软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2022,33(9):3236−3248 [doi: 10.13328/j.cnki.jos.006388]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

硬件加速功能验证问题的 DAG 划分算法*

何天祥[1], 肖 正[1,2], 陈 岑[1,2], 刘楚波[1,2], 李肯立[1,2]

[1](湖南大学 信息科学与工程学院, 湖南 长沙 410082)

[2](国家超级计算长沙中心, 湖南 长沙 410082)

通信作者: 李肯立, E-mail: lkl@hnu.edu.cn

摘 要: 功能验证是超大规模集成电路 (very large scale integration, VLSI) 设计的一个基本环节. 随着超大规模电路的普及与发展, 在单处理器上对整个电路进行功能验证在可行性和效率上都存在较大的缺陷. 基于硬件加速器的功能验证是将整个电路划分成若干个规模更小的子电路; 然后在多个硬件处理器上并行的执行功能验证. 当电路划分结果的并行性较优时可提高功能验证的效率, 缩短时间周期. 类似电路设计中的其他划分问题, 用于硬件加速功能验证的电路划分问题可以被抽象成图划分问题. 相较于传统图划分问题, 硬件加速功能验证的划分问题还需要保证较小的模拟深度和较高的调度并行性. 为了满足硬件加速功能验证的划分需求, 提出了一种基于传统多级图划分策略的有效算法. 该算法结合调度思想, 利用电路的关键路径信息和时序信息, 将硬件加速功能验证问题转化为有向无环图的多级划分问题. 随机电路网表数据的实验结果表明, 所构造的算法可以有效的减少关键路径长度并且不会引起切边数的增长恶化.

关键词: 超大规模集成电路 (VLSI); 硬件功能加速验证; 有向无环图; 多级图划分; 关键路径

中图法分类号: TP301

中文引用格式: 何天祥, 肖正, 陈岑, 刘楚波, 李肯立. 硬件加速功能验证问题的DAG划分算法. 软件学报, 2022, 33(9): 3236–3248. http://www.jos.org.cn/1000-9825/6388.htm

英文引用格式: He TX, Xiao Z, Chen C, Liu CB, LI KL. DAG Partition Algorithm for Hardware Accelerated Function Verification. Ruan Jian Xue Bao/Journal of Software, 2022, 33(9): 3236–3248 (in Chinese). http://www.jos.org.cn/1000-9825/6388.htm

DAG Partition Algorithm for Hardware Accelerated Function Verification

HE Tian-Xiang[1], XIAO Zheng[1,2], CHEN Cen[1,2], LIU Chu-Bo[1,2], LI Ken-Li[1,2]

[1](College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China)

[2](National Supercomputing Center in Changsha, Changsha 410082, China)

Abstract: Functional verification is a basic step in VLSI design. With the popularity and development of VLSI, the feasibility and efficiency of functional verification of the whole circuit on a single processor are greatly deficient. The functional verification based on hardware accelerator divides the whole circuit into several smaller sub circuits. When the parallelism of circuit partitioning is better, the time cycle of function verification can be accelerated. Similar to other partitioning problems in circuit design, the circuit partitioning problem for hardware accelerated function verification can be abstracted into graph partitioning problem. In order to meet the requirements of hardware accelerated functional verification, an effective algorithm based on traditional multi-level graph partition strategy is proposed. The algorithm combines the idea of scheduling, and uses the critical path information and timing information of the circuit. The problem of hardware accelerated function verification is transformed into the problem of multi-level partition of directed acyclic graph. The experimental results of random circuit netlist data show that the proposed algorithm can effectively reduce the critical path length and does not cause the growth and deterioration of the number of cut edges.

Key words: very large scale integration (VLSI); hardware accelerated functional verification; directed acyclic graph (DAG); multi-level graph partition; critical path.

* 基金项目: 国家自然科学基金 (61772182, 61802032)
收稿时间: 2021-03-12; 修改时间: 2021-04-12; 采用时间: 2021-05-03; jos 在线出版时间: 2022-06-15

在进行电路设计时, 为了保证设计电路的正确性需要对电路进行功能验证. 常用的功能验证有两种方法, 一种是基于软件工具对电路逻辑进行验证, 另一种则是借助硬件加速器[1,2]实现. 硬件加速器提供的性能比软件工具要多几个数量级. 通常情况下基于软件工具的模拟验证周期需要数月, 而使用硬件加速器的验证周期则可被缩短至几天甚至几个小时[3], 极大降低了验证环节的时间成本. 同时, 随着电路设计中门原语规模的不断扩大, 需要将一个大型电路划分成多个小规模子电路, 以便能够放置在多个加速处理器或芯片上. 但是不同芯片 (处理器) 之间的高延迟和带宽限制, 应该最大化电路层次结构中每一层的连接, 确保对每个单元的评估可以路由到它的下游, 而尽可能少的在任何一条路径上造成过多的延迟. 虽然在学术界和工业界, 类似的电路划分问题被建模为图划分问题. 但是用于硬件加速功能验证的划分问题不同于传统的图划分问题, 如用于电路布局的图划分问题[4,5]. 在传统的图划分问题中, 通常以最小边割[5−7]为划分目标. 但是在不同电路分区之间并行的进行功能验证, 还应考虑电路门间信号的时序依赖性, 切边引起的通信时延和划分后模拟调度的并行性. 通过划分算法解决硬件加速功能验证问题是为了缩小问题规模, 使划分后的子集能在不同的硬件加速器上并行的进行调度, 以大幅缩小电路验证环节的验证周期. 因此, 硬件加速功能验证的划分是一个需要将图划分思想和调度思想相结合的复杂问题.

应用在电路设计领域的传统图划分问题总以最小边割为划分目标, 追求划分后各个分区间的低通信时延. 与其他划分算法相比, 多级图划分策略具有更高的效率和更好的划分结果. 多级图划分模式将整个划分过程分成 3 个阶段, 首先通过合并顶点来减少顶点数直至图的规模达到最小, 再对规模最小的图执行初始划分; 然后不断移动调整各分区边界的顶点来优化划分质量. 而硬件加速功能验证问题在追求分区间通信时延较小的基础上, 需要保证划分结果的可调度性以及调度的并行性. 因此为了有效的解决 VLSI 物理设计中硬件加速功能验证的划分问题, 本文提出一种基于调度思想的有向无环图 (directed acyclic graph, DAG) 多级划分算法. 该算法在传统多级划分模式基础上, 以 DAG 为划分模型, 使用电路的拓扑时序信息和关键路径信息对边权进行预处理, 使边权对关键路径敏感, 从而更易选择符合调度需求的切边; 并提出了基于 *rankDown* 策略的粗化算法, 并实现了对关键路径敏感的匹配模式, 以更有效地处理电路划分的时序依赖, 提高模拟调度的并行性.同时本文提出算法不会导致切边数的恶化, 减少了由于切边引入的通信时延.

本文第 1 节进行相关工作的总结. 第 2 节简单介绍硬件加速功能验证问题并对其进行问题建模. 第 3 节详细介绍本文提出算法的构造过程, 重点阐述如何使用关键路径信息和电路时序信息对边权进行预处理, 以及基于 rankDown 策略改进的粗化方法. 第 4 节介绍实验所需环境和数据集以及对实验结果的详细分析. 第 5 节给出结论以及进一步的研究方向.

