ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.12, December 2010, pp.3106−3115
doi: 10.3724/SP.J.1001.2010.03676

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

一种高效的凸连通子图枚举算法*

薄 拾+, 葛 宁, 林孝康

(清华大学 电子工程系,北京 100084)

Efficient Algorithm for Convex Connected Subgraph Enumeration

BO Shi+, GE Ning, LIN Xiao-Kang

(Department of Electronic Engineering, Tsinghua University, Beijing 100084, China)

+ Corresponding author: E-mail: boshi99@mails.tsinghua.edu.cn

Bo S, Ge N, Lin XK. Efficient algorithm for convex connected subgraph enumeration. *Journal of Software*, 2010,21(12):3106−3115. http://www.jos.org.cn/1000-9825/3676.htm

Abstract: Enumerating convex connected subgraphs from data flow graphs of application hot-spots is required when designing instructions for a configurable processor. To achieve fast enumeration, properties of convex subgraphs of directed acyclic graph are studied. Based on the properties of convex subgraphs and adjacency of vertices, AS (adjacent search), a novel algorithm for enumerating convex connected subgraphs satisfying I/O constraints is presented. Results of experiments show that AS algorithm is more efficient than the existing algorithm, and rate is 10~1000X as fast. While the existing algorithm fails on large scale data-flow graphs, the AS algorithm is still able to accomplish enumeration successfully.

Key words: convex connected subgraph; directed acyclic graph; dataflow graph; enumeration; configurable processor; custom instruction

摘 要: 在可配置处理器的定制指令设计过程中,需要提取热点代码数据流图的凸连通子图.为实现子图的快速枚举,对有向无环图内的凸子图特性进行了研究.根据凸子图特性和节点邻接关系,提出了一种 AS(adjacent search)算法用于枚举有向无环图内满足 I/O 端口约束的凸连通子图.实验数据显示,AS 算法比现有算法具有更高的效率,加速比可达 10~1000X.当现有算法因数据流图规模较大而失效时,应用 AS 算法仍能成功完成子图枚举.

关键词: 凸连通子图;有向无环图;数据流图;枚举;可配置处理器;定制指令

中图法分类号: TP301 **文献标识码**: A

随着新技术与新业务的不断涌现,电子系统尤其是消费电子产品将更多功能集于一身.在这种情况下,专用电路在灵活性和成本方面的劣势日益凸显.越来越多的功能和处理任务寄希望于采用嵌入式处理器来实现.尽管如此,通用的嵌入式处理器往往不能满足一部分处理任务的性能要求.然而,设计全新的处理器是一项复杂繁重的工作,因此人们希望寻找一种有效的手段以较低的设计代价实现嵌入式处理器性能的提高.于是,可配置处理器应运而生.可配置处理器巧妙地结合了可编程处理器的灵活性和专用硬件电路的速度优势.

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2007AA01Z2b3 (国家高技术研究发展计划(863)); the National Basic Research Program of China under Grant No.2007CB310608 (国家重点基础研究发展计划(973))

Received 2008-10-24; Revised 2009-03-30; Accepted 2009-07-07

可配置处理器利用定制指令(custom instructions,简称 CI)来加速原本不擅长的计算任务.为获得最佳优化效果,CI 往往针对目标应用的热点代码(hot-spot)进行设计.对于规模小且结构简单的应用程序,可以采用人工方式进行热点提取和分析.然而,当应用程序规模变大,则必须借助自动工具完成程序数据流图(dataflow graph,简称 DFG)的探索(如图 1).具体地说,热点代码 DFG 的子图集合中存在着一部分子图具有硬件实现的可能和价值,设计者希望从 DFG 中提取这些子图.DFG 是一种典型的有向无环图(directed acyclic graph,简称 DAG),于是在 CI 的设计过程中需要解决 DAG 中特定类型子图的提取问题.

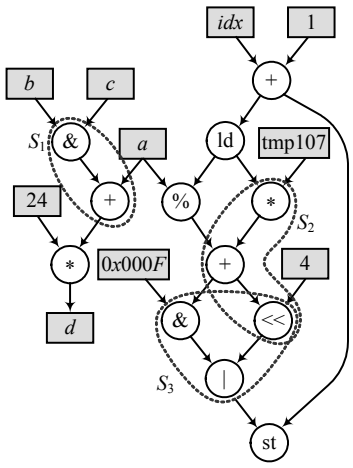


Fig.1 DFG exploration

图 1 数据流图探索

本文从 CI 设计问题出发,提出一种针对 DAG 中凸连通子图的枚举算法.第 1 节回顾子图枚举的研究现状,对部分算法进行简要分析和评价.第 2 节给出 DAG 子图的相关定义和重要性质.第 3 节对 AS 算法的原理和步骤进行详细阐述.第 4 节通过实验评价本文算法,并与现有算法进行比较.最后对全文进行总结.

1 相关研究

一个包含 N 个节点的图具有 2^N-1 个非空的节点导出子图,最坏情况下,子图枚举的复杂度是指数的.尽管如此,对具体问题中的特殊子图类型,仍然可能存在快速有效的算法.Wang 在文献[1]中给出一种统一算法解决了简单图的几类典型子图的枚举,这些子图包括独立集、团、路、无弦路、弦回路等.Hu 在文献[2]中提出了连通无向图中的连通子图的枚举算法.因图在电路、图像、网络、工作流等复杂结构建模中的广泛应用,为解决结构数据挖掘问题而提出的频繁子图发现[3]方法也成为当前研究的热点.

