减少划分块间非原始 I/O 边数。

例 2　图 4(a)给出了一个正在划分的部分 DFG，假设 v_1 和 v_2 点已经被划入到 P_1 中(如图 4(b)所示)。现考察就绪节点 v_3 和 v_4，设 v_3 和 v_4 运算类型相同且位于同一层，在 A_{RPU} 的约束下，P_1 还可以划入 v_3 或 v_4 点中一个，P_1 与 v_3 的边数为 1，与 v_4 的边数为 2，P_1 划入 v_3 的情形如图 4(c)所示，P_1 划入 v_4 的情形如图 4(d)所示，但是图 4(d)的方案要优于图 4(c)，原因是图 4(c)的划分块间边数为 3，而图 4(d)的划分块间边数仅为 2。

因此，每次在线动态划分时可将当前正在划分的模块与就绪节点之间有向边的数量(用 s 表示)作为探测函数的分母的一部分。

第三，在任务 DFG 的划分过程中，为了达到当前层任务节点和层号小的节点被优先响应，可将任务节点层次号作为探测函数分子的一部分。但是这

样做有时会使层次大的优先级高任务节点得不到及时响应。图 5 所示的例子说明，设 v_6 为加法，v_{666} 为乘法，v_{666} 的优先级本应高于 v_6，但 v_{666} 所在的层次远大于 v_6，同时就绪时，为了保证 v_{666} 先执行可以在任务节点层次号前面乘以一个修正系数，其最大值定为 1/maxlevel，这样可以使层次大的优先级高节点优先得以响应，从而提高划分块内部的并行度。

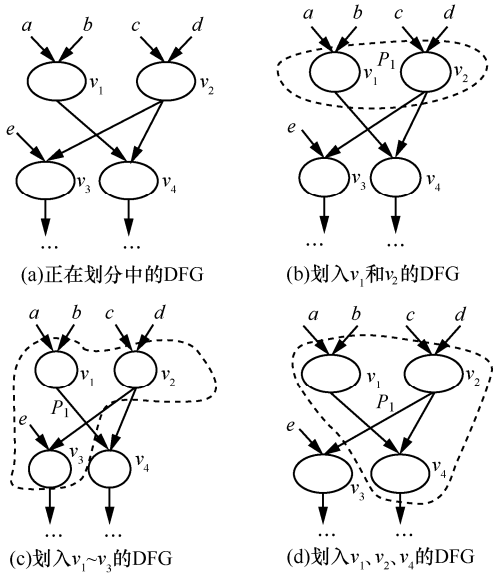


图 4 说明计算正在划分的模块与就绪节点之间的边数 s 的例子

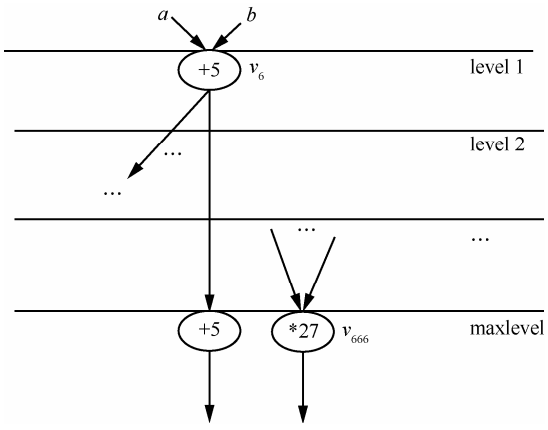


图 5 解释优先级高且层次较大节点优先响应的例子

最后，在进行任务 DFG 时域划分过程中，待划的就绪列表节点的前驱可被划入当前块或已经被划入到上一个划分中了，所以任务节点被调度时其入度必须为 0，这一点可有条件判断语句来实现。但是出度大的任务节点被动态划入当前块时，会使更多的点成为就绪点，选择节点的空间增大有可能优化划分。图 6 所示例子说明，v_1 和 v_2 的优先级相同，若按 FIFO 先划入 v_1，则只能使 v_3 成为就绪点，但是另外一种可能的方法是先划入 v_2，这样可以使 v_4~v_9 同时成为就绪点，从而可能优化划分。所以可以将任务 DFG 的每个节点的出度设定为探测函数的分子或分母，同时为了进行动态调整可以设定一修正系数。

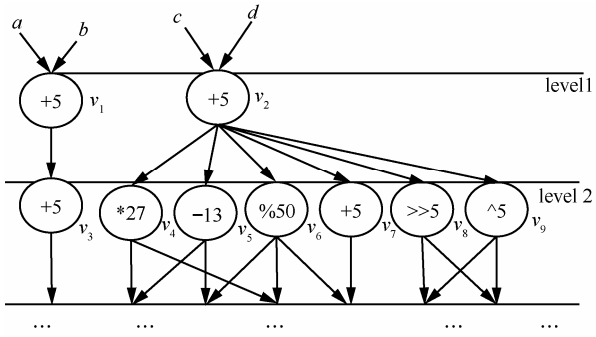


图 6 解释出度大的点被优先划分的例子

这样，对于任一个运算节点 v_i，其探测函数 prior_assigned(v_i)可以设定为如下 2 种形式之一

$$\text{prior\_assigned}(v_i)=(\alpha\times\text{level}(v_i)-\beta\times\text{outdegree}(v_i))\times(1/(w_i+\gamma\times s+d_i)) \quad (1)$$

$$\text{prior\_assigned}(v_i)=(\alpha\times\text{level}(v_i))\times(1/(w_i+\gamma\times s+d_i+\beta\times\text{outdegree}(v_i))) \quad (2)$$

其中，outdegree(v_i)表示节点 v_i 的出度，式(1)把 outdegree(v_i)作为分子并要做减法，因为分母相同的情况下，分子越小函数值越小，prior_assigned(v_i)越小越优先；同理，式(2)把 outdegree(v_i)作为分母并要做加法，目的是使分母变大。函数的其他参数说明如下：maxlevel 表示一个 DFG 的最大层号；maxoutdegree=max{outdegree(v_i), $1 \leqslant i \leqslant n$}；maxinputoutputside 表示一个 DFG 非原始输入输出边数之和；level(v_i)表示节点 v_i 的层次数，α、β 和 γ 为调整系数，α 取值范围为[0,1/maxlevel]，β 取值范围为[0, maxoutdegree]，γ 取值范围为[0, maxinput-outputside]。prior_assigned(v_i)的值越小表示 v_i 的优先级越高，即被划入某个划分的可能性就越大。

为了验证相关划分算法，构造了一组划分基准程序集。它们由 2 部分构成，一是由 LBP 和 CBP 算法所用的基准中值滤波器 MEDIAN、二叉树比较器 BTREE32、一维离散余弦变换 8 次展开 DCT8 和 4×4 矩阵运算 MATRIX4、MOTP 算法所用的基准二阶差分方程 SODE、快速数据加密算法 FEAL、快速离散余弦变换 FDCT、快速离散余弦变换 6 次展开 FDCT6、椭圆波形滤波器 EWF、椭圆波形滤

波器 6 次展开 EWF6，此外还增加了快速傅里叶变换 4 次展开 FFT4(fast Fourier transform 4)、快速傅里叶变换 8 次展开 FFT8(fast Fourier transform 8)2 个基准，这些基准的操作单元数量如表 2 所示。第 2 部分基准程序采用了文献[14]中列出的且与 ESL 和 LSCBP 所用基准特征相同的 10 个随机图。程序的开发环境是 VC++6.0，运行环境是 Windows XP，PC 的处理器为 Intel(R) Core(TM) i3 CPU, 主频为 2.26GHz, 内存为 2GB。