1 相关工作

最早 VLSI 领域中提出的划分问题是为了解决电路布局的规模巨大, 芯片制造工艺以及电路设计工具的局限性等设计问题. 通用的解决方案是采用分治思想, 将电路划分成规模均等的子集, 从而将对大规模电路的复杂处理转化为对子电路的设计. 因此传统的电路划分问题都是以最小边割为划分目标, 尽可能地减少划分后的通信时延. Allam 等人[8]使用贪婪的聚类算法进行电路划分, 该算法首先生成一个好的初始划分, 然后使用模块交换方法来提高初始划分效果; Kolar 等人[9]提出使用模拟退火算法解决图划分问题; Sipakoulis 等人[10]为电路划分提出了改进的遗传算法, 该算法可以在搜索过程中通过评估算法表现迭代的动态更新交叉速率, 变异速率等算法参数, 不断的提高算法能力; Guo 等人[11]提出了一种基于遗传操作的混合粒子群优化算法对电路进行划分, 并使用 FM 策略[12]提高算法的局部搜索能力, 同时采用多样性变异策略来避免算法陷入局部最优; Karypis 等人[13,14]比较了多级划分模式的多个阶段, 提出了改进的粗化方案和 KL 算法[15]并对多级划分算法的能力做出了理论分析. 但是硬件加速功能验证的划分问题与传统的电路划分问题的划分目标并不相同. 硬件加速功能验证问题相较于切边引入的通信时延更关注划分结果的调度性. 因此, 以最小边割为划分目标的多级图划分策略并不能较好的解决硬件加速功能验证的划分问题.

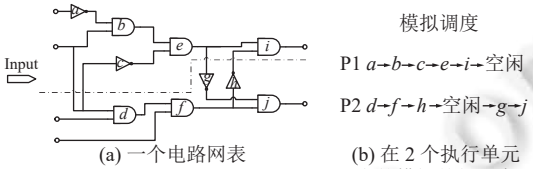
Moffitt 等人[16]在多级图划分策略的基础上, 考虑电路的时序依赖进行了超图建模; 并通过优化关键路径和有向切边来进一步降低模拟执行的指令深度, 最大化机器指令内存限制下的并行性. 同时, 在真实电路环境中的实验

结果中证明了该算法的有效性. 该算法的不足之处在于其划分性能. 和传统图划分算法的划分结果相比, 尽管该算法有效的减少了模拟深度, 但切边数却数倍增加. 并且该算法的实现较为复杂, 也并未在划分阶段考虑由于切边引入的时延.

因此, 我们所在课题组为了进一步构造硬件加速功能验证问题的优化解法, 重新对硬件加速功能验证问题进行了调研建模, 提出了一种有向图的划分算法. 该算法以多级图划分模式为基础, 以减少关键路径长度为划分目标; 将电路网表建模成带权的有向无环图, 并利用路径信息和电路时序信息进行边权预处理使其对关键路径敏感; 同时也构造了一种基于 *rankDown* 策略的粗化算法, 来更好的处理图划分模型的有向性和时序性. 基于随机网表的划分实验结果证明了本文提出算法在有效降低关键路径长度的同时能保证切边数不会恶化, 验证了算法的有效性和鲁棒性. 后文将详细介绍问题的描述和算法的具体构造.

2 问题建模

电路设计中的功能验证环节目的是确保芯片或系统的逻辑符合执行规范. 而对功能验证实行硬件加速可以解决电路设计验证环节中时间成本较高的问题. 硬件加速功能验证的划分问题就是将待验证电路划分成多个子集, 然后在硬件加速器上并行的执行调度, 以此缩短验证周期, 提高验证效率. 图 1(a) 是一个待验证的电路网表, 其中模块 *a*–*j* 表示电路的标准执行单元. 图 1(b) 表示对电路网表进行划分后的调度情况, 其中黑色圆点表示当前时刻处理器处于空闲状态. 在网表的执行逻辑中, 单元的顺利执行往往依赖于前一单元的输出, 例如单元 *e* 的执行依赖于单元 *b*, *c* 的执行输出. 因此在逻辑时序关系上, 单元 *e* 的执行往往晚于单元 *b*, *c*. 对于任意给定的逻辑顺序, 都可以构建一个模拟调度序列. 这里假设将待验证的电路划分至两个支持并行处理的处理器上进行模拟调度. 其中处理器 P1 模拟调度的最大指令深度为 5, 处理器 P2 的最大指令深度为 6. 图 1(b) 中由于执行单元 *a*, *b*, *c*, *e* 不依赖于单元的 *d*, *f*, *h* 的输出, 因此 *a*, *b*, *c*, *e* 和 *d*, *f*, *h* 可以被划分至 P1, P2 两个处理器上并行执行. 由于单元 *g* 依赖于单元 *e* 的输出, 当单元 *e* 在处理器 P1 上执行时, 处理器 P2 此时刻就会出现空闲等待. 将图划分模型中的每个顶点视作一个消耗单位时间的执行指令, 模拟深度表示的是多个处理器并行调度划分结果的最大指令长度. 模拟深度也可反映处理器进行并行调度的最大执行时间. 针对同一个图模型的不同划分结果, 划分后的模拟深度越小, 表示处理器并行处理的时间越少, 划分结果的调度并行性越高. 因此, 针对硬件加速功能验证的划分问题, 需要在划分过程中降低模拟深度, 尽量减少处理器空闲等待时间, 来提高模拟调度并行性.



(a) 一个电路网表　　(b) 在 2 个执行单元上的模拟并行调度

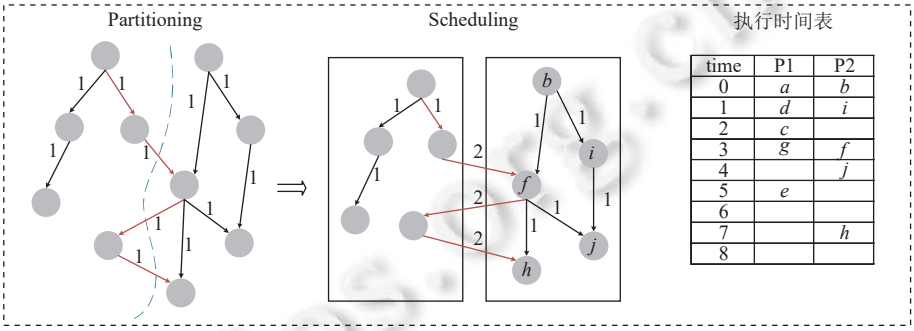
图 1　电路网表划分调度实例

硬件加速功能验证问题与 DAG 调度问题类似. 在 DAG 调度中, 任务被抽象成 DAG 中的顶点, 任务之间的依赖被抽象成 DAG 中的有向边. DAG 调度的基本目标是把不同且独立的任务分配到不同的处理器中, 使所有任务的处理时间总和最短. DAG 调度问题同样是一个完全 NP 问题. 已存在的启发式算法有基于表的算法, 例如 HEFT 调度算法[17], HSV 调度算法[18]; 基于聚类的 DAG 调度算法, 例如 CMWSL[19]. 然而, DAG 调度算法的复杂性普遍较高且很难处理硬件加速功能验证等大规模调度问题[17,20,21]. 本文将结合 DAG 调度思想使用多级图划分策略来解决硬件加速功能验证问题.

