一种考虑执行延迟最小化和资源约束的改进层划分算法

陈乃金[1,2,3],江建慧[1],陈 昕[3],周 洲[1],徐 莹[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;3. 同济大学电子与信息工程学院,上海 201804)

摘 要: 本文提出了一种改进的层划分算法.该算法充分考虑了划分块的最小执行延迟和尽可能充分利用可重构资源,能够跟踪层划分算法节点分配过程并进行调整,消除了经典层划分算法不能动态更新就绪节点列表选取节点进行划分的缺陷.实验结果表明,与层划分算法相比,所提出的改进层划分算法在模块数、执行延迟和跨模块间的 I/O 边数等三个方面均获得了改进.与现有的簇划分、增强静态列表、多目标时域划分、簇层次敏感等四种划分算法相比,新算法能获得最少的执行延迟,并且随着可重构处理单元面积的增大,模块数的均值也是最小的.

关键词: 可重构计算; 时域划分; 层划分; 最小化执行延迟; 资源约束
中图分类号: **文献标识码:** A **文章编号:** 0372-2112 (2012) 05-1055-012
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2012.05.032

An Improved Level Partitioning Algorithm Considering Minimum Execution Delay and Resource Restraints

CHEN Nai-jin[1,2,3], JIANG Jian-hui[1], CHEN Xin[3], ZHOU Zhou[1], XU Ying[1]
(1. *School of Software Engineering, Tongji University, Shanghai* 201804, *China*;
2. *College of Computer and Information Engineering, Anhui Polytechnic University, Wuhu, Anhui* 241000, *China*;
3. *College of Electronics and Information Engineering, Tongji University, Shanghai* 201804, *China*)

Abstract: This paper presents an improved level based partitioning (ILBP) algorithm that considers sufficiently minimum execution delay and make good use of reconfigurable resources as soon as possible. ILBP can trace and adjust the node allocation process of level based partitioning (LBP) algorithm. ILBP can eliminate the disadvantage of LBP that can not update dynamically the ready list nodes for partitioning node selection. Experiment results show that the proposed ILBP algorithm can improve number of modules, execution delay and number of input-output edges crossing modules by comparing with LBP. By comparing with cluster based partitioning, enhanced static list, multi-objective temporal partitioning and level sensitive cluster based partitioning algorithms, the proposed algorithm can get the least execution delay, and it can also obtain the least average number of modules with increase of the area of reconfigurable processing unit.

Key words: reconfigurable computing; temporal partitioning; level-based partitioning; minimum execution delay; resource restraint

1 引言

可重构计算(reconfigurable computing)既具有专用集成电路(application specific integrated circuit, ASIC)的高计算速度及效率,又具有微处理器的可编程性和良好的灵活性[1].对于嵌入式系统等计算密集型任务方面的应用已经展示了巨大的优势[2].同时,可重构计算系统又不同于众核图形处理单元(graphic processing unit, GPU)[3],可重构计算结构实质上可以看成是由时间域和空间域构成的二维结构[4],其在时间域的灵活性是通过运算操作节点任务在可重构阵列上的映射和调度实现的;空间域上是通过重复使用可重构的运算单元阵列,通过配置功能块来实现的,它是一个流水线结构.

一个实现某个任务的计算密集型高级语言程序可以通过人工或自动的方法提取出它的数据流图(data flow graph, DFG)(以下简称任务 DFG),它是一个有向无环图(directed acyclic graph),程序如果要在粗粒度可重构计算机系统上运行,则要根据该系统中可重构单元阵列(reconfigurable cell array, RCA)的结构来选择相应的任务划分、映射算法[5].在动态可重构系统中,当因任务 DFG 过大而无法在硬件上一次性处理时,就必须对图进行划分.任务 DFG 的时域划分就是把一个任务 DFG 在时间上划分成互相关联的若干个子任务(或模块).图的分割问题是一个 NP 完全问题,故该类问题常常得到的

收稿日期:2011-04-05;修回日期:2012-02-02
基金项目:国家自然科学基金(No.60903033);国家 863 高技术研究发展计划(No.2009AA011705)

是较优解.传统的任务 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两种.其中行为级时域划分算法又可分为单目标优化、多目标优化等算法.单目标优化算法主要是基于列表调度,常用的有层划分(level-based partitioning,LBP)[6]、簇划分(cluster-based partitioning,CBP)[6]、簇的层次敏感划分(level sensitive cluster-based partitioning,LSCBP)[7]等.多目标优化算法有增强的静态列表(enhanced static list,ESL)算法[8]、多目标时域划分(multi-objective temporal partitioning,MOTP)算法[9]等.

LBP 算法试图通过尽可能早(as soon as possible,ASAP)的调度策略为任务 DFG 中的节点分层,然后在满足硬件提供的资源约束条件下,逐层将计算任务节点放入到相应的划分块中.该算法的优点是尽可能地使每个划分后模块的内部操作的并行度达到最大.然而,它存在以下不足:(1)算法挑选任务节点是按照先来先服务的原则一层一层机械地进行,不能根据实际的划分情况动态调整就绪列表队列,结果产生大量的硬件碎片;(2)在选取运算任务节点进行划分时,不能保证在获得较小执行总延迟的情况下,尽量减少划分块间的 I/O 边数;(3)将第 0 层的所有输入划分为第 0 块,从而增大了任务 DFG 的处理时间.

CBP 算法试图尽可能地将联系紧密型操作放到一个模块当中,以减少划分模块之间的通信代价.LSCBP 算法对 CBP 算法进行了改进,消除了 CBP 算法机械选取节点划分的缺陷,每次划分均充分利用了硬件碎片.缺点有:把一个任务 DFG 的第 0 层的输入划入了第 0 块;在追求划分块间的边数较小化的同时,由于又考虑了运算任务的并行性,因此导致了划分块间的边数仍然偏大;统计划分块间输出边数量的方法不太合理,即当一个运算节点的输出边数超过一时,若直接按实际边数进行统计,则有可能导致该节点的运算结果被多次存储.

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,每次在划分前计算任务 DFG 中每个节点的 ASAP、ALAP(as late as possible)及层次值等,然后用这些值和一个加权公式计算出每个节点的权值,选取权值高的节点,实现对一个任务 DFG 的优化划分.其不足是对并行因素考虑不够,表现为所有任务 DFG 块执行延迟总和较大,且没有考虑任务 DFG 划分中可能产生的硬件碎片问题.

MOTP 算法的优点是用加权的方法把通信量和模块执行时间转化为统一的代价函数,计算通信时间占总代价的比例 $\beta=(\alpha * comcost)/cost$,如果 $\beta\geqslant\gamma(0<\gamma\leqslant 1)$,就在绪表中选择能够减少通信量的节点,反之,选择能够提高并行度的节点[9].缺点一是在每次划分时,没有进行硬件资源面积的估算.在选择节点过程中,如果所选节点不能满足当前的资源限制,就分配新的模块,但是该节点之后如果还有满足要求的节点就有可能会产生硬件碎片;二是 MOTP 算法所用的运算节点所占资源大小和执行延迟全为简单假设;三是统计划分块间输出边的数量不太合理,理由同 LSCBP 算法.

本文在深入分析 LBP 算法的基础上,综合考虑了硬件资源利用率最大化、划分块间的存储成本、第 0 层输入划分的简化等三个方面的因素,提出了一种改进的层划分(improved level-based partitioning,ILBP)算法.该算法获得了更少的执行延迟,相对于 LBP 算法而言获得了较少的划分模块数,较少的块间 I/O 边.因为 ILBP 算法考虑了 LBP、LSCBP、ESL、MOTP 等算法所存在的共性问题,并进行了相应的改进,因此其结果较具优势.由于 LSCBP 算法是对 CBP 算法的改进,故未直接解释 ILBP 算法改进 CBP 算法的理由.

2 问题定义

一个任务 DFG 中的每个节点表示一个操作符(或运算符),对这样的图进行时域划分通常称为行为级时域划分.本文的研究基于如下 3 个条件:(1)待划分的计算密集型任务的关键循环已经转换为一个 DFG;(2)RCA 可以重复使用,一个划分可以按划分形状直接映射到一块 RCA 上,并且一个划分占用一块 RCA;(3)在计算资源成功映射的前提下,每个划分内所有的边都能在硬件上实现连接,并且划分算法只考虑硬件实现的计算延迟,而不考虑互连延迟.下面给出任务划分的相关概念.