不同于上述研究,CI 设计问题中所涉及的图和子图更为特殊.研究者已进行了多种尝试,希望提高这类子图枚举的效率.在以往的研究当中,最为著名的是 Pozzi 等人提出的 PAI 算法[4],该方法采用一种穷举的方式,它利用约束违例的传播性对子图搜索空间中的不可行区域进行剪枝.不幸的是,一方面,PAI 算法的性能会随图规模的增大和输出端口约束的放松而快速恶化;另一方面,它不关心子图的连通性,在用于连通子图的搜索时效率较低.关于在 CI 设计中应该枚举凸连通子图还是凸子图的问题,Yu 等人[5]和 Chen 等人[6]的研究做出了回答:凸连通子图对 CI 设计不仅是有效的,还有助于加快整个设计流程.基于此,我们关注凸连通子图的枚举.

Cong 等人在文献[7]中提出了一种递归算法实现了多输入单输出凸连通子图的提取,在输入端口约束不大于 5 时,该算法性能理想,但它不支持多输出端口子图的枚举.Yu 等人在文献[5]中提出了一种 union 算法,该算法的基本思想是,先找到一系列小的子图,再利用它们合并生成更多的子图.union 在枚举过程中可能产生大量非法和重复的子图,这极大地影响了它的效率.Chen 等人[6]在 union 的基础上进一步进行了优化,但他们的算法仍需要处理大量的图合并与分割运算,当问题规模较大时,这些运算十分耗时.Gutin 等人在文献[8]中给出了一种$\mathcal{A}$算法,不过$\mathcal{A}$不支持对子图端口施加约束.

2 定义与性质

2.1 相关定义

对有向无环图 $G(V,E)$,将其节点导出子图(以下简称子图)记为 $S(V_S,E_S)$,即 $V_S\subseteq V,E_S\subseteq E$ 且 $\forall u,v\in V_S$,若 $(u,v)\in E$ 则必有 $(u,v)\in E_S$.以下 $G(V,E)$简记作 G,$S(V_S,E_S)$简记作 S.下面给出相关的定义.

定义 1(凸子图). 对于 G 的子图 S,$\forall u,v\in V_S$,若在 G 中 u 与 v 之间的任何路径都只经过 S 中的节点,则称 S 是 G 的凸子图(convex subgraph).

定义 2(连通性). 若将 G 的全部有向边替换为无向边而产生的无向图 $\hat{G}$ 是连通的,则称 G 是连通的;若将子图 S 的全部有向边替换为无向边而产生的无向图 $\hat{S}$ 是连通的,则 S 是 G 的连通子图.

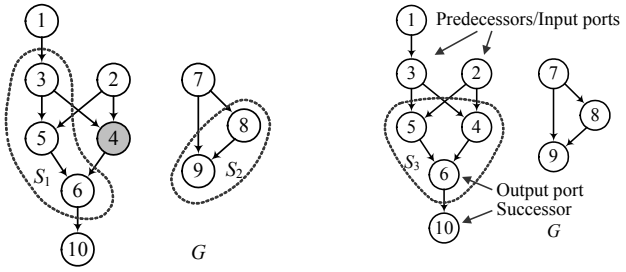
对任意 u,v,若存在路径 $u\to v$,u 即是 v 的前辈节点,v 则是 u 的后代节点.特别当 u,v 相邻时,有如下定义.

定义 3(前驱节点和后继节点). $\forall u,v\in V$,若 $(u,v)\in E$,称 u 是 v 的前驱节点(predecessor),v 是 u 的后继节点(successor);若 $v\in V_S$ 且 $u\notin V_S$,称 u 为 S 的外部前驱节点;若 $u\in V_S$ 且 $v\notin V_S$,则称 v 为 S 的外部后继节点.

节点 v 的前驱节点集记为 $pred(v)$,即 $pred(v)=\{u|u\in V,(u,v)\in E\}$;$v$ 的前驱节点集记为 $succ(v)$,即 $succ(v)=\{u|u\in V,(v,u)\in E\}$.对于子图 S,其外部前驱节点集记作 $pred(S)$,则 $pred(S)=\left(\bigcup_{v\in V_S}pred(v)\right)\setminus V_S$;$S$ 的外部后继节点集记为 $succ(S)$,有 $succ(S)=\left(\bigcup_{v\in V_S}succ(v)\right)\setminus V_S$.

我们将子图的外部前驱节点叫做输入端口(input ports),外部前驱节点集称为输入端口集,记为 $ip(S)$,即 $ip(S)=pred(S)$;与外部后继结点相邻接的子图内节点叫做输出端口(output ports),它们组成的集合称为输出端口集 $op(S)$,即 $op(S)=\{u\mid u\in V_S,v\in succ(S),(u,v)\in E\}=\left(\bigcup_{u\in succ(S)}pred(u)\right)\cap V_S$.

下面用图 2 对上述概念进行示意.图 2(a)中,子图 S_2 是凸连通子图,而 S_1 是非凸子图,原因在于路径 {(3,4),(4,6)}经过了 S_1 外的节点 4;在图 2(b)中,S_3 是凸连通子图,前驱节点集(即输入端口集)为{2,3},后继节点集为{10},输出端口集为{6}.



