计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
2026,35(1):197−208 [doi: 10.15888/j.cnki.csa.010053] [CSTR: 32024.14.csa.010053]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

精确缓存分类分析 NP 特征消减技术①

邢海峰, 王　彪, 高宽云

(内蒙古财经大学 计算机信息管理学院, 呼和浩特 010070)
通信作者: 邢海峰, E-mail: hhhtxinghaifeng@163.com

摘　要: 现代嵌入式系统普遍配置高速缓存, 硬件性能提升的同时系统软件研制变得复杂, 需预测实时任务高速缓存行为. 预测高速缓存行为常用到缓存分类, 实施精确缓存分类可能呈现 NP (non-deterministic polynomial) 特征, 消减精确缓存分类可能呈现的 NP 特征是研究难点. 针对先前工作不足, 本文提出强连通分量消除技术和扩展的反链技术来进一步消减精确缓存分类可能呈现的 NP 特征. 通过测试基准集中程序可知, 提出技术可使大部分分类时间开销下降, 且最大降量超 4 h; 小部分分类时间开销略有上升, 且最大升量不超 3 min. 这对设计高效缓存行为预测工具有帮助.

关键词: 嵌入式系统; 缓存分类分析; 时间开销; 强连通分量消除技术; 扩展的反链技术

引用格式: 邢海峰,王彪,高宽云.精确缓存分类分析 NP 特征消减技术.计算机系统应用,2026,35(1):197–208. http://www.c-s-a.org.cn/1003-3254/10053.html

NP Characteristic Mitigation Techniques in Exact Cache Classification Analysis

XING Hai-Feng, WANG Biao, GAO Kuan-Yun

(School of Computer Information Management, Inner Mongolia University of Finance and Economics, Hohhot 010070, China)

Abstract: Modern embedded systems are typically equipped with high-speed caches, which enhance hardware performance but complicate system software design, necessitating the prediction of cache behavior for real-time tasks. Predicting cache behavior often involves cache classification, and implementing an exact cache classification may exhibit non-deterministic polynomial (NP) characteristics. Reducing these NP characteristics in an exact cache classification presents a challenge. To address previous shortcomings, this study proposes both the strongly connected component elimination technique and the extended anti-chain technique to further reduce the NP characteristics. Through benchmark tests, it is found that the proposed techniques can significantly reduce most classification time overhead, with the maximum reduction exceeding 4 h; while a small portion of classification time overhead slightly increases, with the maximum increase not exceeding 3 min. It helps design more efficient cache behavior prediction tools.

Key words: embedded system; cache classification analysis; time cost; strongly connected component elimination technique; extended anti-chain technique

20 世纪 70 年代起, 实时嵌入式系统就被广泛应用于工业自动化控制、国防及航天领域, 在物联网、云计算、边缘计算等新型计算范式出现后, 其应用在社会各个领域得到延伸, 相关研究得到更广泛关注[1,2]. 随着嵌入式处理器运算速度不断提升, 嵌入式系统中的存储墙现象日趋凸显, 为嵌入式处理器配置高速缓存成为业界共识[3]. 嵌入式系统的系统软件研制需估计实时任务执行时间, 高速缓存的配置使得估计过程必须

① 基金项目: 内蒙古自治区自然科学基金青年项目 (2022QN06001)
收稿时间: 2025-06-17; 修改时间: 2025-08-01; 采用时间: 2025-08-15; csa 在线出版时间: 2025-12-01
CNKI 网络首发时间: 2025-12-02

© 中国科学院软件研究所 http://www.c-s-a.org.cn

包含高速缓存缺失命中行为预测, 进而使得系统软件研制变得复杂[4]. 高速缓存缺失命中行为预测常用到缓存分类分析[5], 该分析实施在一个抽象于实时任务程序的控制流图 (control flow graph, CFG) 上, 通过在控制流图的各程序控制点处迭代由计算程序内存块构成的缓存实例, 来识别程序内存块是否于程序控制点处驻留高速缓存, 进而实现缓存总是命中或总是缺失的分类. 随着程序结构变复杂, 控制流图程序控制点处待计算的缓存实例数量可能呈现指数级规模, 精确缓存分类分析的时间开销可能呈现 NP (non-deterministic polynomial) 特征[6]. 最近提出的由 3 个阶段分析构成的精确缓存分类研究[7], 在第 3 阶段分析中引入反链技术, 以期消减 NP 特征. 可反链技术有待进一步深入研究, 且控制流图中可能存在的强连通分量, 都阻碍着 NP 特征的进一步消减. 为了尽可能消减 NP 特征, 助力高效缓存行为预测工具设计, 本文提出使用强连通分量消除技术和扩展的反链技术来重构第 3 阶段分析, 并在升级版 Chronos[8]上实现.