定义 1 一个任务 DFG 可以表示为一个四元组 $G=(V,E,W,D)$,顶点集 $V=\{v_i\mid v_i$ 是有序运算符,$1\leqslant i\leqslant n\}$,$|V|=n$ 表示运算符的个数;边集 $E=\{e_{ij}\mid e_{ij}=<v_i,v_j>,1\leqslant i,j\leqslant n\}$,$e_{ij}$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,表示了 v_i 和 v_j 两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 $W=\{w_i\mid$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 个运算的延迟.

一般而言,组成 RCA 的可重构处理单元(reconfigurable processing unit,RPU)的面积为一个定值,表示为 A_{RPU}.对于任一个任务 DFG,在满足相关约束条件下,采用某种分割方法可以获得一个具有 M 个划分块的划分,表示为 $P=\{P_1,P_2,\cdots,P_M\}$,$|P|=M$,其中,第 i 个划分块 P_i 由任务 DFG 的若干个节点组成,$1\leqslant i\leqslant M$.P_i 和 P_j 要求具有严格的执行顺序,若 P_i 先执行,P_j 紧跟 P_i 后执行,表示为 $P_i\rightarrow P_j$,则称 P_i 为 P_j 前驱模块,P_j 为

P_i 的后继模块.

定义 2　待划分的任务 DFG 是一个有向无环图,如果存在一个划分 P,对于某两个划分块 P_i 和 P_j,执行顺序的要求是 $P_i \rightarrow P_j$,但 P_i 中的某个节点却是 P_j 中某个节点的后继,则称 P_i 和 P_j 产生了非法依赖关系.

如果两个划分块之间产生了非法依赖关系,那么会导致两个有前驱后继关系的划分块中的任务节点得不到正确执行.

例 1　图 **1**(a)是一个任务 DFG,$n = 6$. 为了简化,假设图中各个节点的权值相同,这样,我们可以将面积的单位等价于节点的个数. 设 $A_{\mathrm{RPU}} = 3$,这意味着图 1(a)中的任务 DFG 分割的限定条件之一是每个划分块最多包含三个节点,即 $\max(|P_i|) \leqslant 3, i \in [1, n]$. 图 1($b$)是采用 CBP 算法分割之后得到的一个划分:$P' = \{P_1, P_2\}$. 在图 1($b$)中,若两个划分的执行顺序为 $P_1 \rightarrow P_2$,则它是一个合理的划分,但如两个划分的执行顺序为 $P_2 \rightarrow P_1$,这将导致 P_2 得不到正确执行,因为 P_2 和 P_1 产生了非法依赖关系,故它就是一个不合理的划分.

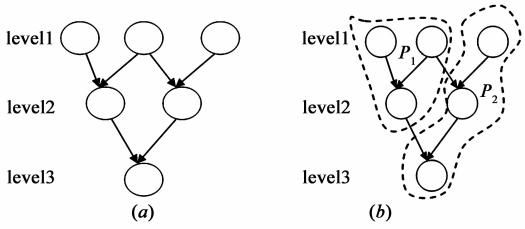


图1　解释任务DFG的合理划分和不合理划分的例子

定义 3　若一个任务 DFG 存在一个划分,对于划分块的一个执行顺序要求,它的所有划分块之间均不产生非法依赖关系,则该划分就是一个合理的划分.

定义 4　任务 DFG 执行的并行性是指按某种方法分割任务 DFG 得到的若干个子图,每个子图内部可以并行执行的运算节点的最大个数.

一个任务 DFG 的划分问题可以描述为:

输入:$G = (V, E, W, D)$.

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

约束条件:①$\bigcap_{i=1}^{M} P_i = \varnothing$;②$\bigcup_{i=1}^{M} P_i = V$;③$\forall P_i \in P, 1 \leqslant i \leqslant M$, $\sum_{j=1}^{n} w_j \leqslant A_{\mathrm{RPU}}$, w_j 为划分 P_i 中节点 j 的权值.④P 中所有的前驱和后继划分块之间不存在非法依赖关系;⑤RCA 是流水线结构,处理流程是导入数据→运算→导出数据,RCA 的内部节点的输出数据不能先存到外部存储器,再通过配置字把这个数据读入供当前块内下一节点使用.

目标:①执行延迟较短;②运行并行度较大;③划分块数较少.

3　ILBP 算法

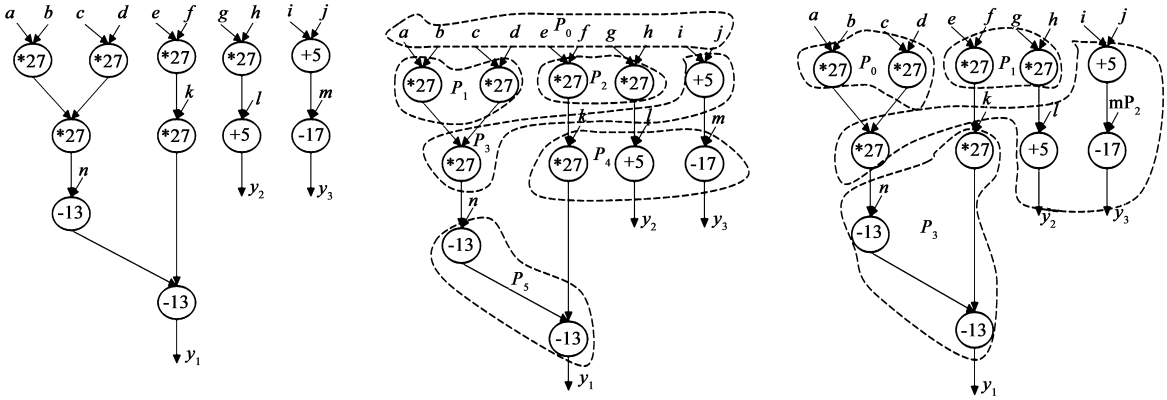
3.1　LBP 算法的不足

(1)不能根据实际划分情况调整就绪列表队列,会产生大量硬件碎片.

LBP 算法选取任务 DFG 中的节点采用的是一种按层从左到右,从上到下顺序选取的方法. 当遇到当前不能满足硬件资源约束的节点时,就重新分配划分块,即使当前节点的后续就绪节点满足要求也不能划入当前块,这就容易导致硬件资源的浪费,从而划分块数较多. 块数越多,配置次数就越多,从而导致关键循环执行时间增大.

(2)不能保证任务 DFG 在获得较小执行总延迟的情况下,使划分块间的 I/O 边数(该边是指连接不同划分块之间的非原始 I/O 边)较小化. 若划分块间的 I/O 边越多,则划分块间通信成本就越大.

例 2　对于如图 **2**(a)所示的二阶差分方程(SODE[10])的任务 DFG,有 11 个运算任务节点,其原始输入边有 14 条,原始输出边有 3 条,非原始输入输出边数为 8 条. 设 $A_{\mathrm{RPU}} = 55\mathrm{CLB}$,加法、减法、逻辑比较和乘法的执行延迟分别为 1、1、1 和 2 个时钟周期[11]. 采用 XC4000E 系列的 FPGA 实现,各种运算的位数为 8 位,相

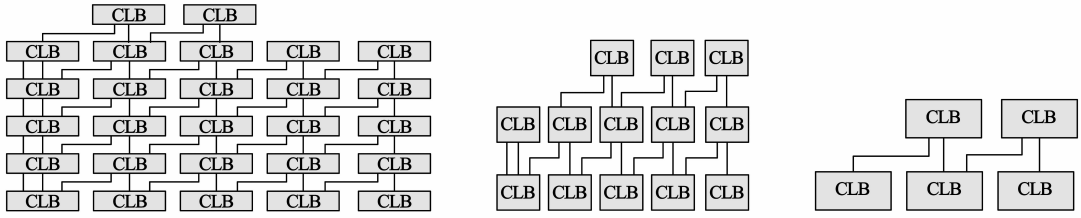


(a) SODE例子　(b) 基于LBP算法的划分图　(c) 基于ILBP算法的划分图

图2　SODE及其划分

应运算所需要的 CLB 个数分别为 5、13、17 和 27 个.图 3 给出了乘法、减法和加法三种 8 位运算的 CLB 结构.