(a) (b)

Fig.2 Subgraphs of a DAG

图 2 DAG 子图

我们现在给出定制指令设计中的 DAG 凸连通子图枚举的严格定义.

定义 4(具有端口约束的 DAG 凸连通子图枚举). 给定 DAG $G(V,E)$,最大输入与输出端口数 $I_{\max}$ 和 $O_{\max}$,以及禁忌节点集 F,枚举所有满足下列约束条件的子图 $S(V_S,E_S)$:

1) $S(V_S,E_S)$是连通的;
2) $S(V_S,E_S)$是凸的;
3) $|ip(S)|\leqslant I_{\max},|op(S)|\leqslant O_{\max}$;
4) $V_S\cap F=\varnothing$.

在定义 4 中,条件 1)和条件 2)要求子图满足连通性和凸性,条件 3)对子图的输入和输出端口数目进行约束,

条件 4)要求子图不包含禁忌节点.在定制指令设计中,所谓禁忌节点是指从硬件实现或处理器架构等角度出发,定制指令硬件不应包含的特殊原子运算或操作,如存储器访问操作 ld 与 st.

2.2 凸子图的性质

我们给出 DAG 凸子图所特有的一些性质,它们有助于设计高效的算法.

引理 1. 当且仅当 S 满足下列所有条件时,S 是凸子图:

1) $pred(S)\cap succ(S)=\varnothing$;

2) $\forall u'\in succ(S),\forall v'\in pred(S)$,$G$ 内不存在由 u' 到 v' 的路径.

证明:采用反证法.假设 S 满足条件 1)和条件 2),但 S 非凸,必$\exists u,v\in V_S$,它们之间的路径 p 经过 S 外的节点,其中包含$\exists u'\in succ(S),(u,u')\in E$ 以及$\exists v'\in pred(S),(v',v)\in E$.此时存在两种可能:若 $u'=v'$,那么 $pred(S)\cap succ(S)\neq\varnothing$,与条件 1)矛盾;$u'\neq v'$,那么必然存在由 u' 到 v' 的路径 p',与条件 2)矛盾.

因此,当条件 1)和条件 2)都满足时,S 必是凸子图. □

引理 2. 给定凸子图 S 及节点 $r\notin V_S$,S'是由 $V_S\cup\{r\}$导出的子图,若下列条件均成立,则 S'是凸的:

1) $\forall v\in pred(S)$,G 中不存在路径 $r\to v$;

2) $\forall u\in succ(S)$,G 中不存在路径 $u\to r$.

证明:假定条件 1)和条件 2)均成立,首先计算 S'的外部前驱节点集与外部后继节点集:

$$pred(S')=(pred(S)\cup pred(r))\backslash\{r\}=(pred(S)\backslash\{r\})\cup pred(r) \tag{1}$$

$$succ(S')=(succ(S)\cup succ(r))\backslash\{r\}=(succ(S)\backslash\{r\})\cup succ(r) \tag{2}$$

$$pred(S')\cap succ(S')=((pred(S)\cap succ(S))\cup(pred(S)\cap succ(r))\cup(pred(r)\cap succ(S))\cup(pred(r)\cap succ(r)))\backslash\{r\} \tag{3}$$

因 S 是凸的,则有 $pred(S)\cap succ(S)=\varnothing$;因 G 为 DAG,则有 $pred(r)\cap succ(r)=\varnothing$.由条件 1)知 $pred(S)\cap succ(r)=\varnothing$,由条件 2)知 $pred(r)\cap succ(S)=\varnothing$.整理公式(3),得到 $pred(S')\cap succ(S')=\varnothing$.

任取$\forall u\in succ(S)\backslash\{r\},\forall v\in pred(S)\backslash\{r\}$,引理 1 保证不存在路径 $u\to v$;$\forall u\in succ(S)\backslash\{r\},\forall v\in pred(r)$,由条件 2)可知,不存在路径 $u\to v$;取$\forall u\in succ(r),\forall v\in pred(S)\backslash\{r\}$,由条件 1)可知,则不存在路径 $u\to v$;对$\forall u\in succ(r),\forall v\in pred(r)$,不存在路径 $u\to v$.于是,$\forall u\in succ(S'),\forall v\in pred(S')$,不存在由 u 到 v 的路径.

综上所述,根据引理 1,S'也是凸子图,引理 2 得证. □

下面通过图 3 对引理 2 进行简单示意.在图 3(a)中,S_1 为凸子图,$pred(S_1)=\{1,3\}$,节点 3 满足引理 2 条件,则由 S_1 和节点 3 合并生成的子图 S_2 为凸子图;由于存在路径 1→3,节点 1 不满足引理 2 的条件 1),由 S_1 和节点 1 合成的子图 S_3 非凸.图 3(b)中,S_4 为凸子图,$pred(S_4)=\{2,4\}$,节点 2 符合引理 2 条件要求,由 S_4 和节点 2 合成的 S_5 为凸子图;由于路径 2→4 的存在,节点 4 不满足引理 2 的条件 2),于是 S_6 为非凸子图.