1 缓存分类分析回顾

控制流图 (CFG) 是一个抽象于实时任务程序的有向图, 其中的顶点表示程序控制点, 指示一个程序内存块被访问的开始点或结束点; 有向边指示程序控制点间的可达性, 边上的备注表示可达控制点间待访问的程序内存块.

缓存语义 (cache semantic, CS) 是一个封装了实时任务运行期间高速缓存所装载内容及内容更新操作的抽象语义, 其中所装载内容为程序内存块所构成的序集, 该序集称为缓存实例; 内容更新操作抽象了依据缓存替换策略把程序内存块更新到缓存实例的过程.

数据流分析 (data flow analysis, DFA)[9]是一个通过正向或逆向扫描 CFG 来计算或搜集特定信息的分析. 如在缓存分类中, 该分析扫描 CFG 每一条边时均会调用 CS 内容更新操作来将边上备注的程序内存块更新到边的开始点处的缓存实例中, 从而计算出边的结束点处的缓存实例. 由于被视为边开始点 (结束点) 的程序控制点可能是多条边的结束点 (开始点), 因此任意程序控制点处计算的缓存实例会构成一个集合, 称为缓存实例集. 依据固定点理论 (fixed point theory)[10], CFG 中各个程序控制点处、用于缓存分类的缓存实例集可经多轮 DFA 求得.

早期研究把数据流分析引入到直接映射[11,12]及集合关联映射[13,14]缓存分类分析中, 实现缓存总是命中 (always hit, AH)、总是缺失 (always miss, AM) 以及首次缺失 (first miss) 的分类. 经典的 Must-May 方法[15,16]于同时期被提出, 并被用于后期的精确缓存分类[7,8,17]以及缓存分类分析的模块化设计[18], 并对本文不涉及的缓存持久分析[19]起推动作用.

Must 旨在识别 AH 程序内存块, 因过高估计程序内存块在 LRU (least recently used policy) 缓存中的缓存年龄而把部分 AH 内存块归类为 NC (not certain). May 旨在识别 AM 程序内存块, 因过低估计内存块缓存年龄而把部分 AM 内存块归类为 NC. 后期研究[8]在 Must-May 分类基础上, 采用模型检测 (model checking, MC) 对不确定的 NC 做进一步的分析, 并把 Must-May+MC 构成的两阶段分析引入原预测工具 Chronos 中[20].

鉴于后期研究在分类精确性和时间开销两方面均有提升空间[8], 近期研究[7,17]均采用三阶段分析实现了精确缓存分类: 第 1 阶段通过 Must-May 对程序内存块进行初步分类; 第 2 阶段通过存在命中 (∃hit) 和存在缺失 (∃miss) 方法把 NC 分类为∃hit、∃miss 及确定 NC (definitely NC); 第 3 阶段, 早期研究[17]采用 MC 分别于∃hit 和∃miss 程序内存块中识别出 AH 和 AM. 后续研究[7]采用集包含关系反链技术支持的方法分别于∃hit 和∃miss 程序内存块中识别出 AH 和 AM. 两种精确缓存分类分析[7,17]相比, 反链技术的应用有助于 NP 特征的消减, 可反链技术的进一步扩展和 CFG 中强连通分量的消除均有利于 NP 特征的进一步消减.

2 强连通分量消除技术

强连通分量是有向图中的特殊分量[21], 其上任意两顶点可互达. 当精确缓存分类分析[7]的第 3 阶段实施在一个包含强连通分量的子 CFG (CFG 的子图) 上时, 不论在∃hit 程序内存块中识别 AH 还是在∃miss 内存块中识别 AM, 子 CFG 程序控制点处缓存实例集的规模均可能是指数级, DFA 迭代计算的时间开销均可能具备 NP 特征. 下面先分析强连通分量可能导致的 NP 特征.