表 2　　划分基准程序集

划分用例	操作单元数量							
	总数	加法	减法	乘法	取模	逻辑比较	异或	左移
SODE	11	2	2	6	—	1	—	—
FEAL	34	6	—	—	4	—	20	4
FFT4	12	4	4	4	—	—	—	—
FFT8	36	12	12	12	—	—	—	—
EWF	34	28	—	6	—	—	—	—
EWF6	204	168	—	36	—	—	—	—
FDCT	42	13	13	16	—	—	—	—
FDCT6	252	78	78	96	—	—	—	—
MATRIX4	112	48	—	64	—	—	—	—
DCT8	90	40	16	34	—	—	—	—
MEDIAN	19	—	—	—	—	19	—	—
BTREE32	31	—	—	—	—	31	—	—

随机选取的 A_{RPU} 分别为 56 CLB、64CLB、75 CLB，取 α=1/maxlevel，β=γ=1，统计划分块间输出边的数量时，如块间一个运算节点的输出边数超过 1，也只算一条边，这样做的理由是避免划分块间同一个节点的运算结果被多次存储。设 M 表示一个 DFG 的划分块数，SD 表示一个 DFG 所有划分块执行延迟总和，N 表示一个 DFG 所有划分块之间的非原始 I/O 边数。

AEMO 算法描述如图 7 所示。

4.2　算法时间复杂度分析

AEMO 算法在执行前，一个 DFG 总的运算节点数、每个运算节点的执行延迟、运算类型和占用的资源数，每个节点前驱与后继的个数及列表均已知。

设一个 DFG 有 n 个运算节点，assign_level()求出每个运算节点层次及该 DFG 的最大层次号，时间复杂度为 $O(n^2)$；每次划分所用到的就绪任务节点，在入队时均要通过探测函数 priority_assigned()计算其优先级，其时间复杂度为 $O(n^2)$；用快速排序函数 quicksort()对每次计算出的探测函数值按从小到大

算法名称　AEMO 算法
输入：一个 DFG
输出：一个划分后的 DFG 及其 M、SD、N 值
约束：A_{RPU}；DFG 所有划分块之间均不产生非法依赖关系；阈值 value=10
目标：获得最小的 M；较小的 SD；尽可能小的 N
1) Initializing: inputing data table by function Init(); the partitioning number variable j=0, filling−area variable *area_filled*=0, the number of node accumulation variable n=0, the area of hardware fragments by DFS pre−partitioning *area*1=0, the partitioning mark by DFS *flag*1=0, the initialization of A_{RPU}, *Area_cost* =0; *area_consumption*=0;
float a[nodenumber]={0.0}; /*nodenumber is the number of nodes in a DFG*/
2) Computing the level number each node of the DFG and the maximum level of the DFG by function assign_levels();
3) Computing the priority weight value for each node of the DFG by detecting function
priority_assigned(int v_i, int j);
4) for i=1 to nodenumber
node[i].flag=0;
end for
5) for i=1 to nodenumber
node[i].priority=priority_assigned(i, j);
a[i]=node[i].priority;
end for
6) Sorting the priority weight value in ascending order by function quicksort();
7) while (n<nodenumber)
8) for i=1 to nodenumber
9) for v_i =1 to nodenumber
10) if (a[i]=node[v_i].priority&& a[i]!=∞)
11) if (flag1=0) /* partitioned by DFS*/
Area_cost =node[v_i].size;
12) if (*area_filled*+*Area_cost* <=A_{RPU})
node[v_i].partition=j; *area_ consumption* +=node[v_i].size; n++; node[v_i].flag=1;
node[v_i].flag2=1;/* the temporarily mark of partitioning nodes*/
area_filled=*area_filled*+*Area_cost*; *area*1=A_{RPU}−*area_filled;*
Updating the indegree of successor nodes;
13) if ((the indegree of the current partitioned successor nodes k=0)&&
(*area_filled*+*Area_cost* <=A_{RPU}))
node[k].partition=j; *area_ consumption* +=node[k].size; n++;
node[k].flag=1; node[k].flag2=1;*area_filled*=*area_filled*+ *Area_cost*;
*area*1=A_{RPU}−*area_filled*; Updating the indegree of successor nodes;
14) else if ((the indegree of the current partitioned successor nodes k!=0) &&
(*area_filled*+*Area_cost*<=A_{RPU}))
15) for j=1 to node[k]. precursor_ number/* scanning the precursor node */
l = node[k].precursor[j];
16) if (node[l].flag=0&&node[l].indegree=0) /*the precursor of current nodes can be partitioned*/
node[l].partition=j; *area_ consumption* +=node[l].size; n++;
node[l].flag=1;node[l].flag2=1;area_filled=area_filled+Area_cost;
*area*1=A_{RPU}−*area_filled*; Updating the indegree of successor nodes;
17) end if
18) end for
Computing the priority weight value each node of the DFG by detecting function priority_assigned(int v_i, int j); Sorting the priority weight value
in ascending order by function quicksort();/* goto 29*/
19) end if
20) end if
21) else if (flag1=1) /* partitioned by BFS */
Area_cost =node[v_i].size;

22) if((*area_filled*+*Area_cost*)<=A_{RPU})
node[v_i].partition=j; n++; node[v_i].flag=1; *area_filled*= *area_filled*+ *totalcost*;
Updating the indegree of successor nodes;
Computing the priority weight value each node of the DFG by detecting function;priority_assigned(int v_i, int j);
Sorting the priority weight value in ascending order by function quicksort();
else partitioned greedy by BFS; /* goto 21*/
23) end if
24) end if
25) end if
26) end if
27) end for
28) end for
29) if ((*area*1>=10)&&(flag1=0))
recovering the partitioned nodes to the former state; flag1=1; /* goto 21*/
30) else if (*area*1<10) partitioned greedy by DFS; /* goto 11*/
31) end if
32) if (i=nodenumber)
j++; Computing the priority weight value for each node of the DFG by detecting function;
priority_assigned(int v_i, int j); Sorting the priority weight value in ascending order by function quicksort(); *area_filled*=0; flag1=0; /*goto 11*/
33) end if
M=j;
34) end while
35) Computing N and SD by function edges() and delays();
36) end AEMO.