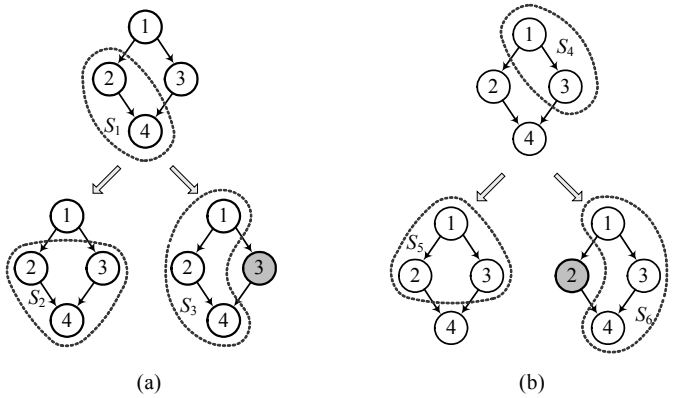


Fig.3 Inheritance of convexity

图 3 凸性的继承

引理 3. 给定凸子图 S 及节点 $r\notin V_S$,S'是由 $V_S\cup\{r\}$导出的子图,以下结论成立:

1) 若 $r\notin succ(S)$,则 $op(S)\subseteq op(S')$;

2) 若 $r\notin pred(S)$,则 $ip(S)\subseteq ip(S')$.

证明:首先证明结论 1).假设 $r\notin succ(S)$,已知

$$op(S)=\left(\bigcup_{u\in succ(S)} pred(u)\right)\cap V_S \tag{4}$$

计算

$$op(S')=\left(\bigcup_{u\in succ(S')} pred(u)\right)\cap V_{S'} \tag{5}$$

根据公式(2)及条件 $r\notin succ(S)$,进一步整理公式(5)得

$$op(S')=\left(\bigcup_{u\in succ(S)\cup succ(r)} pred(u)\right)\cap (V_S\cup\{r\}) \tag{6}$$

对比公式(4)和公式(6),可得 $op(S)\subseteq op(S')$.

现在证明结论 2).假设 $r\notin pred(S)$,根据公式(1)可得 $pred(S')=pred(S)\cup pred(r)\supseteq pred(S)$,实际上,$ip(S)=pred(S)$,$ip(S')=pred(S')$,因此 $ip(S)\subseteq ip(S')$. □

3 AS 算法

满足凸性、连通性及端口约束的子图只是 DAG 全体子图集合中的很小一部分,算法可以采用一些技巧尽可能地使搜索过程遇到的子图首先满足凸性和连通性的要求.尽管如此,以往算法还是不能避免在搜索过程中时常进行凸性和连通性的检测,这些检测大都基于深度优先搜索,其复杂度为 $O(|V_S|+(|E_S|)$.我们的做法则是,首先获得一个凸连通子图(最简单的情况即单一节点),将该子图作为出发点,通过对其进行扩展得到更多的凸连通子图.为保证连通性,扩展采用向子图添加其邻接节点的方式;为维持凸性,添加的邻接节点必须满足引理 2 的条件要求.由于算法核心基于邻接节点的搜索,我们将其命名为 Adjacent Search(AS).子图扩展如图 4 所示,从节点 v 开始,通过不断向外围"生长"得到更多子图.对 DAG 而言,扩展分为前向和后向两类,图 4 中的 step 1~step 3 为前向扩展,step 4~step 6 为后向扩展.

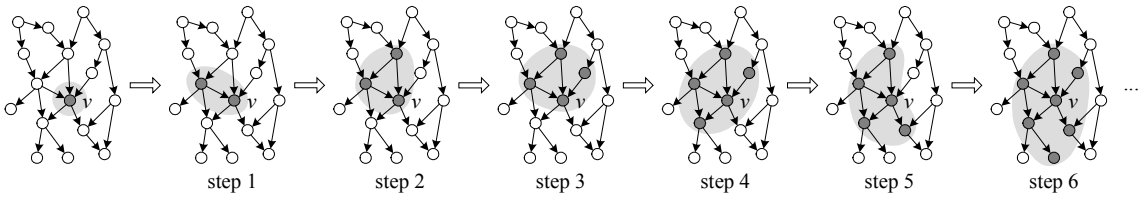


Fig.4 Growth of subgraph

图 4 子图的生长

AS 算法必须支持对端口约束的处理.根据引理 3,端口数违例时必须对扩展方向进行控制,我们将其称为端口修复.若子图的输入端口数违例,只有继续前向扩展才有可能得到输入端口数合法的新子图;相反,仍进行后向扩展则只可能增大新子图的输入端口数.输出端口数违例的情况,处理方式则正好相反.当两个方向的端口数都违例时,应优先选定一个方向修复,待其合法后再修复另一个方向.若修复过程中无法继续向该方向扩展,即该方向已没有可用的邻接节点,则修复过程结束.

AS 算法的入口为 *ccsg_enum*(),形参 G 为目标 DAG,*CCSG_SET* 为合法子图集合.算法 1 第 1 行对 G 的节点进行拓扑排序,保证若存在路径 $u\to v$,则有 $u<v$.搜索过程中,节点有 3 种可能状态:未访问(UNVISIT)、属于子图(SELECT)和被丢弃(DROP),用向量 R 进行记录.扩展过程只考虑未访问的节点,该节点有两种处理对策:被加入子图或丢弃.在第 2 行即对 R 进行初始化时,所有节点均被置为未访问.第 3 行~第 5 行按照逆序调用 *AdjacentSearch*()搜索包含 v 的合法子图,此时,*AdjacentSearch*()只能在{1,2,…,v}上进行搜索,这样可以避免从不同节点出发的扩展过程产生重复的子图.