采用 LBP 算法对 SODE 进行划分后的结果如图 2(*b*)所示,它有 6 个划分块 $P_0 \sim P_5$,划分块之间的 I/O 边数为 7 条.由此可知,LBP 算法虽然使划分块内部运算节点之间的并行度较高(表现为划分后所有模块执行延迟总和较小),但是,相对于节点之间的非原始输入输出边数,划分块之间的 I/O 边数还是较多.



(*a*) *w*(*mul*)=27 (*b*) *w*(*sub*)=13 (*c*) *w*(*add*)=5

图3 8位乘法、减法和加法的CLB结构

(3)对于第 0 层输入的处理不合理.

LBP 算法把任务 DFG 第 0 层的所有输入划为第 0 块,这样做会延长任务 DFG 的分割时间,而实际上这些输入却可由文件直接读取,没有必要将其作为一个独立的划分块,第 0 块没有运算,无需配置,但是运算阵列的空载运行仍然消耗了时间.

3.2 ILBP 算法设计

为了保证某个任务 DFG 获得较小执行总延迟的情况下,减少划分块之间的存储边数并充分利用资源,ILBP 算法在满足可重构资源面积约束的条件下,按层次遍历的方式,同时考虑执行延迟和硬件资源两个因素来对原 LBP 算法进行优化.在 ILBP 算法初始化过程中,将原始输入作为文件直接读取.

ILBP 算法采用以下策略来改进 LBP 算法.

策略 1 保证执行延迟最小化.

与 LBP 算法相同,ILBP 算法采取从左到右,从上到下顺序选取节点的方法对任务 DFG 按层划分,当遇到不满足要求的节点时跳过,继续搜索其后处于就绪状态的节点,而当搜索到满足要求的节点时,才考虑加入当前块.当前块加入节点的条件是加入节点后的块延迟不大于当前块延迟.这样做的主要目的是增大节点之间的并行度并最小化整个任务 DFG 的执行延迟.

策略 2 在保证当前划分块执行延迟最小化的情况下,可重构运算阵列面积利用最大化.

LBP 算法在划分完当前就绪节点的面积仍有剩余时就停止,但是该剩余面积还可能满足剩下的队首节点后面节点的面积要求,因此在保证当前划分块执行延迟最小化的条件下,可以在队首节点之后,按同时满足加入某节点后不增加当前划分块延迟和尽可能填满可重构运算阵列的原则,对就绪运算节点进行贪婪搜索.尽可能填满可重构运算阵列原则的含义是优先划入当前队首节点之后满足剩余硬件碎片且占用硬件资源大的节点,以提高系统资源的利用率.例如,在队首节点之后,如果第一个遇到的节点是加法,后面是乘法,那么就优先划入占用资源大的乘法运算.

策略 3 在保证当前划分块执行延迟最小化的情况下,尽量减少划分块之间的 I/O 边数.

策略 1 ~ 2 的目的是最小化任务 DAG 的执行延迟和减少硬件碎片,但 LBP 算法对划分块之间 I/O 边数多的因素考虑不足,ILBP 算法每次选定一节点进行划分后,均要及时更新其直接后继入度,当遇到不满足要求的节点时则跳过,在保证当前划分块执行延迟最小化的条件下,每次都要计算当前块的块间边数,如果加入点后块间边数小于等于当前块的块间边数,则加入该节点;否则不加入.而且,若有多个节点就绪,则优先选择当前划分块内的直接后继节点.这样做的目的是使联系紧密的节点尽可能地处于同一划分块当中.这样,就可以达到保持或减少划分块间 I/O 边数的目的.

设 M 表示一个 DFG 的划分块数,SD 表示一个 DFG 的所有划分块执行延迟总和,N 表示一个 DFG 的所有划分块之间的非原始 I/O 边数.基于上述三个策略的 ILBP 算法描述如下:

算法:ILBP

输入:一个计算密集型任务 DFG(以数据表形式表示)

输出:任务 DFG 的一个划分及其相应的 M、SD、N、所占的内存 *Mem*、运行时间 T

约束:A_{RPU};任务 DFG 所有划分块之间均不产生非法依赖关系

(1)读入一个表示计算密集型任务 DFG 的数据表;用函数 Init()求出每个节点入度和出度个数、前驱与后继列表;用函数 *assign_levels*(v_i)求出每个运算节点层次和整个任务 DFG 的最大层次;初始化各个变量,设定 A_{RPU}的值,每个节点的划分标志置 0,设置用于存储动态更新的节点的入度数组,*Area_Filled* = 0;

(2)while (节点没有被划分完) do

if (flags = 0) //划分遇到的节点满足面积要求,直接

进行划分
扫描数据表;
if(节点入度为 0&& 没有被划分)
if ((Area_Filled + node[vi].size) < = A_{RPU})
{n ++ ;node[vi].partition = i;节点的划分标志置 1;更新填充面积和后继节点的入度;}
else if((Area_Filled + node[vi].size) > A_{RPU}) {flags = 1;break;}
End if;
if(flags = 1)//划分遇到的节点不满足面积要求,跳过该点找下一个节点
InitQueue();扫描数据表;
if(节点入度为 0&& 没有被划分)
AddQueue(vi);queueSort()排序求出可划入当前块占用硬件资源最大的节点;
while(pQueue! = NULL)
vi = DelQueue();
if((Area_Filled + node[vi].size) < = A_{RPU})
{分别计算当前块和加入新节点后延迟和边数;node[vi].partition = i;节点划分标志置 1;}
if (加入新节点后的延迟和边数均小于等于加入新节点前的延迟和边数)
{则划入该点,多个节点就绪,优先选择当前划分块内节点的直接后继;n ++ ;
更新填充面积和后继节点的入度;}
else if(加入新节点后,不满足要求)
{把已经划分的节点恢复划分前状态,
continue;}
End while;
End if;
End while;求出划分块数;
(3)开辟新块;填充面积变量置 0,为下次划分做准备;
(4)用 edges()和 delays()求出任务 DFG 划分块间的边数和划分块执行总延迟;同时求出该任务 DFG 划分所消耗的内存 *Mem* 和运行时间 *T*;
(5) End ILBP.

3.3 算法时间复杂度分析

已知一个有 n 个运算节点的任务 DFG、每个运算节点的执行延迟、运算类型和占用的资源数,每个节点入、出度,前驱与后继的列表由初始化函数 Init()通过任务节点的二维关系矩阵求得,该运算的时间复杂度为 $O(n^2)$;分层函数 assign_level()通过二维关系矩阵求得每个运算节点层次及最大层次的时间复杂度为 $O(n^2)$;edges()通过扫描 n 个运算节点及其后继列表来求出一个划分后的任务 DFG 块之间总的 I/O 边数,其时间复杂度为 $O(n^2)$;假设一个任务 DFG 被划分为 M 块,用直接递归调用来求一个划分后的任务 DFG 所有块执行总延迟函数 delays()的时间复杂度为 $O(nM)$.

遍历所有节点,当遇到满足要求的节点时,就加入当前块,同时更新划入当前块的后继节点的入度,这部分时间复杂度为 $|n|$;当遇到不满足要求的节点时,考虑到硬件碎片的利用,ILBP 跳过该点继续向后搜索满足面积要求的节点.当找到一个节点之后,先计算当前块的总延迟和块间边数,再计算加入该节点后的总延迟和块间边数.若加入新节点后的划分块的总延迟和划分块之间的边数均小于或等于加入节点之前划分块的总延迟和划分块之间的边数,则加入该节点,否则不加入.另外,ILBP 算法中当节点未被划分完全时,要重新排序,每次通过冒泡排序求出可以划入当前块占用硬件资源最大的节点,该处理过程的平均时间复杂度为 $O(n^2)$.综上,ILBP 算法的时间复杂度约为 $O(n^3)$.

例 3　继续采用图 2(a)中的例子,在保持延迟最小化前提下,采用 ILBP 算法的划分图如图 2(c)所示.图 2(b)和(c)所示的 SODE 采用两种不同划分算法的划分结果如表 1 所示.显然,ILBP 算法有一定的改进效果.