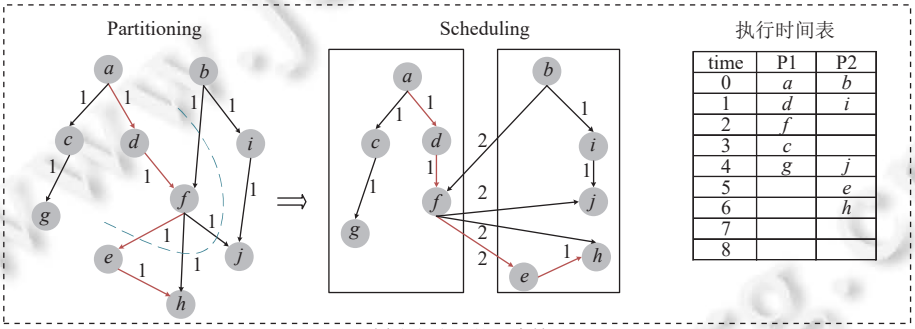
2.1 划分目标

图 2 展示了使用不同的划分方式将同一个图模型划分至两个处理器上时处理器调度的差异. 其中不同顶点之间的传输时延均设置为 1, 顶点自身处理时延忽略不计, 切边引入的通信时延设置为 2. 图 2 中有向图的关键路径

为 (a,d,f,e,h). 从图 2 可以看出, 图 2(a) 处理器的模拟深度为 7, 图 2(b) 处理器的模拟深度为 6; 图 2(a) 的划分方式切割了 3 条关键路径上的边; 而图 2(b) 中的划分方式尽可能少的对关键路径上的边进行切割, 只切割了 1 条关键路径上的边. 虽然相对于图 2(a), 图 2(b) 的划分方式额外引入了 1 条切边, 增加了更多的通信时延, 但是图 2(b) 中划分结果的关键路径长度 (考虑切边引入的通信时延) 更短, 处理器的空闲等待时间和总调度时间更短, 处理器调度的并行性更高. 相较于传统图划分问题, 硬件加速功能验证问题需要考虑划分后调度的并行性. 而图 2(b) 的划分方式使得划分结果的模拟深度更小, 处理器调度总时间更少, 该划分方式更符合硬件加速功能验证划分问题在模拟调度方面的需求.因此为了减少模拟深度, 保证划分结果的调度并行性, 在划分过程中划分算法需要尽可能的减少对关键路径上的边进行切割, 从而减少划分后关键路径长度的增长.本文将减少关键路径长度作为算法的划分目标, 保证划分结果的调度并行性.



(a) 划分方式 1 的调度情况



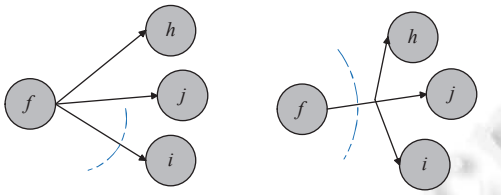
(b) 划分方式 2 的调度情况

图 2　不同划分方式在调度方面的区别

2.2　DAG 建模

在传统图划分算法中, 一些划分算法使用简单无向图作为划分模型[13], 部分划分算法使用无向超图或者有向超图进行建模[16,22,23], 也有部分研究[24,25]基于 DAG 提出了多级划分算法. 文献 [24] 提出一种 DAG 的多级无环划分算法, 并提出一种直接 k-way 划分模式. 文献 [26] 为了提高多级划分算法的划分质量, 提出了一种基于最大流-最小边割和 FM 算法的局部改进算法以及基于多网格线性解的全局搜索策略. 但诸多划分算法的划分目标通常为最小边割, 旨在减少切边引入的通信时延, 且并未在划分算法中考虑划分结果的可调度性以及关键路径在划分算法中的影响. 文献 [16] 在解决硬件加速功能验证问题时, 将电路建模成有向无环超图 (directed acyclic hypergraph, DAH), 其中每个顶点代表电路的一个标准单元, 每条有向超边代表电路中上游单元至下游单元的连接. 对于超图模型, 一条超边被切割对应着电路中多条连接被切割. 图 3 展示了在 DAG 和 DAH 模型中切割电路中同一条连接的情况. 从图 3 可知, 当切割单元 f 到单元 i 之间的连接时, DAG 模型中顶点 f 到顶点 i 的有向边会被切割; 而在 DAH 模型中, 单元 f 和单元 h,j,i 中任意一个之间的连接被切割时, 对应着的都是 DAH 中的同一条超边被切割. 第 2.1 节中分析了关键路径对于提高划分结果调度并行性的重要性. 当使用 DAH 作为划分模型时, 切割多条连接

对应的都是关键路径上同一条边被切割, 从而导致 DAH 划分结果的关键路径长度不能准确表示划分结果被模拟调度的情况. 显然, 关键路径思想并不能很好的和 DAH 划分结合. 本文第 3.1 节介绍的边权预处理策略需要计算每条边的路径信息, 而 DAH 划分模型同样不能很好的表达电路的有向路径信息. 因此, 本文基于 DAG 划分模型, 将电路转换成 DAG 型并对其进行拓扑时序分析, 从而得到每个顶点的拓扑时序信息和每条边的路径信息. 同时, 本文提出的边权预处理策略和改进的粗化方法也将结合 DAG 模型进一步, 降低模拟深度, 提升调度并行性.



(a) 在 DAG 中切割连接 (f, i) (b) 在 DAH 中切割连接 (f, i)

图 3 DAG 和 DAH 中切割同一条连接的情况

2.3 问题形式化定义

定义 $G(V,E,W_d,W_e)$ 为一个带权的有向无环图, 用来描述待功能验证的电路; 其中顶点集 V 表示电路中的模块 (逻辑门和触发单元等标准单元) 集合, 顶点的权重集 W_d 表示电路中模块对电路中不同种有限资源的需求量; 有向边集 E 表示电路中各模块间的连接和时序依赖, 初始边权 W_e 表示电路中单元间的传输时延.

硬件加速功能验证的划分问题就是将 V 中的顶点划分到数量均等的 K 个子集 $\{V_1,V_2,V_3,\ldots,V_k\}$ 中, 使各分区需求的总资源量满足电路资源约束, 并使各分区间资源负载均衡, 同时应减少划分后的关键路径长度, 并尽可能少的增加划分过程中产生的切边. 该问题的划分目标是最小化对 G 划分后的关键路径长度, 问题约束是均等划分各分区并保证分区间资源负载均衡, 即:

$$\min\left\{\max_{e\in E}\{path(e)\}\right\} \tag{1}$$

$$\text{s.t.}\begin{cases}\max\left\{(\max\{r_{ij}\}-\text{avg}\left\{\sum_{i=1}^{k}r_{ij}\right\})/\text{avg}\left\{\sum_{i=1}^{k}r_{ij}\right\},(\text{avg}\left\{\sum_{i=1}^{k}r_{ij}\right\}-\min\{r_{ij}\})/\text{avg}\left\{\sum_{i=1}^{k}r_{ij}\right\}\right\}\leqslant imbalance\\ \sum_{i=1}^{k}r_{ij}\leqslant R_j, R_j\in\{R_1,R_2,R_3,\ldots,R_m\}\\ V_1+V_2+\ldots+V_k=V, V_a\cap V_b=\varnothing, a,b\in 1,\ldots,k\end{cases} \tag{2}$$

其中, $path(e)$ 表示划分结果中包含有向边 e 的路径的最长路径长度, 最长路径长度的计算在第 3.1 节利用顶点的拓扑时序信息和路径信息求得; 集合 $\{R_1,R_2,R_3,\ldots,R_m\}$ 表示电路设计中 m 种不同资源的约束量; r_{ij} 表示第 i 个分区对第 j 种资源的需求量, $imbalance$ 表示最大可接受不平衡因子.