2.1 强连通分量 NP 特征分析

假设 LRU 集合关联高速缓存的关联度为 k, 即任一缓存集至多可装载 k 个程序内存块, 不同缓存集上的更新彼此独立. 为了便于分析, 仅抽象一个缓存集上缓存实例, 且映射到该缓存集上的程序内存块所构成

© 中国科学院软件研究所 http://www.c-s-a.org.cn

的集合为 $M=\{m_i|i\in[1,|M|],|M|>k\}$, $|M|$是集合的基; 抽象的缓存实例是以程序内存块缓存年龄为序的序集且基不超过 k, 在此简化表示为集合$\{m_i|m_i\in M\}$.

● 分析 1: 于∃hit 程序内存块中识别 AH 时, 包含强连通分量的子 CFG 上可能存在的 NP 特征. 假设待进一步识别的∃hit 程序内存块是 m_k, 其所处子 CFG 见图 1(a), 控制点 1 处缓存实例集表示为{{}}.

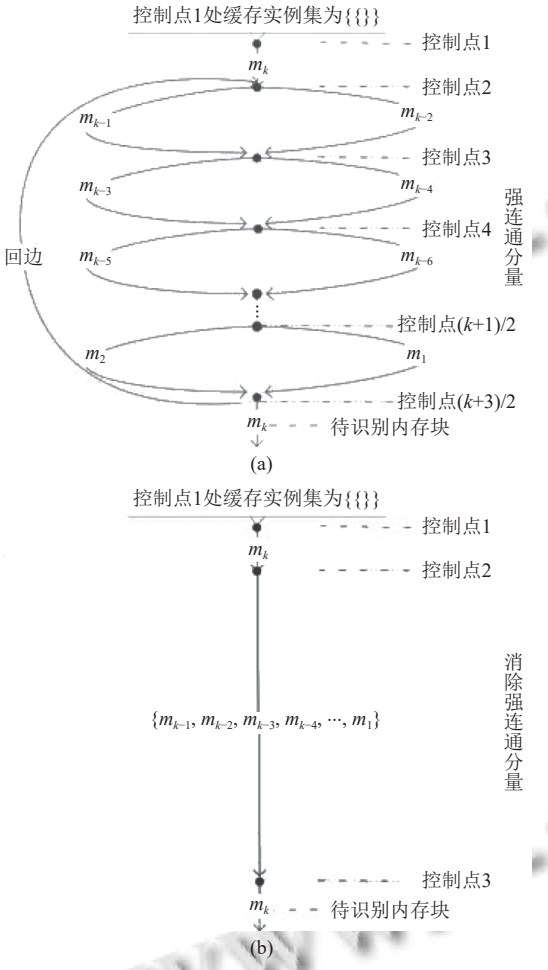

图 1 ∃hit 程序内存块所处子 CFG

以控制点 1 为起点, DFA 正向扫描子 CFG 并逐边计算其余控制点处缓存实例集, 并于各控制点处应用集包含关系反链技术[7]来消减所计算出的缓存实例集规模. 具体消减规则如下.

消减规则 1: 如缓存实例集中元素存在包含关系, 那么被包含者出局.

第 1 轮迭代中, 任意控制点处的缓存实例集均不符合消减规则 1, 因此导致计算的缓存实例共计 $2^{(k+1)/2}$. 第 2 轮迭代中, 任意控制点处缓存实例集中部分元素间符合消减规则 1, 因此随着控制点编号的递增而逐步计算出的缓存实例集基数逐渐减少为 1, 计算的缓存实例共计 $2^{(k+1)/2}$. 由于第 2 轮迭代于控制点 $(k+3)/2$ 处计算出了包含子 CFG 中全部程序内存块的缓存实例$\{\{m_i|i\in[1,k]\}\}$, 因此第 3 轮迭代中各控制点处的缓存实例集均完全符合消减规则 1, 缓存实例集的基数均被消减为 1, 缓存实例共计 $(k+3)/2$. 综合 3 轮迭代, 缓存实例合计 $2^{(k+3)/2}+(k+3)/2$. 3 轮 DFA 迭代结果见表 1.