图 7　AEMO 算法

排序，其最好情况时间复杂度为 $O(n)$，最坏情况为 $O(n^2)$，平均时间复杂度为 $O(n\log 2^n)$；用函数 edges()求一个划分后的 DFG 块间总的非 I/O 边数，时间复杂度为 $O(n^2)$；假如一个 DFG 被划分为 M 块，用直接递归调用来求一个划分后的 DFG 所有模块执行总延迟函数 delays()的时间复杂度为 $O(Mn)$。

AEMO 算法在对 DFG 进行划分时，首先要找到当前优先级最高的待划分节点，故每次在划分前通过探测函数计算每个就绪节点优先级的值并排序，找到优先级最高的待划分节点放入当前的划分中，这是一个循环迭代的过程，加上判断 DFG 的运算节点是否全部被划分完的约束，共耗费的时间复杂度为 $O(n^4\log 2^n)$。综上，AEMO 算法的平均时间复杂度为 $O(n^4\log 2^n)$。

为了比较 2 个探测函数的优劣性，AEMO 算法分别用式(1)和式(2)实现，再用如表 2 所示的划分基准程序集进行了实验，结果如表 3 所示，其中 AEMO1 算法采用的探测函数是式(1)。表 3 分别给出了 A_{RPU}=56、64 和 75 时划分基准程序采用 2 个不同划分算法所得到的划分结果，对应于每个指标(M、N、SD)下每个划分算法和改进的百分比(Δ%)所列出的 3 列内容。以 SODE 为例，当 A_{RPU}=56 时，采用 AEMO1 算法所获得的 M=5，而采用 AEMO 算法所获得的 M=4，因此，Δ%=−20.0%(负值表示改进)。通过对比可知，用式(2)所获得的结果符合本文的目标，即 M 要最小化，同时获得一个较小 SD，且尽可能减少 N。因此，本文的后续工作均基于采用式(2)探测函数的 AEMO 算法。

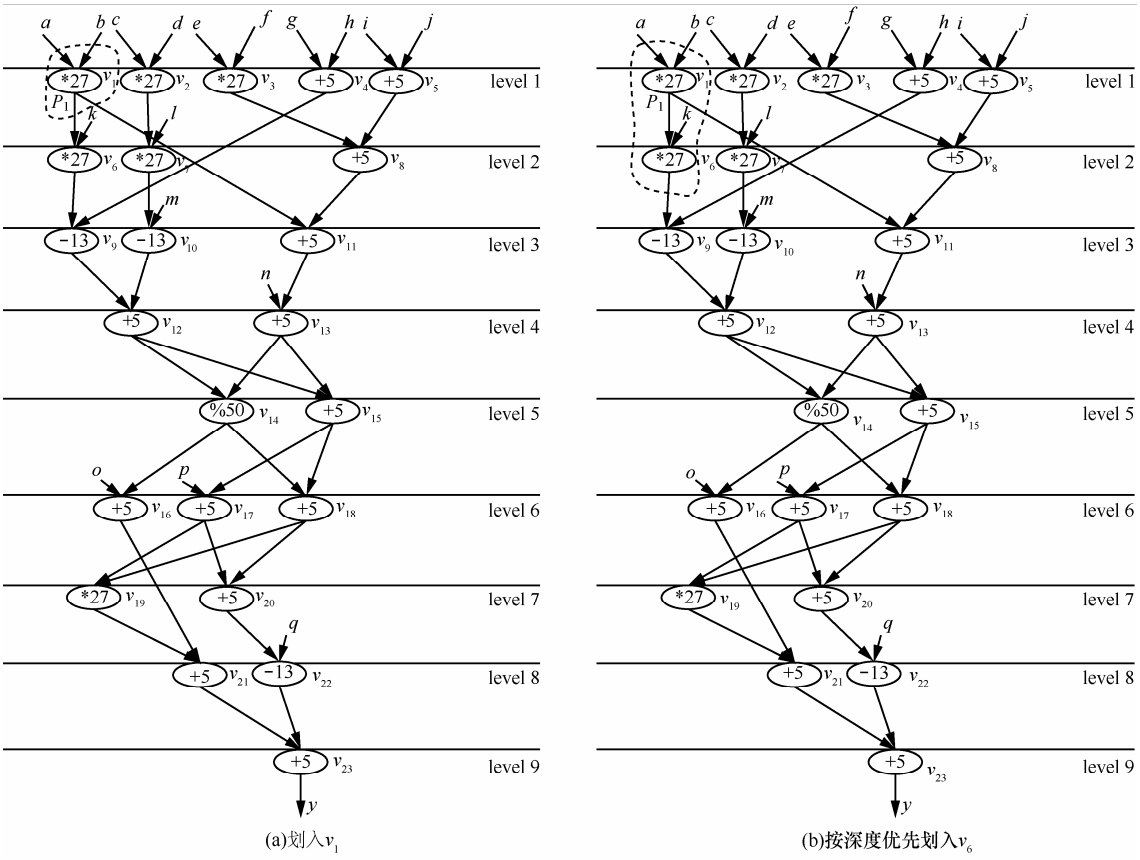
4.3 一个算术表达式划分实例分析

针对图 3(a)算术表达式的 DFG，划分前节点有 23 个，设 A_{RPU}=65，maxlevel=9，α=1/9，β=γ=1，面积填充变量 area_filled=0，硬件碎片面积变量 area1= 0(按 DFS)，value=10，图 8 和图 9 给出了根据 AEMO 算法每一个步骤的划分子图。

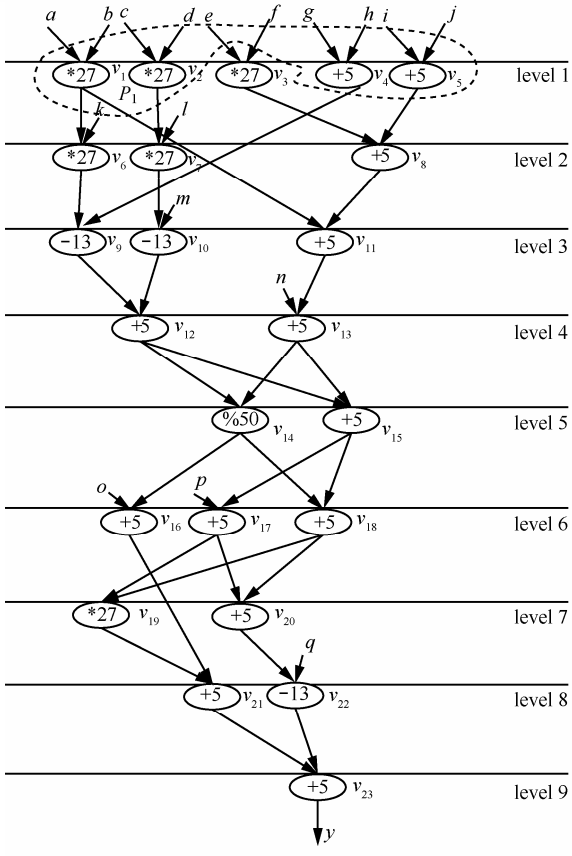
1) 图 3(a)入度为 0 的就绪节点为 v_1~v_5，w_1= w_2=w_3=27，w_4=w_5 =5，d_1=d_2=2，d_3= d_4= d_5=1，就绪节点 v_1、v_2、v_3、v_4、v_5 与当前正在划分的模块有向边的数量均为 0，level(v_1)= level(v_2)= level(v_3)= level(v_4)= level(v_5)= 1，outdegree(v_1)=2，outdegree(v_2)= outdegree(v_3)= outdegree(v_4)= outdegree(v_5)=1，prior_ assigned(v_1)= (α×level(v_1))× (1/(w_1+ γ×s+d_1+ β× outdegree(v_1))=((1/9)×1))×(1/(27+1×0+2+1×2))≈0.003 6，同理，prior_assigned(v_2)= prior_ assigned(v_3) ≈0.003 7，prior_assigned(v_4) =prior_ assigned (v_5) ≈ 0.015 9。由于 prior_assigned(v_1)最小，故以 v_1 作为起点进行 DFS 贪婪划分，v_1 被划入 P_1(如图 8(a)所示)，则划分后剩余节点集合 V={v_2，v_3，v_4，v_5，v_6，v_7，v_8，v_9，v_{10}，v_{11}，v_{12}，v_{13}，v_{14}，v_{15}，v_{16}，v_{17}，v_{18}，v_{19}，v_{20}，v_{21}，v_{22}，v_{23}}，P={P_1}，P_1={v_1}，area_filled=27，area1= A_{RPU}−area_ filled =65−27=38，v_6 的入度更新为 0，v_{11} 的入度更新为 1。