算法 1. *ccsg_enum*(*G*,*CCSG_SET*).

1. *topological_sort*(*G*);
2. **for each** *v*=1 to *n* **do** *R*[*v*]=UNVISIT;
3. **for each** *v*=*n* to 1 **do**
4. *AdjacentSearch*(*R*,*v*,*v*,WELL,*CCSG_SET*);
5. *R*[*v*]=DROP;

递归函数 *AdjacentSearch*()以 *v* 为起点,通过扩展方式在 *G* 内搜索合法子图.形参列表中,$v_{current}$ 表示当前处理的节点,*v* 表示扩展起点,*status* 记录了当前子图的端口状态.端口状态共有 4 种:合法(WELL)、输入违例(IP_EXCEED)、输出违例(OP_EXCEED)以及双向违例(IP_EXCEED|OP_EXCEED).算法 2 的第 1 行~第 6 行,若 $v_{current}$ 不是禁忌节点,则将其加入子图,并检查新的子图是否满足端口约束.函数 *IP*()和 *OP*()分别用来提取新子图的输入和输出端口,$I_{\max}$ 和 $O_{\max}$ 则分别为输入和输出端口数约束常数.经过第 4 行、第 5 行的检验,若合法,则利用 *build_ccsg*()将 *R* 转化为图数据结构并存入 *CCSG_SET*.第 7 行~第 9 行尝试进一步扩展,首先调用 *dead*()检查 *IP*(*R*)和 *OP*(*R*)中确定无法修复的节点数,即 *IP*(*R*)中状态为 DROP 的节点数及 *OP*(*R*)中与状态为 DROP 的外部后继结点相邻接的节点数,若它们均未超过约束则继续探索;在第 8 行,*find_next*()根据 $status_{current}$ 寻找下一个可行邻接节点 v_{next},若它存在,则将其作为扩展点递归调用 *AdjacentSearch*().第 10 行~第 13 行考虑将 $v_{current}$ 排除在子图之外的情况.此时,不仅 $v_{current}$ 自身将被置为 DROP,与它相关的一些节点也将被直接丢弃.当 $v_{current}$ 来自于子图的前驱节点集时,所有 $v_{current}$ 的前辈节点可直接丢弃;当 $v_{current}$ 来自于的后继节点集时,$v_{current}$ 的后代节点可直接丢弃.这个工作由函数 *drop*()实现.第 12 行~第 14 行,在丢弃 $v_{current}$ 及相关节点后,根据 *dead*()检查的结果决定是否继续进行搜索.

算法 2. *AdjacentSearch*(*R*,$v_{current}$,*v*,*status*,*CCSG_SET*).

1. **if** $v_{current}\notin F$ **then**
2. $R[v_{current}]$=SELECT;
3. $status_{current}$=WELL;
4. **if** $|IP(R)|>I_{\max}$ **then** $status_{current}$=IP_EXCEED;
5. **if** $|OP(R)|>O_{\max}$ **then** $status_{current}=status_{current}$|OP_EXCEED;
6. **if** $status_{current}$==WELL **then** *insert*(*CCSG_SET*,*build_ccsg*(*R*));
7. **if** $dead(IP(R))\leqslant I_{\max}$ && $dead(OP(R))\leqslant O_{\max}$ **then**
8. $v_{next}=find\_next(R,status_{current})$;
9. **if** v_{next}!=NULL **then** *AdjacentSearch*(*R*,v_{next},*v*,$status_{current}$,*CCSG_SET*);
10. **if** $v_{current}$!=*v* **then**
11. *drop*(*R*,$v_{current}$);
12. **if** $dead(IP(R))\leqslant I_{\max}$ && $dead(OP(R))\leqslant O_{\max}$ **then**
13. $v_{next}=find\_next(R,status)$;
14. **if** v_{next}!=NULL **then** *AdjacentSearch*(*R*,v_{next},*v*,*status*,*CCSG_SET*);

find_next()为子图扩展选择最优的邻接节点.函数首先寻找前驱节点集中拓扑编号最大的未访问节点 *p* 和后继节点集中拓扑编号最小的未访问节点 *s*.算法 3 的第 1 行、第 2 行,寻找 *p* 和 *s* 的工作分别由 *max_unvisit*()和 *min_unvisit*()实现.第 3 行~第 11 行,根据 *status* 状态字选择 *p* 和 *s* 中最合适的一个,选取规则为:子图仅输入违例则输出 *p*;仅输出违例则输出 *s*;合法则优先选择输出 *p*,若 *p* 不存在再输出 *s*;双向违例时,若 *p* 和 *s* 同时存在则输出 *p*,否则输出 NULL 终止修复.

算法 3. *find_next*(*R*,*status*).