表 1　SODE 的任务 DFG 的两种划分结果比较

参数 \ 划分算法	LBP	ILBP
M	6	4
SD	10	9
N	7	5

一个任务 DFG 在一个可重构计算平台上执行所需的时间大致包括以下四个部分:划分块的配置时间、所有划分块执行延迟总和、数据输入时间、数据输出时间.以文[14]所给平台为例,它有负责配置硬件的功能和连接关系,有控制寄存器 17 个,运算阵列的空载运行为 6cycles 且可以完全被使用,每种运算和每个控制寄存器的配置耗时为 1cycle,计算平台上执行一次所用的数据输入或输出所需的周期数是该划分块所有数据输入或输出个数的一半(只考虑划分块之间的非原始 I/O 边数的输入与输出次数).这样,对于图 2 中的例子,LBP 算法所需的时钟周期为 119 cycles,而 ILBP 算法为 93 cycles.

4 实验及其分析

4.1 实验的设计

4.1.1 划分基准程序的设计

为了对不同划分算法进行较为全面的比较,我们综合考虑了 *M*、*N*、*SD* 等指标,设计并实现了一组用于评价划分算法性能的划分基准程序,它们由两部分构成,一是由 LBP 和 CBP 算法所用的基准 MEDIAN(中值滤波器)、BTREE32(二叉树比较器)、DCT8(一维离散余弦变换)和 MATRIX4(4 × 4 矩阵运算),MOTP 算法所用

的基准 SODE、FEAL、EWF、EWF6、FDCT 和 FDCT6,此外还增加了 FFT4(4 次展开 FFT)、FFT8(8 次展开 FFT)两个基准,这些划分基准程序如表 2 所列.为了便于比较,我们用重构硬件资源面积作为约束条件.各类运算的时延和占用资源量参照 XC4000E 系列 8 位 FPGA 来确定,如表 3 所列.

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	–	–	–	–
MEDIAN	19	–	–	–	–	19	–	–
BTREE32	31	–	–	–	–	31	–	–
DCT8	90	40	16	34	–	–	–	–
MATRIX4	112	48	–	64	–	–	–	–

表 3 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

第二部分基准程序由 10 个随机图组成,如表 4 所列.ESL 算法使用了 100 个随机图,其中每个图含有 50 个节点,每个节点有 0 ~ 4 或 10 条输出边;LSCBP 算法也使用了随机图作为基准程序,每个图中的节点数不超过 250,每个节点入(出)度的概率分布一致,且最大入(出)度数为 3,层次为 4 ~ 10 层,面积向量 S 用正态概率分布 $Rand(\bar{s},\delta)$ 生成,其中 $\bar{s}$ 表示平均面积,δ 表示方差.实验参数的设置与文献[7]和[8]相同,共生成了 10 个随机图,其中,Random-graph1 ~ Random-graph8 对应于文献[7]的 8 种情况,其节点入出度的概率 $Prob$、$(\bar{s},\delta)$及其相关说明如表 4 所列,Random-graph 9 ~ Random-graph10 对应于文献[8]的两种情况.考虑实际不同位数运算的硬件开销往往处于某个范围内[12],因此,参照表 3,分别以加法、减法、逻辑比较、乘法和取模等五种运算的硬件开销作为面积区间的上限,用文献[7]的方法随机生成 10 个图的 S,然后将其依次分配给上述随机图的节点作为其面积(第 i 个节点所占的面积为 s_i),且 $0 < A_{RPU} \leqslant 50$.同时按如下方法近似指定各随机节点的运算类型和执行延迟($delay$):(a) $s_i \in [1,5]$,假设为加法,$delay = 1$;(b) $s_i \in [6,13]$,假设为减法,$delay = 1$;(c) $s_i \in [14,17]$,假设为逻辑比较,$delay = 1$;(d) $s_i \in [18,27]$,假设为乘法,$delay = 2$;(e) $s_i \in [28,50]$,假设为取模,$delay = 4$.

4.1.2 其它实验参数的设置

我们用 C 语言实现了 ILBP、LBP、CBP、LSCBP、ESL、MOTP 等六种时域划分算法,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.

LBP 和 CBP、LSCBP 算法模块数的计数均包括第 0 层输入的划分,而 ILBP、ESL、MOTP 算法不包括第 0 层输入的划分.六个算法在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一,也只算一条边,其理由是避免划分块间同一个节点的运算结果被多次存储.ESL 和 MOTP 算法相关参数的取值与原文相同,如在 MOTP 算法中,$\alpha = \gamma = 0.6$;ESL 算法中的通信成本的权值 $\alpha = 1$,关键路径的权值 $\beta = 1$,通信成本和关键路径折中权值 $\eta = \beta/(\alpha + 1)$.

表 4 随机图划分基准集及其特征

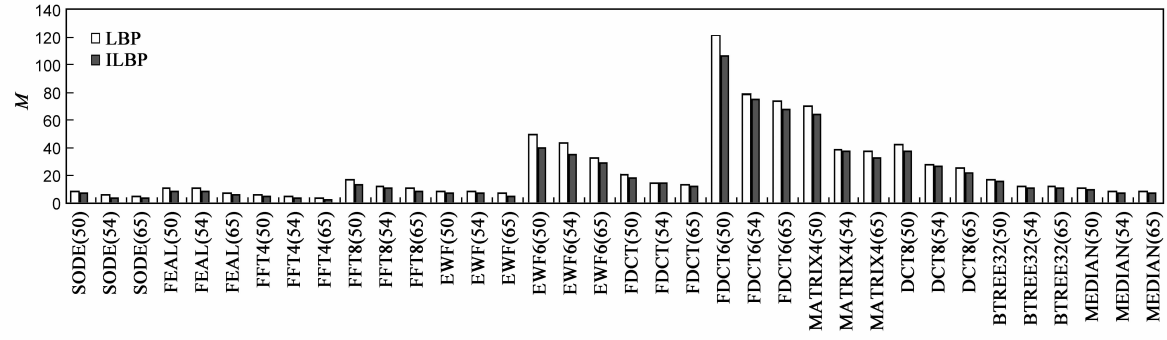
划分随机图用例	Prob	$(\bar{s},\delta)$	说明(其中[0,4]和[0,10]为每个节点输出边的范围)	节点总数	随机节点的运算类型及其所占的硬件面积的范围				
					加法 [1,5]	减法 [6,13]	逻辑比较 [14,17]	乘法 [18,27]	取模 [28,50]
Random-graph1	(0.2,0.2,0.3,0.3)	(15,7)	细粒度	20	1	5	3	11	–
Random-graph2	(0.2,0.2,0.3,0.3)	(30,15)	中粒度	40	–	1	3	9	27
Random-graph3	(0.2,0.2,0.3,0.3)	(50,20)	粗粒度	60	–	–	3	2	55
Random-graph4	(0.2,0.2,0.3,0.3)	(40,10)	小方差	80	–	–	–	5	75
Random-graph5	(0.2,0.2,0.3,0.3)	(40,20)	中等方差	100	2	7	2	12	77
Random-graph6	(0.2,0.2,0.3,0.3)	(40,30)	大方差	120	14	12	3	8	83
Random-graph7	(0.1,0.1,0.4,0.4)	(30,15)	大节点度	120	4	12	10	22	72
Random-graph8	(0.2,0.4,0.2,0.2)	(30,15)	小节点度	150	4	17	14	29	86
Random-graph9	—	(50,20)	小节点度,[0,4]	50	3	11	4	12	20
Random-graph10	—	(50,20)	大节点度,[0,10]	50	5	15	6	16	8

A_{RPU}随机选取 50CLB、54CLB 和 65CLB 三个值,同时还统计了不同划分算法对不同划分基准所占的内存空间 Mem(单位:MB)和运行时间 T(单位:秒).对于不同的 A_{RPU}值,采用划分基准程序集对其进行了验证,结果表明,自动划分的结果与手工划分的结果是吻合的.