表 3 A_{RPU}=56、64、75 时 AEMO1 与 AEMO 的划分结果

划分用例	M									N									SD								
	AEMO1			AEMO			Δ%			AEMO1			AEMO			Δ%			AEMO1			AEMO			Δ%		
SODE	5	4	4	4	4	4	−20	0	0	4	5	5	3	5	5	−25	0	0	13	11	10	13	11	10	0	0	0
FEAL	7	6	6	7	6	5	0	0	−17	15	14	13	15	12	15	0	−14	15	29	25	27	28	28	26	−3	12	−4
FFT4	4	3	3	4	3	3	0	0	0	6	7	7	4	7	7	−33	0	0	9	9	6	10	9	6	11	0	0
FFT8	11	9	8	10	9	8	−9	0	0	20	23	21	26	24	20	30	4	−5	31	27	23	19	25	22	−39	−7	−4
EWF	7	6	5	6	5	5	−14	−17	0	18	15	16	19	16	13	6	7	−19	30	25	25	26	25	21	−13	0	−16
EWF6	38	29	25	35	29	25	−8	0	0	109	97	102	115	114	102	6	18	0	150	118	107	111	96	94	−26	−19	−12
FDCT	13	12	10	13	12	10	0	0	0	26	25	20	30	26	24	15	4	20	26	28	30	30	31	29	−15	11	−3
FDCT6	75	72	60	75	68	56	0	−6	−7	152	155	118	163	168	155	7	8	31	186	168	173	192	170	144	3	1	−17
MATRIX4	37	33	32	37	33	32	0	0	0	76	48	36	75	48	36	−1	0	0	79	98	109	80	98	109	1	0	0
DCT8	30	22	21	25	22	19	−17	0	−10	51	43	42	60	47	47	18	9	12	82	64	67	64	67	61	−22	5	−9
MEDIAN	7	7	5	7	7	5	0	0	0	10	16	12	11	14	12	−10	−13	0	18	9	13	17	11	13	−6	−22	0
BTREE32	11	11	8	11	11	8	0	0	0	10	29	16	10	28	16	0	−3	0	21	12	17	21	13	17	0	8	0
平均 Δ%	—	—	—	—	—	—	−14	−12	−11	—	—	—	—	—	—	1	2	9	—	—	—	—	—	—	−11	−1	−9



(a)划入v_1　　(b)按深度优先划入v_6



(c)按广度划入v_1、v_2、v_4、v_5

图 8　图 3(a)算术表达式 DFG 的第一个划分过程

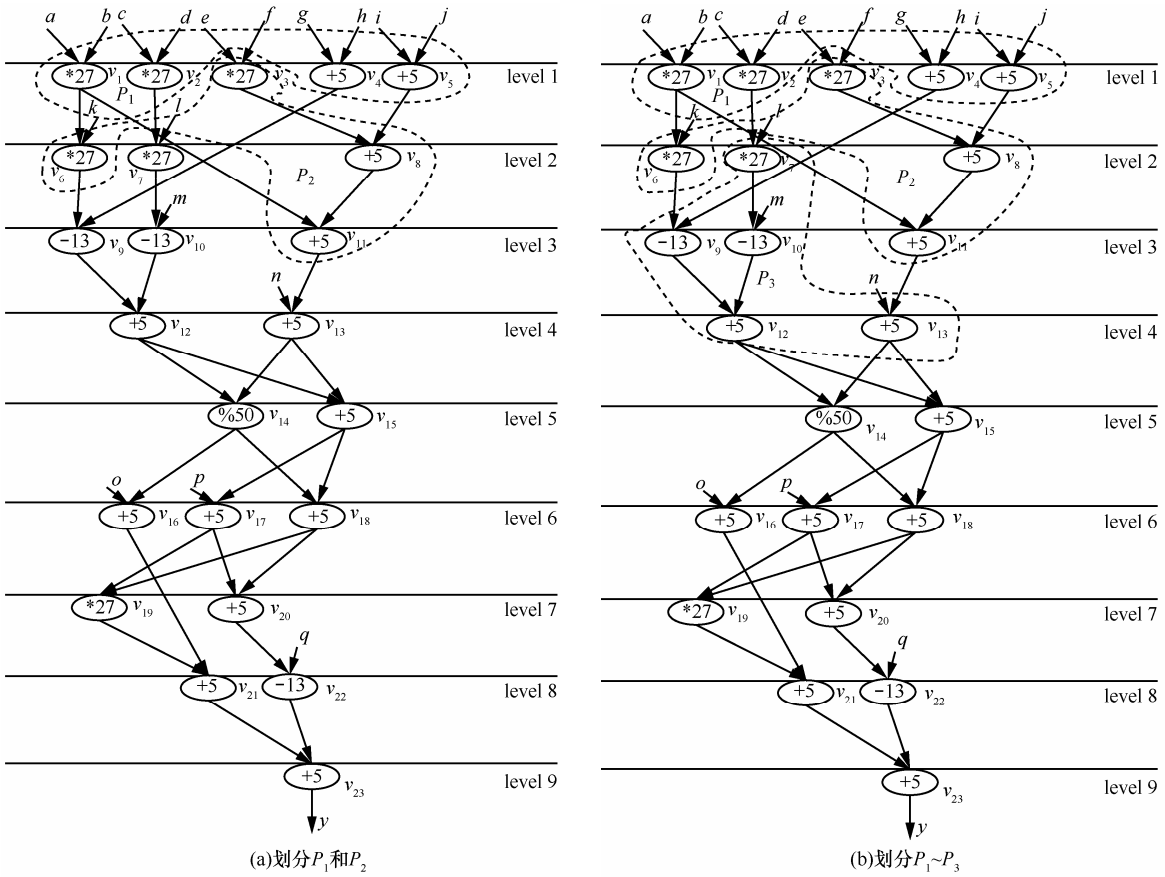


图 9　图 3(a)算术表达式 DFG 的后续划分过程

2) 由于 v_6 为 v_1 的入度已经更新为 0 的后继，在满足 A_{RPU} 前提下，按 DFS 直接预划入(如图 8(b)所示)，划分后剩余节点集合 V={v_2，v_3，v_4，v_5，v_7，v_8，v_9，v_{10}，v_{11}，v_{12}，v_{13}，v_{14}，v_{15}，v_{16}，v_{17}，v_{18}，v_{19}，v_{20}，v_{21}，v_{22}，v_{23}}，P={P_1}，P_1={v_1，v_6}，area_filled=54，area1=11，v_9 的入度更新为 1；如果 v_9 和其前驱 v_4 一并划入，则 w_9+w_4=18>area1，由于 *area*1=11 大于阈值 value，故本次按 DFS 划分不成功，把划入的节点退回去，相关节点的信息复原。