1. *p*=*max_unvisit*(*pred*(*R*));
2. *s*=*min_unvisit*(*succ*(*R*));
3. *switch*(*status*)

```
4.    case IP_EXCEED: return(p);
5.    case OP_EXCEED: return(s);
6.    case WELL:
7.      if p==NULL then return(s);
8.      else return(p);
9.    otherwise:
10.     if p==NULL||s==NULL then return (NULL);
11.     else return(p);
```

下面对算法复杂度做简单分析.每次迭代时,首先要在本地复制 R 的副本,其工作量为 $O(|V|)$;调用 $IP()$和 $OP()$,它们的复杂度为 $O(|V|)$;函数 $dead()$因在输入和输出节点集上进行遍历,复杂度为 $O(|V|)$;函数 $drop()$需要查找被丢弃节点的前辈和后代,该工作通过访问传递闭包 G^{TC} 的邻接矩阵实现,其复杂度为 $O(|V|)$;$find_next()$遍历子图邻接节点,工作量也是 $O(|V|)$.总体上看,$AdjacentSearch()$单次迭代的复杂度上界为 $O(|V|)$.另外,G 的密度也会影响运算工作量,密度越大,则子图的端口和邻接节点可能越多,这将增大 $dead()$和 $find_next()$的工作量,每次迭代的运算时间也随之增长.

4 实验结果与分析

下面通过实验对 AS 算法进行测试和评估.所有实验都在一台配有 1.66GHz Intel T2300E 处理器和 2GB 内存的 debian 机器上进行.AS 算法采用 C++语言编写,经 GCC4.2 编译.实验分为两部分:第 1 部分利用随机图探索子图数量及算法性能与 DAG 规模和结构以及 I/O 约束的关系;第 2 部分将 AS 算法应用于 CI 设计中的数据流图探索,并与 PAI 算法进行性能对比.

4.1 随机图测试

我们利用随机图探索子图的数量和 AS 算法的性能与$|V|$,$|E|$及 I/O 约束的关系.测试 DAG 来自于 BGL[9]随机图生成器,对每组$|V|$,$|E|$取值都生成 100 个随机图进行实验,最后对结果求取均值.

首先将$|V|$固定为 25,观察$|E|$取不同值时的运行情况.实验结果如图 5 所示,图中横轴为$|E|/|V|$,它既可显示$|E|$的取值变化,也能反映$|E|$与$|V|$的相对关系;图例中,All 表示无 I/O 约束子图,4-2,6-3,8-4 则分别表示满足相应 I/O 约束的合法子图.图 5(a)显示了子图数量与$|E|/|V|$的关系.当$|E|/|V|<1$ 时,随$|E|$增大子图的数量快速增长,这是由于 DAG 自身连通性随$|E|$的增大而改善,导致连通子图的数量激增;此后,随$|E|$的继续增大,子图数量反而下降,原因在于节点间路径增多导致凸子图数量迅速下降.图 5(b)显示了算法时间消耗与$|E|/|V|$的关系,与图 5(a)对比发现,运行时间与子图数量存在较强相关性.观察图 5(c)发现,每获得一个子图的平均时间代价与$|E|/|V|$呈近似线性.另外,图 5(a)和图 5(b)显示,I/O 约束越强则子图数量越少,时间消耗也越低;图 5(c)则显示,I/O 约束越强平均时间消耗越大,这是由于 I/O 约束越强需要排除的非法子图也就相对越多.

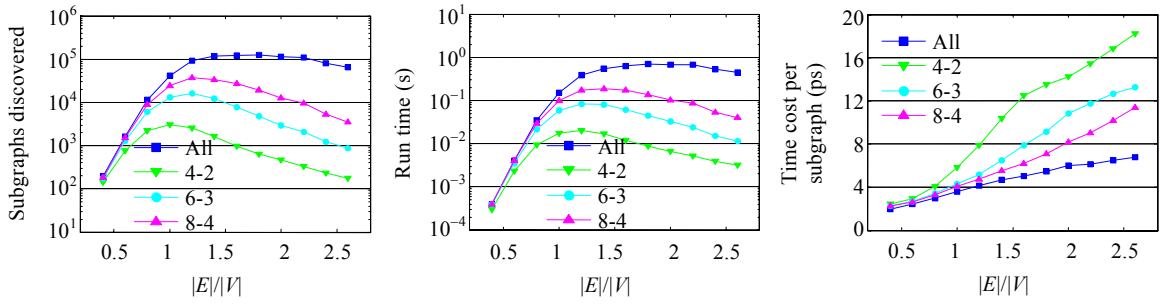


Fig.5 Relationship of convex connected subgraph enumeration and $|E|$

图 5 凸连通子图枚举与$|E|$的关系

接下来将比值|E|/|V|固定为 2,考察|V|对算法的影响,实验结果如图 6 所示.子图数量与|V|的关系通过图 6(a)得到反映,不论何种 I/O 约束,子图数量都随|V|的增大而快速增长.图 6(b)显示,AS 算法的运行时间同样随|V|的增大而迅速上升.图 6(c)显示,平均时间代价也随节点数|V|缓慢增长;特别当枚举无 I/O 约束的凸连通子图时,平均时间代价与|V|呈近似线性关系.