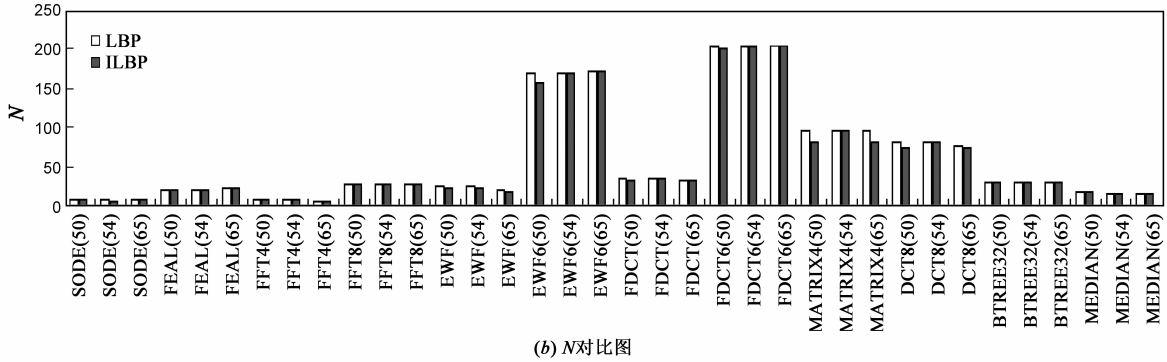
4.2 ILBP 算法与其它算法的比较

4.2.1 ILBP 算法与 LBP 算法的比较

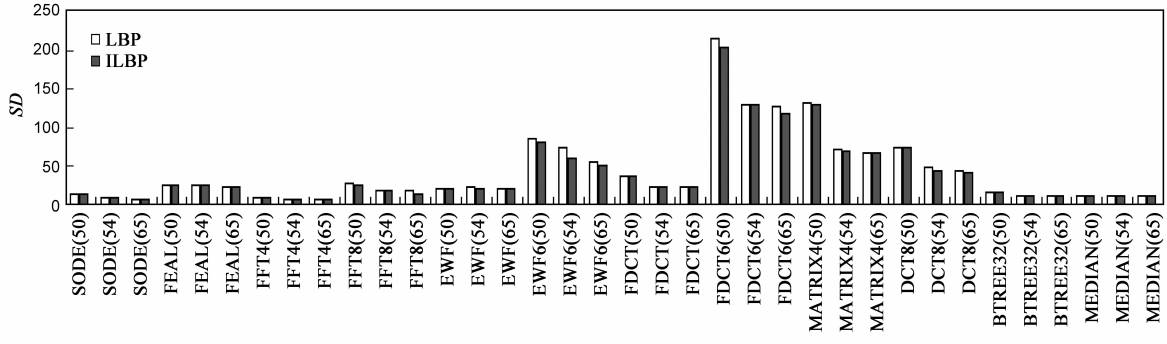
当采用表 2 中的 12 个基准程序时,ILBP 算法和 LBP 算法对比的实验数据见图 4.相对于 LBP 算法,ILBP 算法的 M、N 和 SD 均有了一定的改进.



(*a*) *M*对比图



(*b*) *N*对比图



(*c*) *SD*对比图

图4　LBP算法和ILBP算法对比图

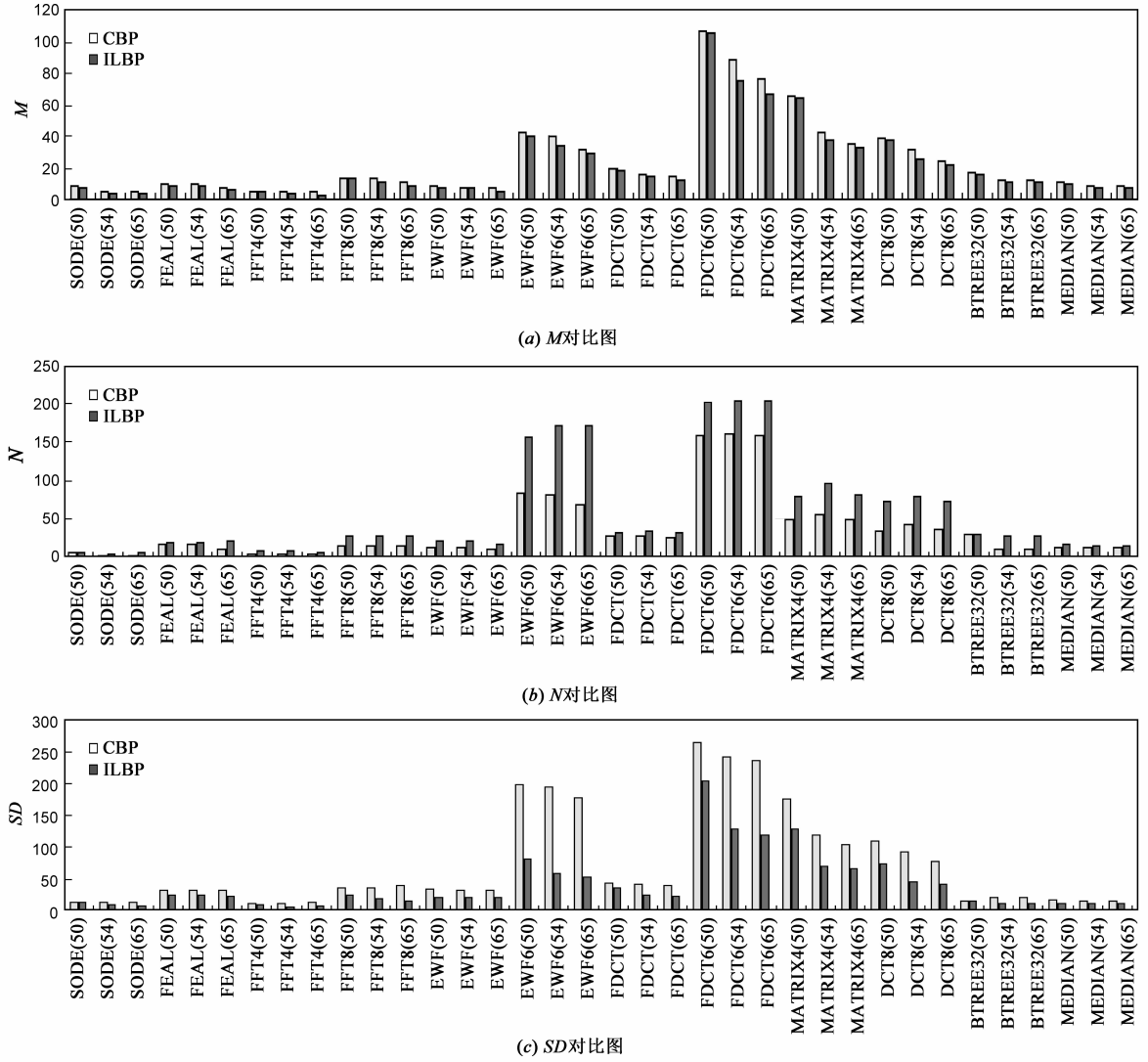
当 $A_{RPU}=50$ 时的对比数据如表 5 所示,表中的 Δ% 表示增量百分比.因篇幅所限,当 A_{RPU}为 54 或 65 时,仅给出结论性的数据(同样的原因,ILBP 算法与其它算法比较时,也采用类似的做法).对 M 改进的相对平均值分别为 -13.8% 或 -14.7%,最大值分别为 -33.3%(SODE)或 -28.6%(EWF),最小值分别为 -5.1%(FDCT6)或 -7.7%(FDCT);对 N 改进的相对平均值分别为 -18.5% 或 -9.5%,最大值分别为 -28.6%(SODE)或 -14.6%(MATRIX4),最小值分别为 -8.3%(EWF)或 -3.9%(DCT8);对 SD 改进的相对平均值分别为 -8.6% 或 -8.0%,最大值分别为 -20.0%(EWF6)或 -16.7%(FFT8),最小值分别为 -2.8%(MATRIX4)或 -2.9%(MATRIX4);对 T 改进的相对平均值分别为 -5.1% 或 -6.8%,最大值分别为 -16.1%(FEAL)或 -14.8%(EWF),最小值分别为 -0.1%(FDCT6)或 -0.1%(BTREE32);对 Mem 改进的相对平均值均为 -0.8%,最大值均为 -2.7%(FDCT6),最小值均为 -0.4%(EWF6).