3 算法构造

本文构造的算法基于传统多级图划分模式, 以减少关键路径长度为划分目标, 注重提高对划分结果执行调度的并行性. 同时, 电路中各单元间存在的通信时延是影响电路功能验证效率的重要因素. 而切边会破坏电路单元间的时序关系, 引入不必要的通信时延. 因此切边数也是划分过程中的重要考量因素. 针对时延问题, 本文在计算关键路径敏感的新边权时将考虑电路单元本身的处理时延, 以及在计算划分结果的关键路径长度时考虑切边引入的通信时延, 同时使用 DAG 对电路进行建模. DAG 和 DAH 在划分时的差异见本文第 2.2 节. 图 4 为本文构造算法的基本框架图. 在该算法中, 首先通过对整个电路模型进行拓扑时序分析计算顶点的时序信息和路径信息, 再结合顶点的时序信息重新计算边权, 使每条边的边权对关键路径敏感; 然后根据顶点的拓扑时序信息计算出粗化阶段的待访问顶点队列, 对待访问顶点执行约束判断并合并满足约束条件的顶点对 (即匹配) 直至粗化图的顶点数足够小; 再对最小的粗化图执行初始划分; 最后在细化阶段使用增益调整策略对分区边界的顶点进行调整, 直至图恢复至原始图. 本节将详细介绍算法的构造过程.

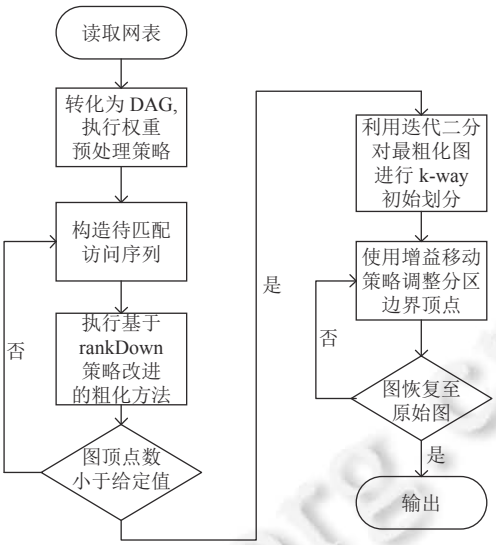


图 4　本文构造算法框架图

3.1 权重预处理

文中第 2.1 节介绍了不同的划分方式对划分结果可调度性的影响. 在划分过程中, 减少对关键路径的破坏可以更少地破坏电路单元间的时序关系, 提高划分结果的调度并行性. 因此, 如何将关键路径信息融入到划分过程中成为关键. 为此, 首先定义表示顶点拓扑时序信息的 *rankDown* 值和 *rankUp* 值, 计算见公式 (3) 和公式 (4). 顶点 p 的 *rankDown* 信息描述的是沿着有向边的方向由入度为 0 的顶点开始到顶点 p 的最长路径长度, 顶点 p 的 *rankUp* 信息描述的是沿着有向边的反方向由出度为 0 的顶点开始到顶点 p 的最长路径长度.

$$rankDown(p)=\max_{p_i\in inputs(p)}(rankDown(p_i)+W_d(p_i)+W_e(\langle p_i,p\rangle)) \tag{3}$$

$$rankUp(p)=\max_{p_i\in outputs(p)}(rankUp(p_j)+W_d(p_j)+W_e(\langle p_j,p\rangle)) \tag{4}$$

其中, $inputs(p)(outputs(p))$ 表示顶点 p 的入 (出) 顶点集合. $\langle p_i,p\rangle$ 和 $\langle p_j,p\rangle$ 表示连接两个顶点的边. 初始边权 $W_e(\langle p_i,p\rangle)$ 和 $W_e(\langle p_j,p\rangle)$ 表示顶点之间的传输时延. 初始点权 $W_d(p_i)$ 和 $W_d(p_j)$ 表示顶点自身的处理时延. 这里将电路网表中标准单元的入出引脚之间的处理时延抽象为 DAG 模型中顶点的自身处理时延. 如果顶点没有入 (出) 顶点, 则其 $rankDown(rankUp)$ 值设置为 0.

然后, 根据公式 (5) 使用拓扑时序信息定义 DAG 中包含有向边 e 的所有路径中最大的路径长度.

$$path(e)=rankDown(source(e))+W_e(e)+rankUp(sink(e)) \tag{5}$$

其中, $source(e)$ 和 $sink(e)$ 表示有向边连接的上游顶点和下游顶点, $W_e(e)$ 表示边 e 的初始边权.

最后, 为了充分利用多级划分过程中的关键路径信息, 重新定义了新边权 $W'_e(e)$. 新边权 $W'_e(e)$ 通过结合有向边的最长路径信息和 DAG 的全局最短路径信息计算得到. 边权 $W'_e(e)$ 的计算详见公式 (6) 和公式 (7). 边权 $W'_e(e)$ 定义了该边所在路径的最长路径长度和全局最短路径长度之间的差异. 边权 $W'_e(e)$ 越小, 表示该边在关键路径上的可能性越小, 从而更适合被切割. 通过定义关键路径敏感的边权可以使得切边的选择更加符合划分目标, 并且在资源负载均衡的同时也能更灵活的判断出适合切分的有向边.

$$path_{\min}=\min_{e\in E}(path(e)) \tag{6}$$

$$W'_e(e)=path(e)-path_{\min} \tag{7}$$

在对电路进行 DAG 建模时, 初始边权代表着电路中该边连接的上游单元和下游单元之间的传输时延. 为了构造关键路径敏感的划分策略, 基于 DAG 划分模型的初始边权再利用公式 (3)–公式 (7) 计算出新边权 $W'_e(e)$; 然后用新边权 $W'_e(e)$ 代替初始边权以便在划分的粗化阶段更优的进行顶点合并以及在初始划分阶段更好的进行切边选择.

细化阶段的增益移动策略会通过计算增益值来判断是否调整顶点的分区位置. 而增益值的计算依赖于初始边权 $W_e(e)$, 因此增益移动策略也需要根据重新计算的边权值 $W'_e(e)$ 进行调整. 边权预处理策略将关键路径信息融入到边权中, 以便多级划分算法中的各个阶段都能将其作为全局指导更优的进行顶点选择, 分区划分以及顶点调整.

3.2 基于 *rankDown* 策略改进的粗化方法

多级划分算法中的粗化阶段可大致分为 3 个步骤: (1) 构造随机待匹配序列, 选择未被匹配顶点作为待匹配顶点; (2) 待匹配顶点与其未匹配邻接顶点执行匹配模式, 构造全局匹配映射; (3) 根据匹配映射构造下一级图结构. 在多级划分的粗化中, 随机匹配模式[13]中合并顶点对的选择随机性较强, 而最大边权匹配模式[13]对于顶点的匹配只简单进行边权的比较而不考虑路径信息对调度的影响. 因而简单的多级划分算法不足以满足硬件加速功能验证的划分问题对于调度并行性的需求. 为此, 本节基于顶点的 *rankDown* 值重新设计了粗化方法并改进了两种匹配模式.

基于 *rankDown* 策略的随机匹配模式: 按照顶点 *rankDown* 值递增的顺序构造待匹配序列. 首先遍历待匹配访问序列, 为当前未匹配顶点随机选择一个未匹配邻接顶点进行匹配; 然后判断这两个顶点合并后是否满足资源负载均衡; 并将匹配过的两个顶点均标记为已匹配状态. 在这个过程中, 将边权按照加权取和的规则进行动态更新. 如果有多个顶点可与待匹配顶点进行匹配, 则选择具有较低 *rankDown* 值的顶点进行匹配.