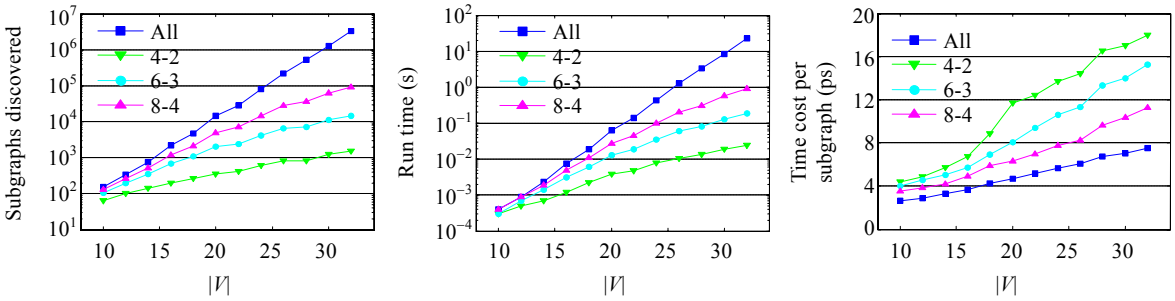


Fig.6 Relationship of convex connected subgraph enumeration and |V|

图 6 凸连通子图枚举与|V|的关系

4.2 AS应用于数据流图探索

我们从测试基准集 MiBench[10]和 MediaBench[11]中选取一些应用程序作为定制指令加速的目标,这些程序首先通过 LLVM[12]C/C++前端转化为中间表达形式(intermediate representation,简称 IR),而后从基本块中提取 DFG.根据这些 DFG 的大小,将实验分为两组.

常规 DFG 节点数均小于 100 它们来自于应用程序 g721enc,adpcmcaudio,h263dec 和 mpeg2enc.我们将著名的 PAI 算法作为参考目标.表 1 列出了典型端口约束下的子图枚举结果.其中的|V|和|V\F|分别表示 DFG 的节点数和其中的非禁忌节点数,I_{max} 和 O_{max} 分别表示输入和输出约束.表中列出了满足不同约束的子图数量和两种算法分别消耗的迭代次数,加速比为 PAI 与 AS 运行时间的比值.数据显示,AS 在各种情况下消耗的迭代次数均小于 PAI,尤其对于较宽松的 I/O 约束,两者的差距十分显著.在运行时间方面,AS 往往百倍千倍地快于 PAI.

Table 1 Results of subgraph enumerations for common DFGs

表 1 对一般规模 DFG 进行子图枚举的结果

G	|V|	|V\F|	Unconstrained subgraphs	I_{max}	O_{max}	Valid subgraphs	Iterations		Time/(s)	Speedup
							PAI	AS	AS	
1	32	19	113	4	2	48	4 226	82	0.000 5	25.6
				6	3	82	16 890	103	0.000 6	144.2
				8	4	94	41 956	107	0.000 6	323.1
2	39	36	14 776	4	2	63	22 508	264	0.002 2	33.5
				6	3	296	98 114	850	0.004 7	89.2
				8	4	1 774	293 402	2 972	0.017 7	100.2
3	48	23	200 737	4	2	33	4 614	169	0.001 4	9.0
				6	3	181	24 934	546	0.004 1	24.2
				8	4	533	101 646	1 304	0.010 8	43.3
4	60	41	17 165	4	2	121	1 584 530	1 174	0.006 1	1 355.0
				6	3	396	4 507 968	4 483	0.023 3	1 098.9
				8	4	1 404	9 176 584	9 916	0.055 5	963.9
5	73	36	5 024	4	2	63	37 860	158	0.001 7	82.7
				6	3	191	419 434	310	0.002 9	719.6
				8	4	354	3 216 524	513	0.004 7	4 144.2
6	84	37	1 329	4	2	92	73 050	390	0.005 3	62.4
				6	3	358	497 978	682	0.005 7	478.8
				8	4	828	1 946 332	988	0.007 6	1 635.6

选取表 1 中的 G_2,图 7 显示了两种算法在不同 I/O 约束下平均每获得一个合法子图所消耗的迭代次数.从

图中可以看出,AS 算法的平均迭代次数总体上远低于 PAI,两者的平均迭代次数都随 I/O 约束的放松而呈下降趋势.相对来说,PAI 算法的图线下降速度较慢,AS 算法的图线下降速度则明显快得多.因此,I/O 约束越宽松(数值越大),AS 越显得高效;同时,由于 I/O 约束的放松导致合法子图数量的激增,此时 AS 算法显现出的优势就更加明显.

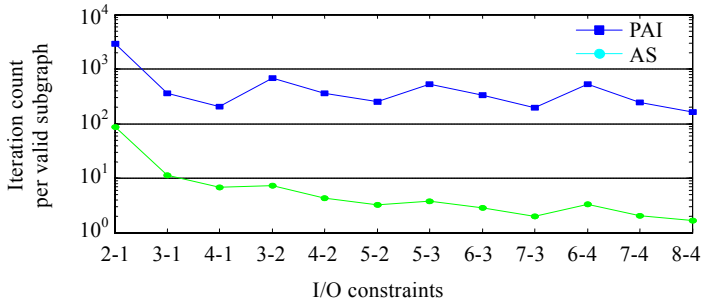


Fig.7 Iteration count per valid subgraph

图 7 平均迭代次数

当 DFG 节点数大于 100 时,PAI 由于耗时太长(>1 小时)而失效.此时检验 AS 算法对较大规模 DFG 的处理能力,实验的结果列于表 2.测试 DFG 来自于 mpeg2dec,djpeg 和 des.多数情况下,AS 都能在较短时间内完成枚举,但在对 G_3 进行处理时消耗了较长的时间.AS 实质上并不是多项式算法,当图的规模增大且结构相对复杂时,算法的性能也随之恶化.值得庆幸的是,在大多数嵌入式应用中,基本块的 DFG 规模往往较小.节点数超过 300 的 DFG 往往是由于程序设计者采用了循环展开的方式书写代码而造成的,对于这类代码,可将其先恢复为循环形式,再从循环内提取基本块,这样得到的 DFG 适宜采用 AS 进行探索.