表 5 $A_{RPU}=50$ 时 ILBP 算法与 LBP 算法的划分结果

划分用例	*M*			*N*			*SD*			*Mem*			*T*		
	LBP	ILBP	Δ%	LBP	ILBP	Δ%	LBP	ILBP	Δ%	LBP	ILBP	Δ%	LBP	ILBP	Δ%
SODE	8	7	−12.5	7	7	0.0	14	13	−7.1	0.85	0.84	−1.2	0.0028	0.0027	−3.6
FEAL	11	9	−18.2	20	20	0.0	26	25	−3.8	0.85	0.84	−1.2	0.0077	0.0060	−22.1
FFT4	6	5	−16.7	8	8	0.0	9	9	0.0	0.85	0.84	−1.2	0.0028	0.0028	0.0
FFT8	17	13	−23.5	28	28	0.0	28	25	−10.7	0.85	0.84	−1.2	0.0066	0.0063	−4.5
EWF	9	7	−22.2	24	21	−12.5	21	21	0.0	0.85	0.84	−1.2	0.0066	0.0057	−13.6
EWF6	49	40	−18.4	170	157	−7.6	86	82	−4.7	1.05	1.03	−1.9	0.3570	0.3565	−0.1
FDCT	21	18	−14.3	34	32	−5.9	36	36	0.0	0.85	0.84	−1.2	0.0066	0.0057	−13.6
FDCT6	121	106	−12.4	204	202	−1.0	216	203	−6.0	1.16	1.14	−1.7	0.4623	0.4590	−0.7
MATRIX4	70	64	−8.6	96	80	−16.7	133	129	−3.0	0.9063	0.9063	0.0	0.0809	0.0754	−6.8
DCT8	42	38	−9.5	82	74	−9.8	75	74	−1.3	0.8750	0.8750	0.0	0.0413	0.0407	−1.5
BTREE32	17	16	−5.9	30	30	0.0	16	16	0.0	0.8398	0.8320	−0.9	0.0074	0.0073	−1.4
MEDLAN	11	10	−9.1	17	17	0.0	11	11	0.0	0.8398	0.8320	−0.9	0.0033	0.0030	−9.1
平均 Δ%	——	——	−8.3	——	——	−8.9	——	——	−5.2	——	——	−1.1	——	——	−7.0

4.2.2 ILBP 算法与 CBP 算法的比较

当采用表 2 中的 12 个基准程序时，ILBP 算法与 CBP 算法比较的实验数据见图 5. 由于 CBP 算法尽可能地将联系紧密的运算节点放到一个模块中，因此 *N* 较小，但当它遇到当前不能满足硬件资源约束的节点时，就产生新的划分块，故 *M* 较大，*SD* 也较大.



(*a*) *M*对比图

(*b*) *N*对比图

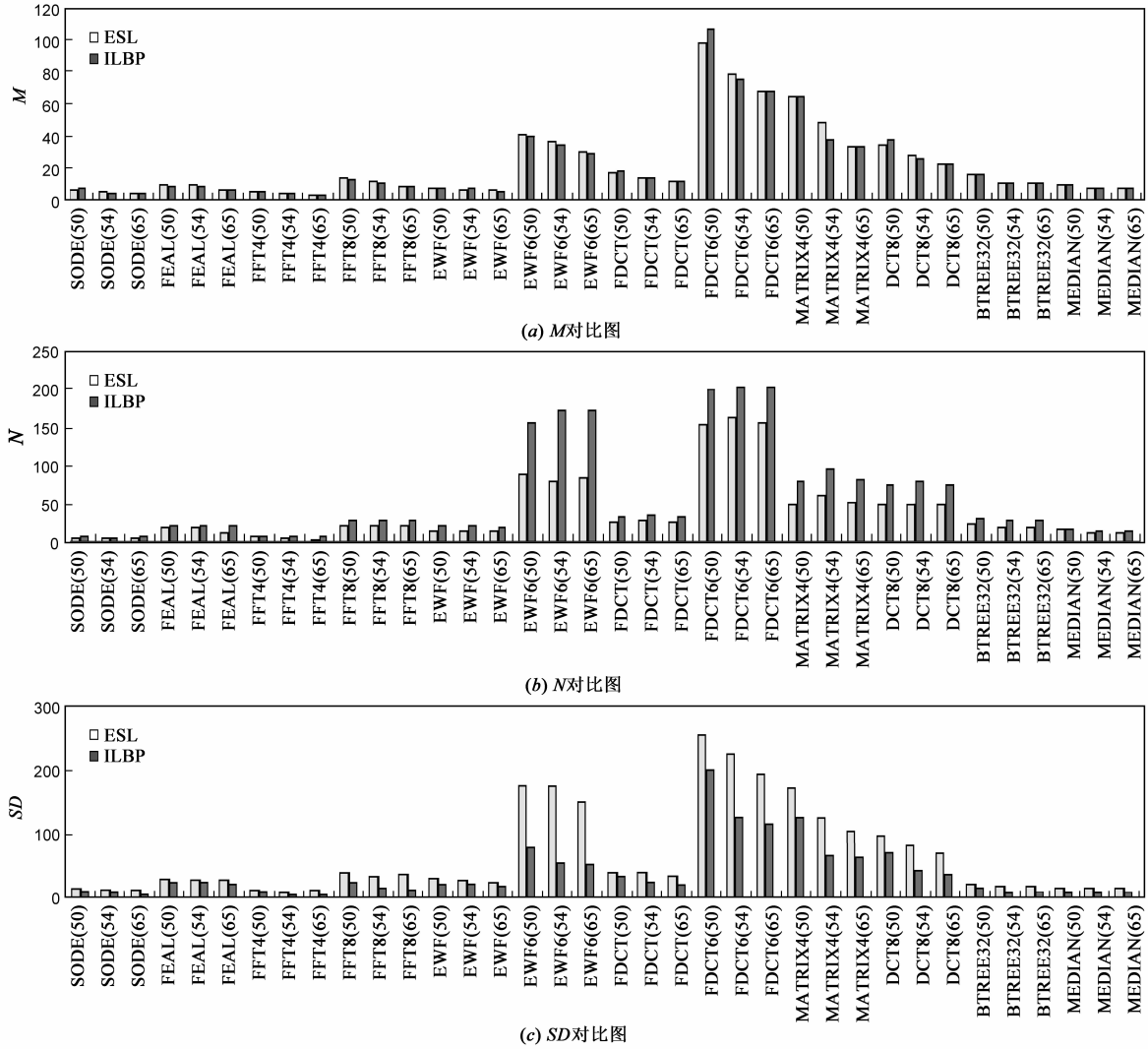
(*c*) *SD*对比图

图5 CBP算法和ILBP算法对比图

以 $A_{RPU}=50$ 为例,ILBP 算法对 M 改进的相对平均值为 -6.5%,最大值为 -12.5%(SODE),最小值为 -0.9%(FDCT6);对 N 改进的相对平均值为 +53.4%,最大值为 +111.4%(DCT8),最小值为 +16.7%(SODE);对 SD 改进的相对平均值为 -29.0%,最大值为 -58.8%(EWF6),最小值为 -7.1%(SODE).对 T 改进的相对平均值为 -5.5%,最大值为 -13.6%(FDCT),最小值为 -0.1%(EWF6);对 Mem 改进的相对平均值为 -0.9%,最大值为 -2.7%(FDCT6),最小值为 -0.4%(EWF6).

4.2.3 ILBP 算法与 ESL 算法的比较

当采用表 2 中的 12 个基准程序时,ILBP 算法与 ESL 算法比较的实验数据见图 6.以 $A_{RPU}=50$ 为例,ILBP 算法对 M 改进的相对平均值为 -0.01%,最大值为 -12.5%(EWF),最小值为 +16.7%(SODE);对 N 改进的相对平均值为 +41.8%,最大值为 +78.4%(EWF6),最小值为 +5.3%(FEAL);对 SD 改进的相对平均值为 -27.8%,最大值为 -53.4%(EWF6),最小值为 -13.3%(SODE);对 T 改进的相对平均值为 -10.2%,最大值为 -33.7%(FDCT),最小值为 -0.9%(EWF6);对 Mem 改进的相对平均值均为 -1.8%,最大值为 -2.9%(EWF6),最小值为 -0.9%(BTREE32).当 A_{RPU} 增大时,ILBP 算法对 M 改进的相对平均值逐步增大,分别为 -0.01%、-6.5%、-7.2%.