表 1 图 1(a) 子 CFG 上 DFA 分析

轮次	控制点	缓存实例集	基数
第 1 轮	1	{{}}	1
	2	$\{\{m_k\}\}$	1
	3	$\{\{m_{k-1}\vee m_{k-2}, m_k\}\}$	2
	4	$\{\{m_{k-3}\vee m_{k-4}, m_{k-1}\vee m_{k-2}, m_k\}\}$	4
	…	…	…
	$(k+1)/2$	$\{\{m_3\vee m_4,\cdots, m_{k-3}\vee m_{k-4}, m_{k-1}\vee m_{k-2}, m_k\}\}$	$2^{(k-3)/2}$
	$(k+3)/2$	$\{\{m_1\vee m_2,\cdots, m_{k-3}\vee m_{k-4}, m_{k-1}\vee m_{k-2}, m_k\}\}$	$2^{(k-1)/2}$
第 2 轮	1	{{}}	1
	2	$\{\{m_1\vee m_2,\cdots, m_{k-3}\vee m_{k-4}, m_{k-1}\vee m_{k-2}, m_k\}\}$	$2^{(k-1)/2}$
	3	$\{\{m_{k-1}, m_{k-2}, m_1\vee m_2,\cdots, m_{k-3}\vee m_{k-4}, m_k\}\}$	$2^{(k-3)/2}$
	4	$\{\{m_{k-3}, m_{k-4}, m_{k-1}, m_{k-2}, m_1\vee m_2,\cdots, m_k\}\}$	$2^{(k-5)/2}$
	…	…	…
	$(k+1)/2$	$\{\{m_3, m_4,\cdots, m_{k-1}, m_{k-2}, m_1\vee m_2, m_k\}\}$	2
	$(k+3)/2$	$\{\{m_i|i\in[1,k]\}\}$	1
第 3 轮	1	{{}}	1
	2	$\{\{m_i|i\in[1,k]\}\}$	1
	3	$\{\{m_i|i\in[1,k]\}\}$	1
	4	$\{\{m_i|i\in[1,k]\}\}$	1
	…	…	…
	$(k+1)/2$	$\{\{m_i|i\in[1,k]\}\}$	1
	$(k+3)/2$	$\{\{m_i|i\in[1,k]\}\}$	1

注: 表中缓存实例集采用压缩表示法, 其中的∨表示或关系, 取两者之一, 例如$\{\{m_{k-1}, m_k\}, \{m_{k-2}, m_k\}\}$可压缩表示为$\{\{m_{k-1}\vee m_{k-2}\}, m_k\}$

很明显, 精确识别内存块 m_k 呈现 NP 特征. 虽然集包含关系反链技术[7]消减规则 1 在消减缓存实例集中起到了一定作用, 可强连通分量的存在仍然导致 NP 特征的存在. 如果图 1(a) 所示子 CFG 可被转化为图 1(b) 所示子 CFG, 即图 1(a) 中强连通分量被合并为一条从控制点 2 到控制点 3 的边, 边上备注为强连通分量所含内存块的集, 那么图 1(b) 所示子 CFG 上的 DFA 分析将使指数级缓存实例数量降为常量 3, 即图 1(b) 的 3 个控制点处各一个实例, NP 特征会被消减.

● 分析 2: 于∃miss 程序内存块中识别 AM 时, 包含强连通分量的子 CFG 上可能存在的 NP 特征. 假设待进一步识别的∃miss 程序内存块是 m_{2k+1}, 其所处子 CFG 见图 2(a), 控制点 1 处缓存实例集是{{}}.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

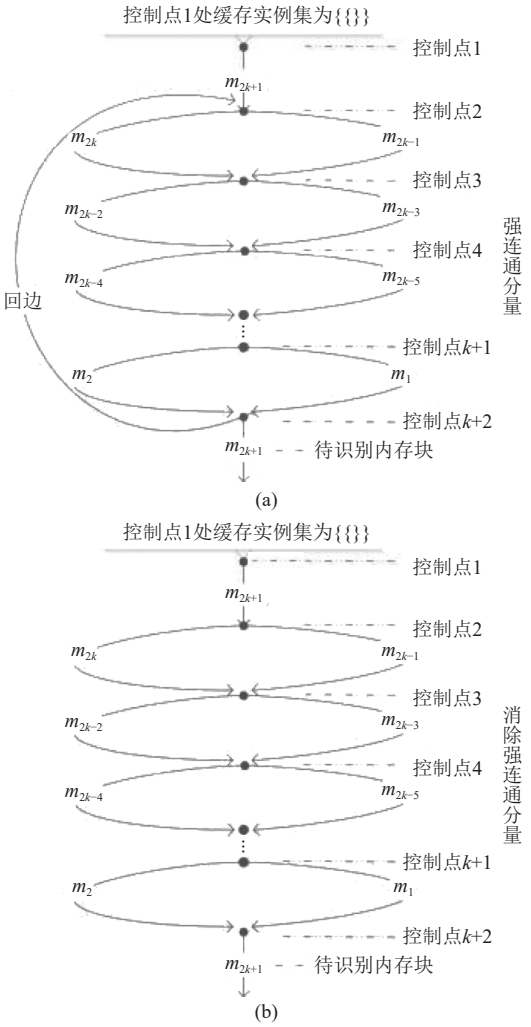

图 2 ∃miss 程序内存块所处子 CFG

以控制点 1 为起点, DFA 正向扫描子 CFG 计算其余控制点处缓存实例集, 并于各控制点处应用集包含关系反链技术[7]来消减所计算出的缓存实例集规模, 具体消减规则如下.