基于 *rankDown* 策略的最大边权匹配模式: 按照顶点 *rankDown* 值递增的顺序构造待匹配序列. 首先遍历待匹配访问序列, 在其邻接边列表中找到具有最大边权的有向边, 同时判断这两个顶点合并后是否满足资源负载均衡; 然后合并由最大边连接的两个顶点. 由于更新后的边权是关键路径敏感的, 较大边权的边位于关键路径上的可能性更高. 此时顶点的匹配可避免后续划分过程中该有向边被切割, 从而更好地保护关键路径不被切割, 保证调度的并行性. 同时, 边权进行加权取和的更新. 如果存在多个顶点可以与当前待匹配顶点进行匹配, 将 *rankDown* 值较低的顶点作为与之匹配的顶点. 此时, 若有多个顶点具有相同的 *rankDown* 值, 则取序列号较低的顶点进行匹配.

由于顶点的 *rankDown* 值和 *rankUp* 值只是从不同的方向性描述 DAG 的拓扑时序信息, 因此在改进的两种匹配模式中仅选用顶点的 *rankDown* 值作为构造访问序列和匹配的影响因子. 在本文算法实现中, 粗化阶段将混合使用这两种改进后的匹配模式. 改进后的粗化步骤如下.

(1) 按照顶点 *rankDown* 值递增的顺序构造待匹配序列, 选择未匹配顶点作为待匹配顶点;

(2) 执行改进的随机匹配模式或者最大边权匹配模式, 构造顶点匹配映射;

(3) 根据顶点匹配结果构造下一级粗化图结构并维持顶点点权以及 *rankDown* 值和 *rankUp* 值的动态更新.

3.3 算法步骤

本文所构造的多级划分算法伪代码如下. 在本文构造算法中, 首先使用转换函数 *Convert*() 将电路网表数据 *data* 转换为带权的 DAG 模型 *graph*, 其初始边权为顶点之间的传输时延, 初始点权为顶点自身的处理时延; 再利用有向图 *graph* 的电路时序信息和拓扑结构计算出每个顶点的 *rankDown* 和 *rankUp* 信息, 并计算出每条有向边的最长路径信息 $path(e)$ 和全局最短路径 *minPath*, 最后使用得到的新边权 $W(e)$ 替换初始边权, 使每条边的边权对关键路径敏感; 在粗化阶段, 先使用 *SelectSortKeysInc*() 函数利用顶点的 *rankDown* 信息计算出粗化阶段的待匹配顶点队列 *perm*, 遍历待匹配序列对待匹配顶点执行基于 *rankDown* 策略改进的顶点匹配模式 *Match_rankdown*(), 构造匹配映射结果 *match* 并使用 *UpdateRankAndWeight*() 函数进行粗化图的数据更新, 再使用 *CreateCoarseGraph*() 函数维护每一级粗化图结构的数据完整性直至粗化图的顶点数足够小; 再继续对最小的粗化图执行初始划分 *Init2WayPartition*(), 使用迭代二分方法将粗化图划分至指定分区数量; 最后在细化阶段使用增益调整策略 *Refine2Way*() 对分区边界的顶点进行调整, 直至图映射回原始图. 最终返回算法的划分结果 *partition*.

算法 1. 以关键路径为目标的多级划分算法.

输入: 电路网表数据 *data*;

输出: 划分结果 *partition*.

1. 图结构 *graph*, 顶点集合 *P*, 有向边集合 *E*, 待访问顶点序列 *perm*, 分区结果 *partition*, 匹配映射 *match*, 粗化比例 *COARSEN_FRACTION*;
2. $graph = Convert(data)$; //将网表数据转化为带权的 DAG 模型
3. for p in P
4. $rankDown(p) = \max_{p_i \in inputs(p)}(rankDown(p_i) + W_d(p_i) + W_e(\langle p_i, p\rangle))$;
5. $rankUp(p) = \max_{p_j \in outputs(p)}\left(rankUp\left(p_j\right) + W_d\left(p_j\right) + W_e\left(\langle p_j, p\rangle\right)\right)$;
6. for e in E
7. $path(e) = rankDown(source(e)) + W_e(e) + rankUp(sink(e))$;
8. $minPath = \min(path(e), minPath)$;
9. for e in E
10. $W(e) = path(e) - minPath$;
11. $perm = SelectSortKeysInc()$; //构造待匹配顶点访问序列
12. while $graph \to nvetex \leqslant COARSEN\_FRACTION$: //顶点数量足够小
13. $match = Match\_rankdown(graph)$; //使用匹配模式
14. $UpdateRankAndWeight(graph)$; //更新权值和 rank 值
15. $graph = CreateCoarseGraph(graph, match)$;//构造下一级粗化图
16. end while
17. $partition = Init2WayPartition(graph)$; //初始划分
18. $partition = Refine2Way(graph)$; //细化方法, 调整分区结果

3.4 评估函数

针对电路硬件加速功能验证的划分问题, 本文仅考虑以减少划分结果的关键路径长度为划分目标的情况. 算法采用划分目标作为主要评估函数, 以划分后产生的切边数作为辅助评估函数, 具体定义如下:

$$f1 = \max_{e \in E}\{path(e)\} \tag{8}$$

$$f2 = \sum_{i=1}^{k}\sum_{j=i+1}^{k} L_{ij} \tag{9}$$

其中, E 表示有向图模型的有向边集合, $path(e)$ 为公式 (5) 定义的包含有向边 e 的最长路径长度, 这里计算的最长路径长度应考虑划分后切边引入的分区间通信时延; L_{ij} 表示分区 i 和分区 j 之间的有向切边数.

4 实验以及结果分析

4.1 测试数据和测试环境

本文算法使用编程语言 C++进行实现, 本文涉及的所有实验均在主频为 2.5 GHz 的 Intel(R) Xeon(R) Gold 5118 处理器, 内存为 264 GB, 操作系统为 Ubuntu 的服务器环境下进行测试. 为了尽可能避免实验测试结果的偶然性, 本文将使用 GNL 随机网表生成工具 (官方网址: https://users.elis.ugent.be/~dstrooba/gnl/) 生成的随机电路网表数据作为算法测试数据. 测试数据集 gnl100w, gnl200w, gnl500w 说明见表 1, 其中关键路径长度为 DAG 划分模型中的最长路径长度.

GNL 是一种生成逻辑电平基准电路的工具. GNL 的电路生成是基于自上而下的递归聚类, 将子模块 (电路) 合成更大规模的模块直至生成完整的电路; 同时模块的生成遵循一定的电路生成准则, 该准则可以防止逻辑环的形成, 并且可以在模块生成过程中控制其最长路径长度. 由于 GNL 生成工具是按照递归聚类的方式将子模块组合成最终电路网表, 导致其生成较大规模的电路数据时效率较低. 为了提高随机网表数据的生成效率, 我们使用了一种拼接方案; 该方案将 GNL 工具生成的多个小规模电路网表拼接成较大规模的网表数据, 以此来提高网表数据生

成效率. 同时, 在 GNL 工具生成随机网表数据后, 利用电路单元输入输出端口之间的映射关系将整个网表转换成将带权的有向无环图. 转换后有向图的无环性依赖于 GNL 生成网表的无环性.