3) 从 v_1 开始，按权值进行 BFS 层贪婪划分。由图 8(a)可知，就绪节点为 v_2、v_3、v_4、v_5、v_6 分别计算其权值 prior_assigned(v_2)=prior_assigned(v_3) ≈ 0.003 7，prior_assigned(v_4)=prior_assigned(v_5)≈0.015 9，prior_assigned (v_6)=0.007 2，由于 v_2、v_3 具有相同的优先级，因此按 FIFO 原则先划入 v_2，划分后剩余节点集合 V={v_3，v_4，v_5，v_6，v_7，v_8，v_9，v_{10}，v_{11}，v_{12}，v_{13}，v_{14}，v_{15}，v_{16}，v_{17}，v_{18}，v_{19}，v_{20}，v_{21}，v_{22}，v_{23}}，P={P_1}，P_1={v_1，v_2}，area_filled=54，area1=11，再依次按层贪婪划入 v_4、v_5，area_filled=64，area1=1，节点调度序列依次为 v_1、v_2、v_4、v_5，则 V={v_3，v_6，v_7，v_8，v_9，v_{10}，v_{11}，v_{12}，v_{13}，v_{14}，v_{15}，v_{16}，v_{17}，v_{18}，v_{19}，v_{20}，v_{21}，v_{22}，v_{23}}，P={P_1}，P_1={v_1，v_2，v_4，v_5}，其划分形状如图 8 所示。

4) 由图 8(c)可知，就绪节点为 v_3、v_6、v_7、v_8，分别计算其权值，prior_assigned (v_3)的值小，优先级高，在 A_{RPU} 约束下，以 v_3 为起点按 DFS 依次划入 v_8、v_{11}、v_{13}，因为 area_filled=42，area1=65−42=23>value，故以 v_3 为起点按权值进行 BFS 层贪婪划分，划分 P_2 节点调度序列依次为 v_3、v_6、v_8、v_{11}，划分后剩余节点集合 V={v_7，v_9，v_{10}，v_{12}，v_{13}，v_{14}，v_{15}，v_{16}，v_{17}，v_{18}，v_{19}，v_{20}，v_{21}，v_{22}，v_{23}}，则 P={P_1，P_2}，P_1={v_1，v_2，v_4，v_5}，P_2={ v_3，v_6，v_8，v_{11}}，v_7、v_9、v_{13} 的入度更新为 0。P_2 的划分形状如图 9(a)所示。

5) 同理，按 DFS 以优先级最高 v_7 为起点，依次划入 v_7、v_{10}，因为 v_{12} 的前驱可以划入，故 v_9、v_{12} 一并划入，而 v_{12}、v_{14} 不能同时划入，故计算 area1 =A_{RPU}−(w_7+w_{10}+w_9+w_{12})=65−(27+13+13+5)=7<value，更新 v_{13}、v_{14} 的入度为 1，故本次划分按 DFS 进行，更新就绪点的探测函数值，发现 v_{12} 可以贪婪划入，综上，划分 P_3 节点调度序列依次为 v_7、v_{10}、v_9、v_{12}、v_{13}，划分后剩余节点集合 V={v_{14}，v_{15}，v_{16}，v_{17}，v_{18}，v_{19}，v_{20}，v_{21}，v_{22}，v_{23}}，则 P={P_1，P_2，P_3}，P_1={v_1，v_2，v_4，v_5}，P_2={v_3，v_6，v_8，v_{11}}，P_3={v_7，v_{10}，v_9，v_{12}，v_{13}}，v_{14}、v_{15} 的入度更新为 0。P_3 的划分形状如图 9(b)所示。

6) 同理，按 DFS 以优先级最高 v_{14} 为起点，其节点调度序列依次为 v_{14}、v_{16}、v_{15}、v_{18}，则 P={P_1，P_2，P_3，P_4}，P_1={ v_1，v_2，v_4，v_5}，P_2={ v_3，v_6，v_8，v_{11}}，P_3={ v_7，v_{10}，v_9，v_{12}，v_{13}}，P_4={v_{14}，v_{16}，v_{15}，v_{18}}，area_filled=65，area1=0(area1<10)，故按 DFS 执行。需要说明的是，在选取 v_{18} 和 v_{17} 时，正在划分的模块与就绪节点之间的边数 s 起了重要的作用，α=1/9，γ=β=1，level(v_{17})= level(v_{18})=6，就绪节点 v_{17}、v_{18} 与当前正在划分的模块有向边的数量分别为 1 和 2，outdegree(v_{17})= outdegree(v_{18})=2，w_{17}= w_{18}=5，d_{17}= d_{18}=1，prior_assigned(v_{17}) =2/27≈0.074 1，prior_assigned(v_{18}) =1/15≈0.066 7，故划入 v_{18}，P_4 的划分形状如图 9(c)所示，其中划入 v_{18} 与划入 v_{17} 相比划分块间的边数少了一条。

7) 同理，按 DFS 贪婪划分得到 P_5 如图 10 所示，其节点调度序列依次为 v_{17}、v_{20}、v_{22}、v_{19}、v_{21}、v_{23}，则 P={P_1，P_2，P_3，P_4，P_5}，P_1={v_1，v_2，v_4，v_5}，P_2={ v_3，v_6，v_8，v_{11}}，P_3={ v_7，v_{10}，v_9，v_{12}，v_{13}}，P_4={ v_{14}，v_{15}，v_{18}，v_{16}}，P_5={ v_{17}，v_{20}，v_{22}，v_{19}，v_{21}，v_{23}}，V={Φ}，划分结束。

5　实验及其分析

本文采用 C 语言实现了 ESL、CBP、LSCBP、MOTP、LBP、AEMO 6 种时域划分算法，所用的划分基准程序集，统计划分块间输出边的数量方式，A_{RPU} 的 3 种取值，参数 α、β、γ 的选取等均同 4.1 节所述。CBP 和 LBP、LSCBP 算法的模块计数均包括第 0 层输入的划分。下面用 AEMO 算法分别与 LBP、CBP、ELS、MOTP、LSCBP 等算法做了比较。

5.1　AEMO 算法与 LBP 算法的比较

用 AEMO 算法划分图 3(a)的 DFG 的结果如图 10 所示(A_{RPU}=65)，所获得的结果为 M=5、N=11、SD=20，而用 LBP 算法划分的结果如图 11 所示，结果为 M=7、N=13、SD=17。

因为 LBP 算法在划分过程中，一遇到不满足硬件资源约束的运算节点就开辟新的划分块，并且把第 0 层输入作为一个划分块，这些导致了 M 较大；LBP 算法追求每个划分块 SD 的较小化，但是这将使 N 较大。AEMO 算法消除了 LBP 算法的缺点，同时对 SD 进行了折中考虑。其他划分基准对比实验的数据如表 4 所示，相对于 LBP 算法，AEMO 算法对