消减规则 2: 如缓存实例集中元素存在包含关系, 那么包含者出局.

第 1 轮迭代中, 任意控制点处的缓存实例集均不符合消减规则 2, 因此计算的缓存实例共计 2^{k+1}. 第 2 轮迭代中, 由于控制点 2 处缓存实例$\{m_{2k+1}\}$和由$\{m_2 \vee m_1, \cdots, m_{2k-2} \vee m_{2k-3}, m_{2k} \vee m_{2k-1}\}$表示的所有缓存实例均不存在包含关系, 且后续计算自两者的缓存实例间也并不存在包含关系, 因此从控制点 2 到控制点 k+1 缓存实例集的基数呈增加态势. 当计算到控制点 k+2 处时, 计算自$\{m_{2k+1}\}$的缓存实例均不包含内存块 m_{2k+1}, 且与计算自$\{m_2 \vee m_1, \cdots, m_{2k-2} \vee m_{2k-3}, m_{2k} \vee m_{2k-1}\}$的缓存实例相同, 因此控制点 k+2 处缓存实例集的基数下降. 第 2 轮迭代计算的缓存实例共计 $(k+2)2^{k+1}$. 两轮迭代计算的缓存实例共计 $(k+4)2^{k+1}$. 两轮 DFA 迭代结果见表 2.

表 2 图 2(a) 子 CFG 上 DFA 分析

轮次	控制点	缓存实例集	基数
第 1 轮	1	$\{\{\}\}$	1
	2	$\{\{m_{2k+1}\}\}$	1
	3	$\{\{m_{2k} \vee m_{2k-1}, m_{2k+1}\}\}$	2
	4	$\{\{m_{2k-2} \vee m_{2k-3}, m_{2k} \vee m_{2k-1}, m_{2k+1}\}\}$	4
	…	…	…
	k+1	$\{\{m_4 \vee m_3, \cdots, m_{2k-2} \vee m_{2k-3}, m_{2k} \vee m_{2k-1}, m_{2k+1}\}\}$	2^{k-1}
	k+2	$\{\{m_2 \vee m_1, \cdots, m_{2k-2} \vee m_{2k-3}, m_{2k} \vee m_{2k-1}, m_{2k+1}\}\}$	2^k
第 2 轮	1	$\{\{\}\}$	1
	2	$\{\{m_2 \vee m_1, \cdots, m_{2k-2} \vee m_{2k-3}, m_{2k} \vee m_{2k-1}\}, \{m_{2k+1}\}\}$	2^k+1
	3	$\{\{m_2 \vee m_1, \cdots, m_{2k-2} \vee m_{2k-3}, m_{2k} \vee m_{2k-1}\}, \{m_{2k} \vee m_{2k-1}, m_{2k+1}\}\}$	2^k+2
	4	$\{\{m_2 \vee m_1, \cdots, m_{2k-2} \vee m_{2k-3}, m_{2k} \vee m_{2k-1}\}, \{m_{2k-2} \vee m_{2k-3}, m_{2k} \vee m_{2k-1}, m_{2k+1}\}\}$	2^k+4
	…	…	…
	k+1	$\{\{m_2 \vee m_1, \cdots, m_{2k-2} \vee m_{2k-3}, m_{2k} \vee m_{2k-1}\}, \{m_4 \vee m_3, \cdots, m_{2k-2} \vee m_{2k-3}, m_{2k} \vee m_{2k-1}, m_{2k+1}\}\}$	2^k+2^{k-1}
	k+2	$\{\{m_2 \vee m_1, \cdots, m_{2k-2} \vee m_{2k-3}, m_{2k} \vee m_{2k-1}\}\}$	2^k

很明显, 精确识别内存块 m_{2k+1} 呈现 NP 特征. 虽然集包含关系反链技术[7]消减规则 2 在消减缓存实例集中起到了一定作用, 可强连通分量的存在会导致 NP 特征尤为突出. 如果图 2(a) 所示子 CFG 可被转化为图 2(b) 所示子 CFG, 即强连通分量中的回边被删除, 那么第 1 轮 DFA 迭代后即可精确识别内存块 m_{2k+1}, 计算的缓存实例总量由 $(k+4)2^{k+1}$ 降为 2^{k+1}, NP 特征会被削弱.