(*a*) *M*对比图

(*b*) *N*对比图

(*c*) *SD*对比图

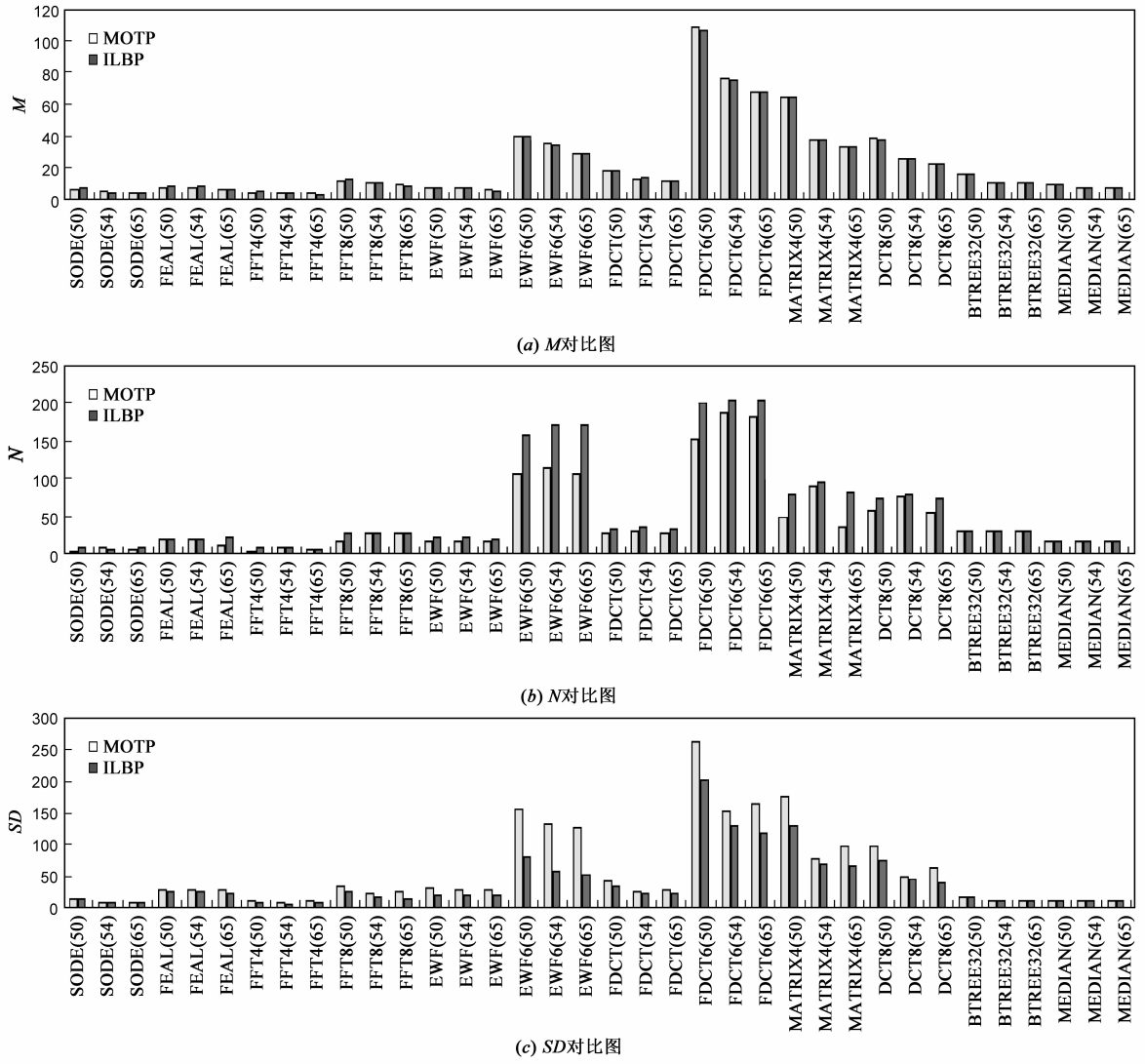
图6 ESL算法和ILBP算法对比图

当采用表 4 中的 10 个随机图基准程序时,在 A_{RPU} 分别为 50、54 和 65 的条件下,ILBP 算法对 M 改进的相对平均值分别为 -2.3%,-2.3%,-4.7%;对 N 改进的相对平均值分别为 +11.4%,+10.7%,+12.4%;对 SD 改进的相对平均值分别为 -8.4%,-7.9%,-16.5%;对 T 改进的相对平均值分别为 -3.5%,-3.7%,-2.5%;但 Mem 没有改进.

4.2.4 ILBP 算法与 MOTP 算法的比较

当采用表 2 中的 12 个基准程序时,ILBP 算法与 MOTP 算法比较的实验数据见图 7.以 $A_{RPU}=50$ 为例,

ILBP 算法对 M 改进的相对平均值为 +9.5%,最大值为 +25.0%(FDCT6),最小值为 -1.9%(FFT4);对 N 改进的相对平均值为 +48.8%,最大值为 +100.0%(FFT4),最小值为 +5.3%(FEAL);对 SD 改进的相对平均值分别为 -25.4%,最大值为 -47.4%(EWF6),最小值为 -13.3%(SODE);对 T 改进的相对平均值分别为 -13.0%,最大值为 -26.9%(FDCT),最小值为 -1.1%(EWF6);对 Mem 改进的相对平均值均为 -0.8%,最大值为 -2.7%(FDTC6),最小值为 -0.4%(EWF6).注意到当 A_{RPU}增大时,ILBP 算法的 M 均值将小于 MOTP 算法(实验数据除 $A_{RPU}=50$ 之外),当 $A_{RPU}=54$ 或 65 时,对 M 改进的相对百分比平均值分别为 -1.1% 或 -13.3%.



(*a*) *M*对比图

(*b*) *N*对比图

(*c*) *SD*对比图

图7 MOTP算法和ILBP算法对比图

4.2.5 ILBP 算法与 LSCBP 算法的比较

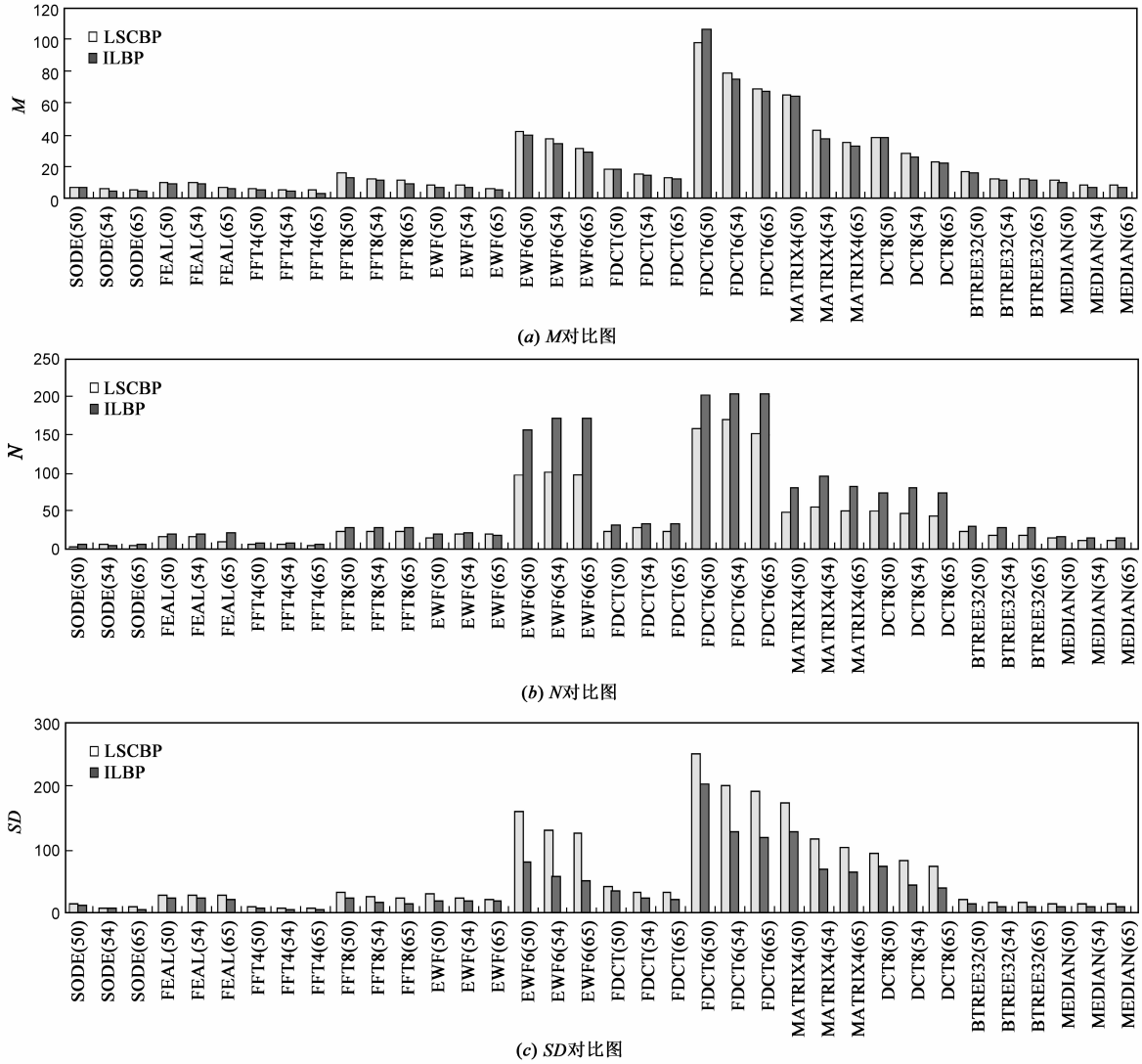
当采用表 2 中的 12 个基准程序时,ILBP 算法与 LSCBP 算法比较的实验数据见图 8.ILBP 算法获得了更小的 SD,并且随着 A_{RPU}的增大,除 FDCT6(50)之外,M 均值也均优于 LSCBP 算法.以 $A_{RPU}=50$ 为例,ILBP 算法对 M 改进的相对平均值分别为 -7.9%,最大值为 -18.8%(FFT8),最小值为 -4.8%(EWF6);对 N 改进的相对平均值为 +38.2%,最大值为 +75.0%(SODE),最小值为 +13.3%(MEDIAN);对 SD 改进的相对平均值为 -25.6%,最大值为 -48.8%(EWF6),最小值为 -13.3%(SODE);对 T 改进的相对平均值为 -10.9%,最大值为 -37.5%(FEAL),最小值为 -0.3%(FDCT6);对 Mem 改进的相对平均值为 -1.2%,最大值为 -3.4%(FDTC6),最小值为 -0.7%(EWF6).