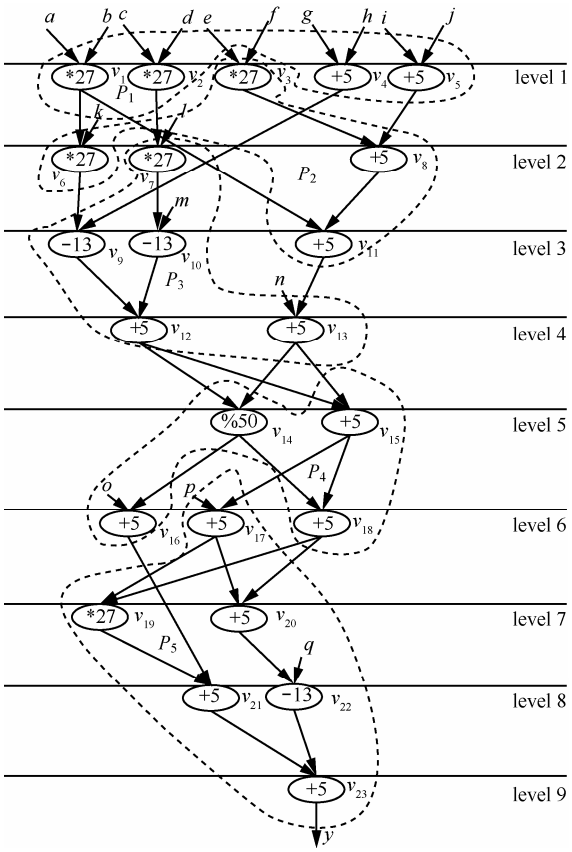


图 10　AEMO 划分图 3(a)的结果

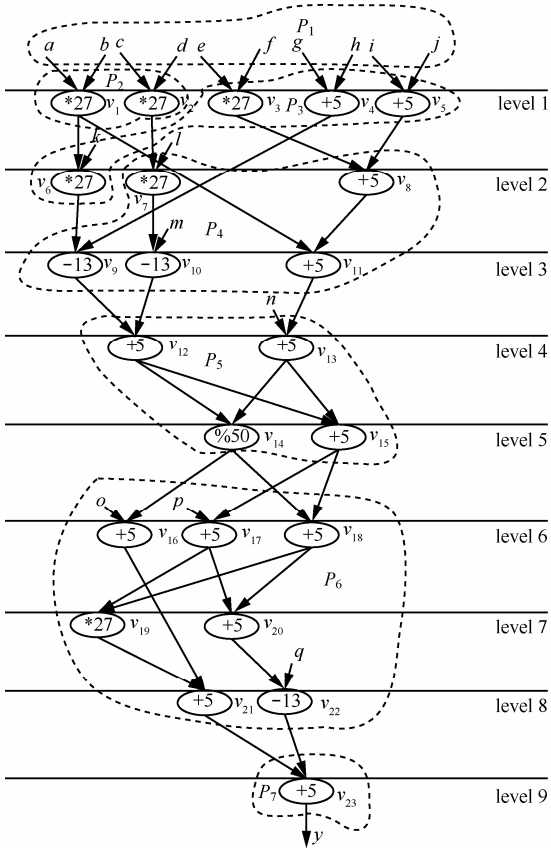


图 11　LBP 划分图 3(a)的结果

表 4　　**A_{RPU}=56、64、75 时 AEMO 算法与 LBP 算法的划分结果**

划分用例	*M*						*N*						*SD*								
	LBP			AEMO			*Δ*%			LBP			AEMO			*Δ*%			LBP			AEMO			*Δ*%		
SODE	6	5	5	4	4	4	−33	−20	−20	7	7	7	3	5	5	−57	−29	−29	10	8	8	13	11	10	30	38	25
FEAL	9	8	7	7	6	5	−22	−25	−29	21	21	19	15	12	15	−29	−43	−37	24	22	23	28	28	26	17	27	30
FFT4	5	4	4	4	3	3	−20	−25	−25	8	6	7	4	7	7	−50	−17	−14	6	8	6	10	9	6	67	13	33
FFT8	12	11	10	10	9	8	−17	−18	−20	28	28	28	26	24	20	−7	−14	−29	18	18	15	19	25	22	6	39	47
EWF	9	7	6	6	5	5	−33	−29	−17	24	20	15	19	16	13	−21	−20	−20	23	19	18	26	25	21	13	32	22
EWF6	42	32	28	35	29	25	−17	−9	−11	172	174	166	115	114	102	−33	−35	−41	73	55	51	111	96	94	52	75	88
FDCT	15	13	13	13	12	10	−13	−8	−23	34	33	33	30	26	24	−12	−21	−33	24	22	22	30	31	29	25	41	46
FDCT6	79	74	66	75	68	56	−5	−8	−15	204	204	204	163	168	155	−20	−18	−27	129	127	115	192	170	144	49	34	35
MATRIX4	38	37	36	37	33	32	−3	−11	−11	96	96	95	75	48	36	−22	−50	−62	70	68	68	80	98	109	14	44	60
DCT8	27	25	23	25	22	19	−7	−12	−17	79	82	77	60	47	47	−24	−43	−33	47	42	41	64	67	61	36	60	29
MEDIAN	8	8	6	7	7	5	−13	−13	−17	15	15	14	11	14	12	−27	7	−21	11	11	11	17	11	13	55	0	18
BTREE32	12	12	9	11	11	8	−8	−8	−11	29	29	28	10	28	16	−66	−3	−43	12	12	9	21	13	17	75	8	89
平均 *Δ*%	—	—	—	—	—	—	−16	−16	−18	—	—	—	—	—	—	−31	−24	−32	—	—	—	—	—	—	37	37	44

M 和 *N* 均有了一定程度的改进，但是 *SD* 不如 LBP 算法。

5.2　AEMO 算法与 CBP 算法的比较

因为 CBP 算法尽可能地将联系紧密的运算节点放到一个模块中，但对每个划分块内运算节点的并行度考虑不足，因此 *N* 较小，*SD* 较大；*M* 较大的原因同 LBP 算法。AEMO 算法与 CBP 算法比较的实验数据如表 5 所示。

5.3　AEMO 算法与 ESL 算法的比较

因为 ESL 算法综合考虑了划分块间的通信代价、关键路径等因素，其划分路径多数是沿着深度优先搜索方向延展的，对并行因素考虑不够，而且没有考虑任务 DFG 划分中可能产生的硬件碎片问题，因此，相对于 AEMO 算法，ESL 算法获得较小的 N，但是 M、SD 均较大。AEMO 算法与 ESL 算法比较的实验数据如表 6 所示。

5.4　AEMO 算法与 MOTP 算法的比较

因为 MOTP 算法折中考虑了 M、SD 和 N 等 3 个目标，因此获得了较小 M，但在每次划分时，没有进行硬件资源面积的估算，这就有可能会产生硬件碎片，所以相比于 AEMO 算法，M 值仍然较大。AEMO 算法与 MOTP 算法比较的实验数据如表 7 所示。

表 5　　A_{RPU}=56、64、75 时 AEMO 算法与 CBP 算法的划分结果

划分用例	*M*									*N*									*SD*								
	CBP			AEMO			*Δ*%			CBP			AEMO			*Δ*%			CBP			AEMO			*Δ*%		
SODE	5	5	5	4	4	4	−20	−20	−20	3	3	4	3	5	5	0	67	25	13	13	10	13	11	10	0	−15	0
FEAL	8	8	7	7	6	5	−13	−25	−29	16	14	13	15	12	15	−6	−14	15	30	31	29	28	28	26	−7	−10	−10
FFT4	5	5	4	4	3	3	−20	−40	−25	5	4	5	4	7	7	−20	75	40	11	13	10	10	9	6	−9	−31	−40
FFT8	12	12	10	10	9	8	−17	−25	−20	20	18	17	26	24	20	30	33	18	33	40	30	19	25	22	−42	−38	−27
EWF	8	7	6	6	5	5	−25	−29	−17	15	10	10	19	16	13	27	60	30	32	35	29	26	25	21	−19	−29	−28
EWF6	38	32	28	35	29	25	−8	−9	−11	90	70	70	115	114	102	28	63	46	177	190	174	111	96	94	−37	−50	−46
FDCT	15	13	12	13	12	10	−13	−8	−17	28	26	23	30	26	24	7	0	4	40	37	34	30	31	29	−25	−16	−15
FDCT6	85	73	62	75	68	56	−12	−7	−10	160	158	138	163	168	155	2	6	12	235	213	199	192	170	144	−18	−20	−28
MATRIX4	39	35	34	37	33	32	−5	−6	−6	63	47	51	75	48	36	19	2	29	89	102	108	80	98	109	−10	−4	1
DCT8	30	23	23	25	22	19	−17	−4	−17	42	38	33	60	47	47	43	24	42	95	80	81	64	67	61	−33	−16	−25
MEDIAN	8	8	6	7	7	5	−13	−13	−17	11	11	11	11	14	12	0	27	9	17	17	15	17	11	13	0	−35	−13
BTREE32	12	12	9	11	11	8	−8	−8	−11	10	10	17	10	28	16	0	180	6	21	21	19	21	13	17	0	−38	−11
平均 *Δ*%	—	—	—	—	—	—	−14	−16	−17	—	—	—	—	—	—	14	48	23	—	—	—	—	—	—	−22	−25	−22