Table 2 Results of subgraph enumerations for large DFGs

表 2 对较大规模 DFG 进行子图枚举的结果

G	$\|V\|$	$\|V\backslash F\|$	I_{max}	O_{max}	Valid subgraph	Iterations	Time/(s)
1	107	79	4	2	182	3 276	0.064 4
			6	3	941	8 285	0.745 5
			8	4	6 524	29 646	8.324 1
2	136	85	4	2	280	1 341	0.014 0
			6	3	2 965	14 680	0.122 2
			8	4	26 182	481 157	5.113 5
3	137	133	4	2	456	288 792	4.238 4
			6	3	5 406	12 403 176	259.059 2
			8	4	67 305	163 012 933	3 724.600 7
4	213	158	4	2	353	5 410	0.041 5
			6	3	2 816	27 572	0.287 5
			8	4	40 976	308 651	4.280 8

5 结 论

本文从 CI 设计中的数据流图探索问题出发,提出一种新型的 DAG 凸连通子图枚举算法——AS.通过利用凸子图在凸性和端口方面的特性,AS 算法有效地规避了探索空间中的不可行区域,提高了枚举效率.对测试基准集 MiBench 和 MediaBench 的实验数据显示,AS 效率较 PAI 有显著的提高,其加速比可达 10~10^3X.

References:

[1] Wang JS. A general algorithm for enumerating some subgraphs of a simple graph. Chinese Journal of Computers, 1986,1(1):37−43 (in Chinese with English abstract).

[2] Hu GP. An algorithm of seeking all subgraphs of a connected simple undirected graph and its application. Computer Engineering, 2003,29(13):101−102 (in Chinese with English abstract).

[3] Li XT, Li JZ, Gao H. An efficient frequent subgraph mining algorithm. Journal of Software, 2007,18(10):2469−2480 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/2469.htm [doi: 10.1360/jos182469]

[4] Pozzi L, Atasu K, Ienne P. Exact and approximate algorithms for the extension of embedded processor instruction sets. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2006,25(7):1209−1229. [doi: 10.1109/TCAD.2005.855950]

[5] Yu P, Mitra T. Scalable custom instructions identification for instruction-set extensible processors. In: Proc. of the Int'l Conf. on CASES 2004. 2004. 69−78.

[6] Chen X, Maskell DL, Sun Y. Fast identification of custom instructions for extensible processors. IEEE Trans. on Computer-Aided Design Integrated Circuits and Systems, 2007,26(2):359−368. [doi: 10.1109/TCAD.2006.883915]

[7] Cong J, Fan YP, Han GL, Zhang ZR. Application-Specific instruction generation for configurable processor architectures. In: Proc. of the 2004 ACM/SIGDA 12th Int'l Symp. on Field Programmable Gate Arrays. New York, 2004. 183−189.

[8] Gutin G, Johnstone A, Reddington J, Scott E, Soleimanfallah A, Yeo A. An algorithm for finding connected convex subgraphs of an acyclic digraph. In: Proc. of the Algorithms and Complexity in Durham. 2007.

[9] Siek J, Lee LQ. The boost graph library. http://www.boost.org/doc/libs/release/libs/graph

[10] Guthaus MR, Ringenberg JS, Ernst D, Austin TM, Mudge T, Brown RB. MiBench: A free, commercially representative embedded benchmark suite. In: Proc. of the 4th IEEE Annual Workshop Workload Characterization. 2001. 3−14.

[11] Lee C, Potkonjak M, Mangione-Smith WH. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems. In: Proc. of the 30th Annual IEEE/ACM Int'l Symp. on Microarchitecture. Research Triangle Park, 1997. 330−335.

[12] Lattner C, Adve V. LLVM: A compilation framework for lifelong program analysis & transformation. In: Proc. of the 2004 Int'l Symp. on Code Generation and Optimization. 2004. 75−86.

附中文参考文献:

[1] 王介生.简单图中若干子图枚举问题的统一算法.计算机学报,1986,1(1):37−43.

[2] 胡广朋.求解连通无向简图的所有连通子图的算法及其应用.计算机工程,2003,29(13):101−102.

[3] 李先通,李建中,高宏.一种高效频繁子图挖掘算法.软件学报,2007,18(10):2469−2480. http://www.jos.org.cn/1000-9825/18/2469.htm [doi: 10.1360/jos182469]

薄拾(1982−),男,陕西咸阳人,博士生,主要研究领域为专用指令集设计,可重配置计算.

林孝康(1947−),男,教授,博士生导师,主要研究领域为通信网,ATM,数字集群系统,数字对讲机.

葛宁(1971−),男,博士,教授,博士生导师,主要研究领域为通信领域片上系统,面向通信系统高速信号处理的芯片设计,超宽带传输与组网技术,认知无线电与协同通信,多业务光传送平台与网络.