表 1 测试数据集说明

数据集名称	点数	边数	关键路径长度
gnl100w	1 048 143	2 491 103	292
gnl200w	2 080 655	4 996 669	391
gnl500w	5 300 002	12 485 557	586

4.2 算法有效性分析

为了分析本文构造算法的有效性, 对 METIS 划分算法、使用原有粗化方法和边权预处理策略的 METIS 算法 (METIS only with preprocessing edge-weigh, METISPW)、结合 *rankDown* 粗化策略和边权预处理策略的 METIS 算法 (METIS with rankdown-coarsen and preprocessing edge-weigh, METISRP) 以及 hMETIS 等多种算法进行了一系列的划分实验. 其中 hMETIS 算法[22]是基于超图的多级图划分算法, 更适合解决大规模图划分任务. 本节结合应用场景对每个数据集进行划分数 K 为 5, 20, 50, 100, 200 的划分测试, 并测试多次取平均切边数和平均关键路径长度以及算法执行时间作为实验结果. 表 2 的 path 一栏记录关键路径长度, cut 一栏记录切边数, time 一栏记录算法执行时间, Total 一行表示对应列的累加和, Average 一行表示在 METIS 算法划分结果的基础上, 其余算法较之在各个指标的优化 (恶化) 比例. 这里进行划分实验的算法实现依赖于权重预处理后对关键路径敏感的边权, 因此在进行这些算法划分之前, 划分算法需要先对输入图进行权重预处理, 更新边权. 在第 2.3 节对划分问题进行了描述, 提出划分算法以减少关键路径长度作为划分目标并需要保持分区之间资源负载均衡. 因此, 本文基于 METIS 进行改进的算法策略期望在不破坏 METIS 原有的负载均衡特性下满足减少分区结果关键路径长度的划分目标. 本文的实验结果均是在 METIS 默认均衡因子为 0.03 的条件下进行的实验, 即本文实验的划分结果依旧保持了良好的均衡性.

4.2.1 算法的有效性

本文从分区数, 切边权重和点权以及边权多个变化维度来对数据集进行综合划分测试, 并取平均关键路径长度和平均切边数作为实验结果来进行算法对比. 表 2 为本文不同策略下划分数量不同时对实验数据集的测试结果, 同时指定切边权重为常数 40 再测试划分后的关键路径长度. 从表 2 的实验结果可以看出, 多种算法不可避免都会导致划分之后的关键路径长度较原始图有所增加. METISPW 算法划分后的关键路径长度相比于 METIS 算法有显著减少, 且切边数仅微弱幅度的增加. 结合边权预处理策略和改进粗化策略的 METISRP 算法较 METISPW 算法的划分结果有了进一步的提升. 虽然 METISRP 算法的实验结果并未全部优于 METISPW 算法, 但是 METISRP 算法对各数据集进行划分后的平均关键路径长度小于 METISPW 算法, METISPW 算法较原始 METIS 划分结果的平均关键路径优化幅度为 15% 左右, METISRP 算法的平均关键路径长度优化幅度为 17% 左右. 但由于边权预处理策略和改进粗化策略的加入, METISRP 算法和 METISPW 算法产生的切边数相较于 METISPW 算法的划分结果呈现了微弱的增长趋势, METISRP 算法和 METISPW 算法的平均切边数增长幅度均在 9% 左右.

表 2 中同样展示了 hMETIS 的实验结果. 除了在分区数量为 5 时的实验结果稍劣于 hMETIS 算法, 在其余划分情况下, METISRP 算法和 METISPW 算法的划分结果均优于 hMETIS 算法. 针对算法的整体执行时间, METISRP 算法较原始 METIS 算法仅增加了维护 *rankDown* 和 *rankUp* 信息和顶点合并决策的时间开销, 执行时间仅平均增长了 15% 左右. 而针对同样的数据集和划分分数, 由于使用超图建模和基于超图的划分方式带来巨大的时间开销, hMETIS 的执行时间是 METIS 算法和 METISRP 算法的 150 倍左右. 较 hMETIS 算法, METISRP 算法划分结果的关键路径长度更小, 切边数的增幅也更小, 且总执行时间大幅度减少.

以上实验结果反映出, 仅基于 METIS 算法对输入图进行边权预处理形成的 METISPW 算法, 其实验结果与 METIS 算法相比实验结果上有明显提升. 并且再加入 *rankDown* 粗化策略后形成的 METISRP 算法的划分效果又

有了些许提升. 可见边权预处理策略将关键路径信息和拓扑时序信息融入到边权中提高了算法对关键路径的敏感程度, 在算法的初始划分阶段和细化阶段可以利用边权更好的选择切边和进行分区边界顶点的移动调整, 从而有效减少了划分后的关键路径长度. 改进的粗化策略消除了传统匹配模式访问顶点的随机性, 使用顶点的 *rankDown* 信息创建访问序列更能反映有向图模型的拓扑结构和时序依赖; 改进后的顶点融合策略在合并顶点时考虑顶点的拓扑结构, 选择层次结构更深且满足匹配约束的顶点进行匹配, 更有目的性的选择对关键路径友好的顶点对进行合并, 进一步减少了对电路单元间时序关系的破坏. 与其他算法对比, 改进后的算法可以得到更少的关键路径长度, 使传统的多级图划分模式在调度能力上有了提升. 同时, 本文提出的策略并没有增加移动分区边界顶点的操作, 因此并未明显增加划分结果的切边数, 切边的平均涨幅维持在较低的范围内, 保证了在调度时不会因为切边数恶化而引入较多的通信时延. 边权预处理策略关注初始边权的更新, 而改进的粗化策略关注顶点合并时的决策判断, 两种策略并不会增加 METIS 算法的整体时间复杂度, 因此较 METIS 算法相比 METISRP 算法的执行时间并未发生明显变化.

表 2 实验结果

Dataset	Part	METIS			METISPW			METISRP			hMETIS		
		path	cut (k)	time (s)	path	cut (k)	time (s)	path	cut (k)	time (s)	path	cut (k)	time (s)
gnl100w	5	683	212	3.4	677	227	3.9	785	223	3.9	638	233	425
	20	845	314	5.4	749	333	6.4	664	337	6.8	819	353	610
	50	1 077	395	6.2	858	431	7.1	840	424	7.5	966	449	772
	100	1 065	463	6.8	969	510	7.9	910	503	8.1	1 080	526	1 008
	200	1 195	524	9.0	977	575	9.0	990	577	9.2	1 138	607	1 365
gnl200w	5	764	431	7.5	603	431	8.5	650	439	8.6	665	443	1 214
	20	1 065	629	12.0	869	685	13.3	831	686	14.2	816	671	1 922
	50	1 233	740	13.0	976	802	15.4	944	811	16.8	1 124	835	2 152
	100	1 261	842	14.3	1 068	947	17.4	1 033	926	18.0	1 094	964	2 580
	200	1 270	944	16.1	1 165	1 071	19.2	1 121	1 096	20.3	1 204	1 115	3 286
gnl500w	5	1 245	1 064	23.2	783	1 098	25.2	784	1 113	25.7	965	1 119	3 490
	20	1 293	1 592	33.9	1 057	1 745	35.4	1 108	1 791	36.5	1 257	1 750	4 163
	50	1 398	1 895	38.6	1 305	2 076	40.3	1 109	2 095	42.5	1 474	2 129	4 894
	100	1 495	2 109	43.2	1 208	2 299	47.9	1 143	2 290	48.7	1 667	2 391	6 326
	200	1 439	2 257	44.2	1 381	2 492	51.5	1 301	2 500	52.6	1 705	2 624	7 622
Total		17 328	14 411	277	14 645	15 722	308	14 213	15 811	319	16 612	16 209	41 829
Average (%)		100	100	100	85	108.5	113	83	108.8	117	95	118.6	14922

但是由于多级划分算法的启发式性质, 划分算法的划分结果并不总是较优的; 且粗化方法, 初始划分和细化方法都有随机选择访问顶点的操作, 多级划分算法的划分结果会在合理的范围内波动; 同时最大边权匹配不能一直保持高效的顶点合并效率, 在某些划分分数时划分算法会由于粗化阶段顶点合并不充分降低划分质量; 以上多种因素都可能会导致本文提出算法的划分结果发生波动, 导致划分结果的关键路径长度出现逆增长的现象. 文献 [27] 为了解决粗化过程中顶点匹配不充分而导致划分质量下降的情况, 基于最大边权匹配模式提出 brotherly matching 模式, adoption matching 模式和 community matching 模式来提高顶点匹配效率, 优化粗化质量. 与 METIS 算法和 hMETIS 算法的实验结果对比, 可以验证本文提出算法的有效性. 在以关键路径长度为主要划分目标的划分问题上, 本文提出的算法思想行之有效.