表 6　　A_{RPU}=56、64、75 时 AEMO 算法与 ESL 算法的划分结果

划分用例	*M*									*N*									*SD*								
	ESL			AEMO			*Δ*%			ESL			AEMO			*Δ*%			ESL			AEMO			*Δ*%		
SODE	5	4	4	4	4	4	−20	0	0	5	5	5	3	5	5	−40	0	0	13	12	11	13	11	10	0	−8	−9
FEAL	7	6	5	7	6	5	0	0	0	14	13	10	15	12	15	7	−8	50	32	30	30	28	28	26	−13	−7	−13
FFT4	4	3	3	4	3	3	0	0	0	5	3	3	4	7	7	−20	133	133	10	13	12	10	9	6	0	−31	−50
FFT8	12	10	8	10	9	8	−17	−10	0	20	20	15	26	24	20	30	20	33	34	38	38	19	25	22	−44	−34	−42
EWF	6	6	5	6	5	5	0	−17	0	15	13	13	19	16	13	27	23	0	30	26	31	26	25	21	−13	−4	−32
EWF6	36	31	25	35	29	25	−3	−7	0	78	88	76	115	114	102	47	30	34	177	158	148	111	96	94	−37	−39	−37
FDCT	14	12	10	13	12	10	−7	0	0	26	26	25	30	26	24	15	0	−4	40	34	35	30	31	29	−25	−9	−17
FDCT6	77	68	56	75	68	56	−3	0	0	161	158	142	163	168	155	1	6	9	219	194	187	192	170	144	−12	−12	−23
MATRIX4	48	33	32	37	33	32	−23	0	0	61	51	46	75	48	36	23	−6	−22	128	104	106	80	98	109	−38	−6	2.8
DCT8	27	23	20	25	22	19	−7	−4	−5	49	48	41	60	47	47	22	−2	15	84	76	73	64	67	61	−24	−12	−16
MEDIAN	7	7	5	7	7	5	0	0	0	12	12	11	11	14	12	−8	17	9	16	16	15	17	11	13	6	−31	−13
BTREE32	11	11	8	11	11	8	0	0	0	18	18	17	10	28	16	−44	56	−6	19	19	19	21	13	17	11	−32	−11
平均 *Δ*%	—	—	—	—	—	—	−11	−9	−5	—	—	—	—	—	—	5	27	25	—	—	—	—	—	—	−19	−19	−22

5.5 AEMO 算法与 LSCBP 算法的比较

LSCBP 算法是对 CBP 算法的改进，消除了任务 DFG 划分过程中产生的硬件碎片问题，在追求划分块间的边数较小化的同时，简单考虑了运算任务的并行性，效果较好。但是，LSCBP 算法把第 0 层输入作为一个划分块，且每次划分时没有进行硬件资源面积的估算，因此导致 *M* 较大，*SD* 考虑仍然不足，所以相比于 AEMO 算法，*M* 和 *SD* 均较大。AEMO 算法与 LSCBP 算法比较的实验数据如表 8 所示。

表 7 A_{RPU}=56、64、75 时 AEMO 算法与 MOTP 算法的划分结果

划分用例	*M*									*N*									*SD*								
	MOTP			AEMO			*Δ*%			MOTP			AEMO			*Δ*%			MOTP			AEMO			*Δ*%		
SODE	5	4	4	4	4	4	−20	0	0	7	5	5	3	5	5	−57	0	0	10	10	9	13	11	10	30	10	11
FEAL	7	6	6	7	6	5	0	0	−17	14	14	12	15	12	15	7	−14	25	31	29	32	28	28	26	−10	−3	−19
FFT4	4	4	3	4	3	3	0	−25	0	7	5	5	4	7	7	−43	40	40	9	11	11	10	9	6	11	−18	−46
FFT8	11	10	8	10	9	8	−9	−10	0	26	26	21	26	24	20	0	−8	−5	23	27	23	19	25	22	−17	−7	−4
EWF	7	6	5	6	5	5	−14	−17	0	17	15	14	19	16	13	12	7	−7	29	29	22	26	25	21	−10	−14	−5
EWF6	35	29	25	35	29	25	0	0	0	105	101	87	115	114	102	10	13	17	134	129	124	111	96	94	−17	−26	−24
FDCT	13	12	11	13	12	10	0	0	−9	29	28	24	30	26	24	3	−7	0	29	32	32	30	31	29	3	−3	−9
FDCT6	75	68	59	75	68	56	0	0	−5	189	175	153	163	168	155	−14	−4	1	157	170	168	192	170	144	22	0	−14
MATRIX4	38	33	32	37	33	32	−3	0	0	90	36	36	75	48	36	−17	33	0	77	97	97	80	98	109	4	1	12
DCT8	26	23	20	25	22	19	−4	−4	−5	75	58	44	60	47	47	−20	−19	7	48	56	65	64	67	61	33	20	−6
MEDIAN	7	7	5	7	7	5	0	0	0	16	16	12	11	14	12	−31	−13	0	12	12	14	17	11	13	42	−8	−7
BTREE32	11	11	8	11	11	8	0	0	0	29	29	15	10	28	16	−66	−3	7	12	12	17	21	13	17	75	8	0
平均 *Δ*%	—	—	—	—	—	—	−10	−14	−9	—	—	—	—	—	—	−20	2	11	—	—	—	—	—	—	14	−4	−10