当采用表 4 中的 10 个随机图基准程序时,在 A_{RPU} 分别为 50、54 和 65 的条件下,ILBP 算法对 M 改进的相对平均值分别为 -5.5%, -5.7%, -8.4%;对 N 改进的相对平均值分别为 +12.0%, +12.0%, +14.2%;对

SD改进的相对平均值分别为 -10.3%, -9.8%, -17.0%;对 T 改进的相对平均值分别为 -4.9%, -5.8%, -6.3%;对 Mem 改进的相对平均值均为 -0.9%.

4.3 比较的结果

从上述实验比较结果可以看出,ILBP算法相比于LBP算法而言,M、N 和 SD 均获得了全面的改进.与ESL、CBP、MOTP、LSCBP等四种算法相比,ILBP算法的划分模块执行总延迟是最小的.ILBP算法的 M 值优于CBP算法,且其均值也是优于ESL、LSCBP等算法的;随着 A_{RPU}的增大,M 均值将小于MOTP算法.然而,ILBP算法的 N 值比ESL、CBP、MOTP、LSCBP算法都大.



(*a*) *M*对比图

(*b*) *N*对比图

(*c*) *SD*对比图

图8　LSCBP算法和ILBP算法对比图

5 结论

国际上最新与时域划分相关的工作更多的是跟具体的粗粒度可重构硬件结构相结合来开展的.比如,基于单元共享的划分算法要求可重构硬件中的配置寄存器必须支持相同的运算单元,配置信息可以共享,以节省配置时间[11].基于相似率计算的划分算法是以划分块可以继承其上一个相邻划分块相似节点对的配置信息为前提的,相邻块的相似对节点不需要重新配置,从而减少了重构时间[13].

本文针对LBP算法划分结果的三种不合理情况提出了一种新的ILBP算法,它不依赖于具体的粗粒度可重构硬件结构,该算法考察了待划分就绪运算节点的实际情况,以及任务DFG划分块数和执行延迟最小化等因素,动态地调整节点的调度次序.新算法与LBP、ELS、CBP、LSCBP和MOTP等五种算法进行了比较,结果表明,ILBP算法全面优化了LBP算法的结果,使一个任务DFG的所有划分块的执行总延迟达到最小,并且相对于CBP、LSCBP、MOTP、ESL等算法而言,随着可重构处理单元面积的增大,模块数的均值也是最小的.我们

下一步的工作重点是在进一步优化 M 和 SD 的同时减小 N 值.

参考文献

[1] Estrin G, Bussel B, Turn R et al. Parallel processing in a restructurable computer system [J]. IEEE Transactions on Electronic Computers, 1963, 12(6): 747 – 755.

[2] 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.

[3] Fatahalian K, Houston M . GPUs: A closer look [J]. ACM Queue, 2008, 6(2): 18 – 28.

[4] João M. P. Cardoso, Pedro C. Diniz, Markus Weinhardt. Compiling for reconfigurable computing: A survey [J]. ACM Computing Surveys, 2010. 42(4): 1301 – 1365.

[5] 于苏东,刘雷波,尹首一,魏少军.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 Sudong, Liu Leibo, Yin Shouyi, Wei Shaojun. Hardware-software co-design flow for embedded coarse-grained reconfigurable processor [J]. Acta Electronica Sinica, 2009, 37(5): 1136 – 1140. (in Chinese)

[6] Karthikeya M, Purna G and Bhatia D. Temporal partitioning and scheduling data flow graphs for reconfigurable computers [J]. IEEE Transactions on Computers, 1999, 48(6): 579 – 590.

[7] 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 Bo, Qiu Weidong, ChenYonghui, 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. (in Chinese)

[8] João M P Cardoso, Neto H. An enhanced static-list scheduling algorithm for temporal partitioning onto RPUs [C]. In: Proceedings of 1999 IFIP International Conference on Very Large Scale Integration, Lisbon, IEEE CS Press, December 1999. 485 – 496.

[9] 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 Xuezeng, Sun Kang, Lu Kuijun et al. Temporal task partitioning algorithm for dynamically reconfigurable systems [J]. Journal of Zhejiang University (Engineering Science), 2007, 41(11): 1839 – 1844. (in Chinese)

[10] Mahmoud M, Masato M. A combined approach to high-level synthesis for dynamically reconfigurable systems [J]. IEEE Transactions on Computers, 2004, 53(12): 1508 – 1522.

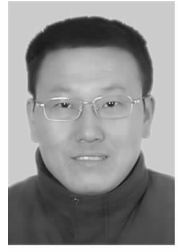
[11] João M P Cardoso. On combining temporal partitioning and sharing of functional units in compilation for reconfigurable architectures [J]. IEEE Transactions on Computers, 2003. 52(10): 1362 – 1375.

[12] Yung-Chuan Jiang and Jhing-Fa Wang. Temporal partitioning data flow graphs for dynamically reconfigurable computing [J]. IEEE Transactions on Very Large Scale Integration Systems, 2007, 15(12): 1351 – 1361.

[13] Mehdipour F, Zamani M and Sedighi M. An integrated temporal partitioning and physical design framework for static compilation of reconfigurable computing systems [J]. Microprocessors and Microsystems, 2006, 30(1): 52 – 62.

[14] 谢厉.粗粒度可重构计算系统中算法映射的研究与设计[D].上海:上海交通大学,2011.
Xie Li. Study and Design of Algorithm mapping in Grain-coarse Reconfigurable Computing System [D]. Shanghai: Shanghai Jiao Tong University, 2011. (in Chinese)

作者简介



陈乃金 男,1972 年生于安徽合肥.博士研究生,研究方向为可重构计算与时域划分等.
E-mail: 86naijinchen@tongji.edu.cn

江建慧(通讯作者) 男,1964 年生于浙江淳安.博士,教授,博士生导师,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.
E-mail: jhjiang@tongji.edu.cn