4.2.2 算法的鲁棒性

在划分实验中存在许多因子会影响算法的整体划分结果, 例如待划分的图规模, 分区数, 计算关键路径长度时赋予切边的权重值以及点权和边权等. 为了探究在一些划分因素发生变化时本文提出算法是否可以继续保持划分结果的有效, 本文分别进行了分区数不同和切边权重不同时的划分实验. 图 5 展示了在切边权重等其他因素一定时, 不同的划分算法对同一数据集 gnl100w 划分后的关键路径长度的平均减少幅度. 图 6 展示了在切边权重等其

他因素一定时, 算法对数据集 gnl500w 划分后的关键路径长度的平均减少幅度. 图 5 和图 6 展示的实验结果均以 METIS 算法的划分结果为对比基准. 从图 5 可以看出, 对于不同的划分分区数, METISPW 算法和 METISRP 算法划分后关键路径长度的平均减少幅度基本上能稳定维持在 10% 左右; 从图 6 可以看出, 对于不同的划分分区数, METISPW 算法和 METISRP 算法划分后关键路径长度的平均减少幅度也可稳定维持在 15%~18% 范围内. 而 hMETIS 算法对数据集 gnl100w 和数据集 gnl500w 进行划分得到的实验结果相差较大; 从图 5 可以看出, hMETIS 算法划分后的关键路径长度的平均减少幅度仅在 3%~4% 左右; 从图 6 可知, 在对数据集 gnl500w 进行划分时, hMETIS 算法划分后的关键路径长度又出现大量逆增长的情况, 平均减少幅度为−2% 左右. 同时, 虽然用于本文实验的数据集的点数和边数均在数百万级别, 但是随着图规模的增大, 本文提出的 METISRP 算法仍能有效减少关键路径长度并保证切边数不出现恶化情况, 仍然保持着较优的划分结果.

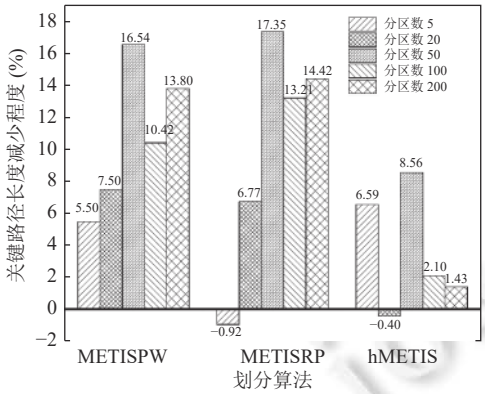


图 5 分区数不同时, 对数据集 gnl100w 的划分结果

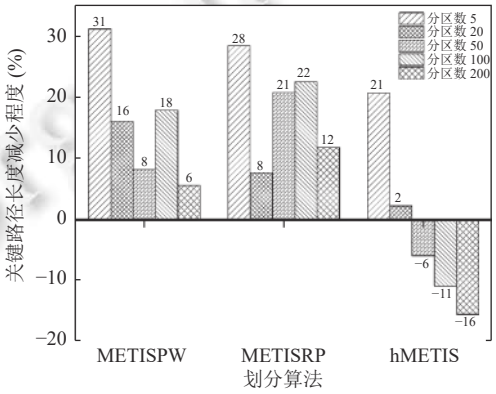


图 6 分区数不同时, 对数据集 gnl500w 的划分结果

在硬件加速器对划分结果的模拟调度中, 切边权重值表示由于不同分区间存在依赖关系从而引入的通信时延, 此时切边权重值越大, 则调度过程中分区之间通信代价越大. 而划分后关键路径长度的计算依赖于切边的权重值, 所以不同的切边权重也会导致不同的划分结果. 图 7 展示了当分区数固定为 200, 切边的权重值不同时, 各个算法对数据集 gnl500w 划分后关键路径长度的减少幅度. 从图 7 可以看出, 在设置不同的切边权重后, METISRP 算法和 METISPW 算法划分后的关键路径长度几乎均能保持 6%~16% 左右的有效减少. 而当切边权重值不同时, hMETIS 算法的实验结果波动较为明显, 并且出现了关键路径长度大幅增加的情况. 从以上实验结果可知, 当某些划分因素变化时, 本文提出的 METISRP 算法可一直保持着关键路径长度的有效减少, 实验结果较其他划分算法鲁棒性更强.

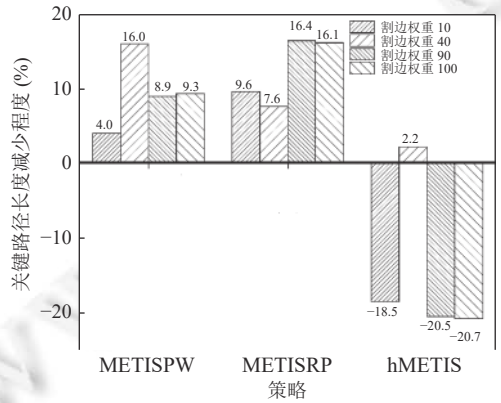


图 7 切边权重不同时, 对数据集 gnl500w 的划分结果

5 结束语

为了解决 VLSI 领域硬件加速功能验证的划分问题, 针对已有算法[16]在降低模拟深度的同时切边数急剧增长的性能缺陷, 本文提出一种有效且鲁棒的图划分算法, 该算法基于传统多级图划分模式并结合调度思想, 对 DAG 模型进行划分. 已有实验可以证明该算法在有效减少模拟深度的同时切边数只会小幅增加, 不会出现切边数恶化的情况, 减少了不必要引入的通信时延, 同时维持了分区结果的资源负载平衡. 本文下一步工作方向是探索多级划分中细化算法的改进以及其他启发式算法是否会进一步优化关键路径指标, 并探究功能验证时的多维权重约束问题, 以及探究强化学习在该问题上的应用.

致谢 在此, 我们向为本文工作提供修改意见的匿名审稿人表示衷心的感谢.

References:

[1] Mammo BW. Reining in the functional verification of complex processor designs with automation, prioritization, and approximation [Ph.D. Thesis]. Ann Arbor: The University of Michigan, 2017.