表 8 A_{RPU}=56、64、75 时 AEMO 算法与 LSCBP 算法的划分结果

划分用例	*M*									*N*									*SD*								
	LSCBP			AEMO			*Δ*%			LSCBP			AEMO			*Δ*%			LSCBP			AEMO			*Δ*%		
SODE	6	5	5	4	4	4	−33	−20	−20	7	5	4	3	5	5	−57	0	25	10	12	13	13	11	10	30	−8	−23
FEAL	8	7	6	7	6	5	−13	−14	−17	15	11	13	15	12	15	0	9	15	30	29	26	28	28	26	−7	−3	0
FFT4	5	5	4	4	3	3	−20	−40	−25	6	5	5	4	7	7	−33	40	40	10	10	9	10	9	6	0	−10	−33
FFT8	12	11	10	10	9	8	−17	−18	−20	23	23	21	26	24	20	13	4	−5	24	26	26	19	25	22	−21	−4	−15
EWF	8	6	6	6	5	5	−25	−17	−17	20	20	17	19	16	13	−5	−20	−24	25	22	23	26	25	21	4	14	−9
EWF6	36	31	26	35	29	25	−3	−7	−4	118	113	106	115	114	102	−3	1	−4	139	130	128	111	96	94	−20	−26	−27
FDCT	15	13	11	13	12	10	−13	−8	−9	23	25	22	30	26	24	30	4	9	38	33	31	30	31	29	−21	−6	−7
FDCT6	78	69	58	75	68	56	−4	−1	−3	168	152	150	163	168	155	−3	11	3	195	190	169	192	170	144	−2	−11	−15
MATRIX4	43	35	34	37	33	32	−14	−6	−6	56	50	51	75	48	36	34	−4	−29	118	104	108	80	98	109	−32	−6	1
DCT8	27	23	21	25	22	19	−7	−4	−10	53	43	44	60	47	47	13	9	7	77	79	67	64	67	61	−17	−15	−9
MEDIAN	8	8	6	7	7	5	−13	−13	−17	12	12	11	11	14	12	−8	17	9	15	15	13	17	11	13	13	−27	0
BTREE32	12	12	9	11	11	8	−8	−8	−11	18	18	17	10	28	16	−44	56	−6	19	19	19	21	13	17	11	−32	−11
平均 *Δ*%	—	—	—	—	—	—	−14	−13	−13	—	—	—	—	—	—	−6	12	3	—	—	—	—	—	—	−6	−11	−15

从上述实验比较结果可以看出，AEMO 算法与 LBP、ESL、CBP、MOTP、LSCBP 等 5 种算法相比，AEMO 算法获得的 *M* 是最小的；除 LBP 算法之外，AEMO 算法的 *SD* 均值是最小的，但 *N* 值完全优于 LBP 算法。

采用文献[14]的 10 个随机图基准程序，在 A_{RPU} 分别为 56、64、75 的条件下，相比 ESL，AEMO 算法对 *M* 改进的相对平均值分别为–5.4%、–6.1%、–8.7%；对 *N* 改进的相对平均值分别为+6.7%、+6.8%、+3.0%；对 *SD* 改进的相对平均值分别为–4.6%、–9.3%、–9.3%；相比 LSCBP，AEMO 算法对 *M* 改进的相对平均值分别为–8.2%、–8.7%、–10.7%；对 *N* 改进的相对平均值分别为+4.5%、+4.7%、+3.2%；对 SD 改进的相对平均值分别为–4.9%、–6.3%、–8.65%。

6　结束语

本文提出了一个 AEMO 算法，该算法考察了待划分就绪运算节点的实际情况，实现动态调整节点的调度次序，而且综合考虑了 DFG 划分块数和执行总延迟、划分块之间非 I/O 边数等多个因素，实验结果表明，与 LBP、ELS、CBP、MOTP 和 LSCBP 等 5 种算法相比较，AEMO 算法具有最小的 *M* 值，即划分模块数少，这意味着配置次数就少，这是影响计算密集型任务关键循环加速的主要因素之一。AEMO 算法对 *N* 值做了一定程度的优化，比 LBP 算法少了很多，并且随着 A_{RPU} 面积的增大，相比 ELS、CBP、MOTP、LSCBP 等算法，AEMO 算法所获得的 *SD* 均值也是最小的，而且它还可以对 α、β、γ 进行动态调整，以获得单一目标的优化。

参考文献：

[1] CARDOSO J M P, DINII C D, *et al*. Compiling for reconfigurable computing: a survey[J]. ACM Computing Surveys, 2010, 42(4): 1301-1365.

[2] COMPTON K, HAUCK S. Reconfigurable computing: a survey of systems and software[J]. ACM Computing Surveys, 2002, 34(2): 171-210.

[3] DASU A, PANCHANATHAN S. Reconfigurable media processing[J]. Elsevier's Parallel Computing, Special Issue on Parallel Computing in Image and Video Processing, 2002, 28(7/8): 1111-1139.

[4] CAMPI F, TOMA M, LODI A, *et al*. A VLIW processor with reconfigurable instruction set for embedded applications[J]. IEEE Journal of Solid-State Circuits, 2003, 38(11): 1876-1886.

[5] 王大伟,窦勇,李思昆. 核心循环到粗粒度可重构体系结构的流水化映射[J]. 计算机学报, 2009, 32(6): 1089-1099.
WANG D W, DOU Y, LI S K.Loop kernel pipelining mapping onto coarse-grained reconfigurable architectures[J]. Chinese Journal of Computers, 2009, 32(6): 1089-1099.

[6] 于苏东,刘雷波,尹首一等.嵌入式粗粒度可重构处理器的软硬件协同设计流程[J].电子学报, 2009, 37(5): 1136-1140.
YU S D, LIU L B, YIN S Y, *et al*. Hardware-software co-design flow for embedded coarse-grained reconfigurable processor[J]. Acta Electronica Sinica, 2009, 37(5): 1136-1140.

[7] DIMITROULAKOS G, GEORGIOPOULOS S, GALANIS M D, *et al.* Resource aware mapping on coarse grained reconfigurable arrays[J]. Microprocessors and Microsystems, 2009, 33(2): 91-105.

[8] LEE G, CHOI K, DUTT N D. Mapping multi-domain applications onto coarse-grained reconfigurable architectures[J]. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, 2011, 30(5): 637-650.

[9] GPURNA K M, BHATIA D. Temporal partitioning and scheduling data flow graphs for reconfigurable computers[J]. IEEE Transactions on Computers, 1999, 48(6): 579-590.

[10] 周博, 邱卫东, 谌勇辉等. 基于簇的层次敏感的可重构系统任务划分算法[J]. 计算机辅助设计与图形学学报, 2006, 18 (5): 667-673.
ZHOU B, QIU W D, CHEN Y H , *et al.* A level sensitive cluster based partitioning algorithms for reconfigurable systems[J]. Journal of Computer Aided Design & Computer Graphics, 2006, 18(5): 667-673.

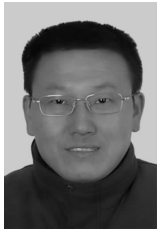
[11] JIANG Y C, WANG J F. Temporal partitioning data flow graphs for dynamically reconfigurable computing[J]. IEEE Transactions on Very Large Scale Integration Systems, 2007, 15(12): 1351-1361.

[12] 潘雪增, 孙康, 陆魁军等. 动态可重构系统任务时域划分算法[J]. 浙江大学学报(工学版), 2007, 41(11): 1839-1844.
PAN X Z, SUN K, LU K J, *et al.* Temporal task partitioning algorithm for dynamically reconfigurable systems[J]. Journal of Zhejiang University (Engineering Science), 2007, 41(11): 1839-1844.

[13] CARDOSO J M P, NETO H C. An enhanced static-list scheduling algorithm for temporal partitioning onto RPU[A]. Proceedings of 1999 IFIP International Conference on Very Large Scale Integration[C]. Lisbon, 1999. 485-496.

[14] 陈乃金,江建慧,陈昕等. 一种考虑执行延迟最小化和资源约束的改进层划分算法[J].电子学报, 2012, 40(5): 1055-1066.
CHEN N J, JIANG J H, CHEN X, *et al.* An improved level partitioning algorithm considering minimum execution delay and resource restraints[J]. Acta Electronica Sinica, 2012, 40(5): 1055-1066.

作者简介：



陈乃金（1972–），男，安徽合肥人，同济大学博士生，安徽工程大学副教授，主要研究方向为可重构计算、时域划分与映射等。

江建慧（1964–），男，浙江淳安人，博士，同济大学教授、博士生导师，主要研究方向为可信系统与网络、软件可靠性工程、VLSI/SoC 测试与容错等。