综上, 于∃hit 程序内存块中识别 AH 时, 子 CFG 中强连通分量的消除可能使 DFA 时间开销的 NP 特征完全消减, 于∃miss 内存块中识别 AM 时, 子 CFG 中强连通分量的消除可使 DFA 时间开销的 NP 特征削弱.

2.2 强连通分量可消除分析

一个∃hit 程序内存块$m_{\exists \text{hit}}$是 AH 时, 识别它的缓存实例集必然满足约束式 (1):

$$\forall c \in C, m_{\exists \text{hit}} \in c, |c| \leqslant k \tag{1}$$

其中, c 是缓存实例, C 是缓存实例集, $|c|$是缓存实例的基, k 是高速缓存的关联度. 在缓存替换策略 LRU 下, C 必然同时满足约束式 (2):

$$\forall c \in C, m_{\exists \text{hit}} \in c \rightarrow age(m_{\exists \text{hit}}) \leqslant |c| \leqslant k \tag{2}$$

© 中国科学院软件研究所 http://www.c-s-a.org.cn

其中, $age(m_{\exists hit})$ 表示 $m_{\exists hit}$ 的缓存年龄. 那么为了识别 $m_{\exists hit}$, 仅需计算 C 中 $m_{\exists hit}$ 的缓存年龄最大的缓存实例.

一个∃hit 程序内存块 $m_{\exists hit}$ 是确定 NC 时, 识别它的缓存实例集必然满足约束式 (3):

$$\exists c \in C,\ m_{\exists hit} \notin c,\ |c| \leqslant k \tag{3}$$

在缓存替换策略 LRU 下, C 必然同时满足约束式 (4):

$$\exists c \in C,\ m_{\exists hit} \notin c \rightarrow age(m_{\exists hit}) > k \tag{4}$$

那么为了识别 $m_{\exists hit}$, 同样仅需计算 C 中 $m_{\exists hit}$ 的缓存年龄最大的缓存实例.

强连通分量的强连通性决定了该分量中包含的程序内存块将会全部更新到缓存实例中, $m_{\exists hit}$ 的缓存年龄将会达到最大 (可能超过 k, 也可能不超过 k). 因此, 综合上述分析结论, 把图 1(a) 转化为图 1(b) 是不会影响∃hit 程序内存块的精确识别. 更一般地, 图 1(a) 中的强连通分量可以是更为复杂的结构, 其数量也可以更多.

一个∃miss 程序内存块 $m_{\exists miss}$ 是 AM 时, 识别它的缓存实例集必然满足约束式 (5):

$$\forall c \in C,\ m_{\exists miss} \notin c,\ |c| \leqslant k \tag{5}$$

在缓存替换策略 LRU 下, C 必然同时满足约束式 (6):

$$\forall c \in C,\ m_{\exists miss} \notin c \rightarrow age(m_{\exists miss}) > k \tag{6}$$

那么为了识别 $m_{\exists miss}$, 仅需计算 C 中 $m_{\exists miss}$ 的缓存年龄最小的缓存实例.

一个∃miss 程序内存块 $m_{\exists miss}$ 是确定 NC 时, 识别它的缓存实例集必然满足约束式 (7):

$$\exists c \in C,\ m_{\exists miss} \in c,\ |c| \leqslant k \tag{7}$$

在缓存替换策略 LRU 下, C 必然同时满足约束式 (8):

$$\exists c \in C,\ m_{\exists miss} \in c \rightarrow age(m_{\exists miss}) \leqslant k \tag{8}$$

那么为了识别 $m_{\exists miss}$, 同样仅需计算 C 中 $m_{\exists miss}$ 的缓存年龄最小的缓存实例.

强连通分量的强连通性会导致 $m_{\exists miss}$ 的缓存年龄向着最大发展, 因此, 综合上述分析结论, 把图 2(a) 转化为图 2(b) 不会影响∃miss 程序内存块的精确识别. 更一般地, 图 2(a) 中的强连通分量可以是更为复杂的结构, 其数量也可以更多.