[2] Blank T. A survey of hardware accelerators used in computer-aided design. IEEE Design & Test of Computers, 1984, 1(3): 21–39. [doi: 10.1109/MDT.1984.5005647]

[3] Schubert KD. POWER7: Verification challenge of a multi-core processor. In: Proc. of the 2009 Int'l Conf. on Computer-Aided Design. San Jose: ACM, 2009. 809–812. [doi: 10.1145/1687399.1687551]

[4] Chandy JA, Banerjee P. A parallel circuit-partitioned algorithm for timing driven cell placement. In: Proc. of the Int'l Conf. on Computer Design VLSI in Computers and Processors. Austin: IEEE, 1997. 621–627. [doi: 10.1109/ICCD.1997.628930]

[5] Gao T, Vaidya PM, Liu CL. A new performance driven placement algorithm. In: Proc. of the 1991 IEEE Int'l Conf. on Computer-Aided Design Digest of Technical Papers. Santa Clara: IEEE, 1991. 44–47. [doi: 10.1109/ICCAD.1991.185187]

[6] Hill D. Alternative strategies for applying min-cut to VLSI placement. In: Proc. of the 1988 IEEE Int'l Conf. on Computer Design: VLSI. Rye Brook: IEEE, 1988. 440–444. [doi: 10.1109/ICCD.1988.25739]

[7] Mak WK. Faster min-cut computation in unweighted hypergraphs/circuit netlists. In: Proc. of the 2005 IEEE VLSI-TSA Int'l Symp. on VLSI Design, Automation and Test, 2005. (VLSI-TSA-DAT). Hsinchu: IEEE, 2005. 67–70. [doi: 10.1109/VDAT.2005.1500022]

[8] Allam MW, Vannelli A, Elmasry MI. A clustering algorithm for circuit partitioning. In: Proc. of the CCECE1997. Canadian Conf. on Electrical and Computer Engineering. Engineering Innovation: Voyage of Discovery. Conf. Proc. St. John's: IEEE, 1997. 12–14. [doi: 10.1109/CCECE.1997.614777]

[9] Kolar D, Puksec JD, Branica I. VLSI circuit partition using simulated annealing algorithm. In: Proc. of the 12th IEEE Mediterranean Electrotechnical Conf. (IEEE Cat. No. 04CH37521). Dubrovnik: IEEE, 2004. 205–208. [doi: 10.1109/MELCON.2004.1346809]

[10] Sipakoulis GC, Karafyllidis I, Thanailakis A. Genetic partitioning and placement for VLSI circuits. In: Proc. of the 6th IEEE Int'l Conf. on Electronics, Circuits and Systems (Cat. No. 99EX357). Paphos: IEEE, 1999. 1647–1650. [doi: 10.1109/ICECS.1999.814490]

[11] Guo WZ, Chen GL, Xiong NX, Peng SJ. Hybrid particle swarm optimization algorithm for VLSI circuit partitioning. Ruan Jian Xue Bao/Journal of Software, 2011, 22(5): 833−842 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3980.htm [doi: 10.3724/SP.J.1001.2011.03980]

[12] Fiduccia CM, Mattheyses BM. A linear-time heuristic for improving network partitions. In: Proc. of the 19th Design Automation Conf. Las Vegas: IEEE, 1982. 175−181. [doi: 10.1109/DAC.1982.1585498]

[13] Karypis G, Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal of Scientific Computing, 1998, 20(1): 359–392. [doi: 10.1137/S1064827595287997]

[14] Karypis G, Kumar V. Analysis of multilevel graph partitioning. In: Proc. of the 1995 ACM/IEEE Conf. on Supercomputing. San Diego: IEEE, 1995. 29. [doi: 10.1109/SUPERC.1995.242800]

[15] Kernighan BW, Lin S. An efficient heuristic procedure for partitioning graphs. The Bell System Technical Journal, 1970, 49(2): 291–307. [doi: 10.1002/j.1538-7305.1970.tb01770.x]

[16] Moffitt MD, Sustik MA, Villarrubia PG. Robust partitioning for hardware-accelerated functional verification. In: Proc. of the 48th Design Automation Conf. San Diego: ACM, 2011. 854–859. [doi: 10.1145/2024724.2024915]

[17] Topcuoglu H, Hariri S, Wu MY. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. on Parallel and Distributed Systems, 2002, 13(3): 260–274. [doi: 10.1109/71.993206]

[18] Xie GQ, Li RF, Li KQ. Heterogeneity-driven end-to-end synchronized scheduling for precedence constrained tasks and messages on networked embedded systems. Journal of Parallel and Distributed Computing, 2015, 83: 1–12. [doi: 10.1016/j.jpdc.2015.04.005]

[19] Kanemitsu H, Hanada M, Nakazato H. Clustering-based task scheduling in a large number of heterogeneous processors. IEEE Trans. on Parallel and Distributed Systems, 2016, 27(11): 3144–3157. [doi: 10.1109/TPDS.2016.2526682]

[20] Liu CB, Li KL, Liang J, Li KQ. COOPER-SCHED: A cooperative scheduling framework for mobile edge computing with expected deadline guarantee. IEEE Trans. on Parallel and Distributed Systems, 2019. [doi: 10.1109/TPDS.2019.2921761]

[21] Zhao S, Dai XT, Bate I, Burns A, Chang WL. DAG scheduling and analysis on multiprocessor systems: Exploitation of parallelism and dependency. In: Proc. of 2020 IEEE Real-time Systems Symp. Houston: IEEE, 2020. 128–140. [doi: 10.1109/RTSS49844.2020.00022]

[22] Karypis G, Aggarwal R, Kumar V, Shekhar S. Multilevel hypergraph partitioning: Applications in VLSI domain. IEEE Trans. on Very Large Scale Integration (VLSI) Systems, 1999, 7(1): 69–79. [doi: 10.1109/92.748202]

[23] Karypis G, Kumar V. Multilevel k-way hypergraph partitioning. In: Proc. 1999 Design Automation Conf. (Cat. No. 99CH36361). New Orleans: IEEE, 1999. 343–348. [doi: 10.1109/DAC.1999.781339]

[24] Herrmann J, Kho J, Uçar B, Kaya K, Çatalyürek ÜV. Acyclic partitioning of large directed acyclic graphs. In: Proc. of the 17th IEEE/ACM Int'l Symp. on Cluster, Cloud and Grid Computing (CCGRID). Madrid: IEEE, 2017. 371–380. [doi: 10.1109/CCGRID.2017.101]

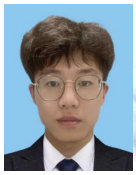
[25] Herrmann J, Özkaya MY, Uçar B, Kaya K, Çatalyürek ÜV. Multilevel algorithms for acyclic partitioning of directed acyclic graphs. SIAM Journal on Scientific Computing, 2019, 41(4): A2117–A2145. [doi: 10.1137/18M1176865]

[26] Sanders P, Schulz C. Engineering multilevel graph partitioning algorithms. In: Proc. of the 19th European Symp. on Algorithms. Saarbrücken: Springer, 2011. 469–480. [doi: 10.1007/978-3-642-23719-5_40]

[27] Davis TA, Hager WW, Kolodziej SP, et al. Algorithm 1003: Mongoose, a graph coarsening and partitioning library. ACM Trans. on Mathematical Software, 2020, 46(1): 7. [doi: 10.1145/3337792]

附中文参考文献:

[11] 郭文忠, 陈国龙, Xiong NX, 彭少君. 求解VLSI电路划分问题的混合粒子群优化算法. 软件学报, 2011, 22(5): 833−842. http://www.jos.org.cn/1000-9825/3980.htm [doi: 10.3724/SP.J.1001.2011.03980]



何天祥(1998－), 男, 硕士生, 主要研究领域为图计算, 图划分.

刘楚波(1988－), 男, 博士, 副教授, 博士生导师, CCF 专业会员, 主要研究领域为并行分布式计算, 博弈论, 体系结构, 人工智能.

肖正(1981－), 男, 博士, 副教授, 博士生导师, CCF 专业会员, 主要研究领域为协同优化, 并行分布式系统, 高性能计算, 智能信息处理, 大数据.

李肯立(1971－), 男, 博士, 教授, 博士生导师, CCF 会士, 主要研究领域为并行分布式处理, 超级计算与云计算, 面向大数据和人工智能的高效能计算.

陈岑(1985－), 男, 博士, 主要研究领域为并行与分布式计算, 机器学习, 深度学习.