综上, 于∃hit 程序内存块中识别 AH 或于∃miss 内存块中识别 AM 均可消除子 CFG 中强连通分量.

2.3 强连通分量消除方法

在∃miss 程序内存块中识别 AM 时, 如用于分析的子 CFG 中存在强连通分量, 消除其中强连通分量仅需断开回边 (如图 2(b) 所示). 本文工作实现环境为升级版 Chronos[8], 该工具在分析 CFG 中循环嵌套层数时已标记了回边, 在此仅需于子 CFG 中断开被标记的回边. 下面将聚焦于∃hit 程序内存块中识别 AH 的强连通分量消除. 首先需提取子 CFG 中强连通分量, 为此给出子 CFG 定义及两个连通性的性质.

定义 1. 设 $G(V,E,B,M,S,m^*,v^*)$ 是包含了待识别程序内存块 m^* 的子 CFG, V 是顶点集, E 是边集, B 是回边集, M 是与 m^* 映射到相同缓存集的程序内存块集, S 是子 CFG 的起始顶点集, 顶点 v^* 是识别 m^* 类型的控制点处. V 中任一顶点 v 处均存储三元组 (E^{in}, E^{out}, c), E^{in} 是以 v 为结束点的入边集, E^{out} 是以 v 为开始点的出边集, c 是 v 的涂色. E 中任一边 e 可表示为<s, m, d>, s 是 e 的开始点, d 是 e 的结束点, m 是 e 上备注且 $m \in M$. S 中顶点类型最多不超两种, 第 1 种是 CFG 的起始顶点, 第 2 种是备注为与 m^* 相同程序内存块的边的开始点 (图 1(a) 属于 S 仅包含第 2 种顶点的子 CFG).

性质 1. v 是 G 中顶点, 从 v 出发可达的所有顶点构成的集合为 D, 包含于 D 且可达 v 的所有顶点构成的集合为 P, P 是包含 v 的强连通分量顶点集.

证明: (连通性) 取 P 中顶点 w 和 u, 则 w 和 v 彼此可达, u 和 v 彼此可达, 那么途经 v、w 和 u 也彼此可达. (极大性) 由题设可知 $P \subset D$, 且 D–P 并不包含可达 v 的顶点, 因此 P 包含了与 v 可互达的所有顶点. 证毕.

性质 2. v 和 w 是 G 中顶点, H 是 G 中一个强连通分量顶点集, $v, w \notin H$. 如果 v 途经 H 中顶点可达 w, 那么 v 和 w 必不隶属同一强连通分量.

证明: 任取 H 中顶点 u, v 途经 u 可达 w. 假设 v 和 w 隶属同一强连通分量, 那么一定会有 w 可达 v, 进而有 u 途经 w 可达 v, 三者同属 H. 这与 $v, w \notin H$ 矛盾, 假设不成立. 证毕.

性质 1 给出了提取一个强连通分量的思路. 采用广度优先方式求解 D 集合过程中, 可用数值为奇数的颜色对属于 D 中顶点涂色. 以 D 涂色为基础, 采用广度优先方式求解 P 集合过程中, 可用比 D 中顶点涂色大 1 的偶数颜色对属于 P 中顶点涂色. 提取 G 中全部强连通分量的涂色方法可设计为算法 1.

算法 1. G 中全部强连通分量涂色算法

1) 使用颜色变量 r=0 涂色 V 中全部顶点;

© 中国科学院软件研究所 http://www.c-s-a.org.cn

2) 若 G 中存在未提取的强连通分量, 则:
3) 使用 r+1 涂色属于一个 D 集的全部顶点;
4) 使用 r+2 涂色 D 中属于 P 集的全部顶点;
5) 颜色变量数值增加为 r=r+2, 返回第 2) 步.

算法 1 中的第 3) 步可设计为算法 2. 性质 2 揭示了在求解 D 的过程中, 如遇到涂色数值为非 0 偶数的顶点则无需途经其继续计算.

算法 2. 求解一个 D 集合的奇数色涂色算法

1) 从 B 中取未提取的强连通分量的回边 b, 把 b 的开始点 $b.s$ 涂色为 r+1, 并把 $b.s$ 入队列 Q;
2) 若 Q 不空, 则出队一个顶点 v:
3) 取 v 出边集 E^{out} 中的每一边, 如边的结束点涂色不是非 0 偶数, 颜色也不是 r+1, 且边上备注也不是m^*, 那么边的结束点涂色 r+1 并入队列 Q, 返回第 2) 步.

算法 1 中的第 4) 步可设计为算法 3.

算法 3. 求解 P 集合的偶数色涂色算法

1) 把算法 2 中开始点 $b.s$ 涂色为 r+2, 并把 $b.s$ 入队列 Q;
2) 若 Q 不空, 则出队一个顶点 v:
3) 取 v 入边集 E^{in} 中的每一边, 如边的开始点涂色是 r+1, 那么边的开始点涂色 r+2 并入队列 Q, 返回第 2) 步.

经算法 1, G 中隶属同一强连通分量的边的顶点均被涂以相同非 0 偶数色, 不同强连通分量的涂色构成以 2 为首项差为 2 的等差数列, 而非强连通分量边的两顶点涂色不会出现相同非 0 偶数色. 据此涂色结果, 边集 E 上存在一个划分[22] P^E={E^1, E^2, E^4, …, E^o, …, E^{2n}}, o 表示 2 与 2n 之间的偶数, n 表示强连通分量的个数, E^1 是非强连通分量边的集合, 其余上标为偶数的任一个均是一个强连通分量的边集, 上标是顶点涂色. 消除强连通分量分两部分完成: 首先是把划分中上标为偶数的每一个边集转化为一条新边, 新边的备注为边集中所有边备注构成的集合; 其次是修复新边顶点与其他顶点的连通性. 完整的消除方法为算法 4, 消除强连通分量的 G 见定义 2.

算法 4. G 中全部强连通分量消除算法

/*第 1 部分*/
1) 取 P^E 中一个上标为偶数的边集 E^o;
2) 创建新边 e, 把 e 的开始点和结束点涂色为偶数 o;
3) E^i 中所有边备注构成内存块集 M_e, 并备注到 e 上, 把 e 同时并入 e 开始点 $e.s$ 的出边集 E^{out} 和 e 结束点 $e.d$ 的入边集 E^{in}, 把 e 并入新边集 E';
4) 若 P^E 中存在上标为偶数的边集, 返回第 1) 步.
/*第 2 部分*/
5) 取 E^1 中一条边 e;
6) 若 e 的开始点 $e.s$ 的颜色是非 0 偶数色 o, 则:
7) 取 E'中顶点涂色为 o 的边 $e1$, 用 $e1.d$ 替换 $e.s$, 把 e 并入 $e1.d$ 的出边集 E^{out};
8) 若 e 的结束点 $e.d$ 的颜色是非 0 偶数色 o, 则:
9) 取 E'中顶点涂色为 o 的边 $e1$, 用 $e1.s$ 替换 $e.d$, 把 e 并入 $e1.s$ 的入边集 E^{in};
10) 若 E^1 中存在未处理的边, 返回第 5) 步;
11) 合并 E^1 到 E'中, 提取 E'中顶点构成集合V'.

定义 2. 设$G'(V',E',M,S,m^*,v^*)$是由 G 消除了强连通分量而重构的子图, V'是顶点集, E'是边集, M 源自 G, S、m^*和v^*见定义 1. V'中任一顶点 v 处均存储三元组 (E^{in}, E^{out}, c), E^{in} 是以 v 为结束点的入边集, E^{out} 是以 v 为开始点的出边集, c 是 v 的涂色. E'中任一条边 e 可表示为<s, M_e, d>, s 是 e 的开始点, d 是 e 的结束点, M_e 是 e 上备注的内存块集且 $M_e \subset M$.

至此, 不包含强连通分量的G'被构建. 其构建时间开销由算法 1 的涂色开销和算法 4 的消除开销组成. 每个强连通分量的提取需对 G 中顶点集 V 涂色两次, 强连通分量数量 n 不超回边数$|B|$, 算法 1 的时间开销不超 $2|B||V|$. 算法 4 的两部分是对 G 中边集 E 做了一次扫描, 因此算法 4 的时间开销可表示为 $c|E|$, c 是常数. 综上, 构建G'的总时间开销可表示为$|V|$和$|E|$的多项式 $2|B||V|+c|E|$.

3 扩展的反链技术

消除子 CFG 中的强连通分量使 DFA 时间开销的 NP 特征得到消减或削弱, 而对集包含关系反链技术[7]的扩展将促进 NP 特征的进一步消减或削弱. 正如图 2 所示的例子, 消除了强连通分量的图 2(b) 会使 DFA 迭代计算轮数降为 1, 削弱了 DFA 时间开销的 NP 特征, 而对集包含关系反链技术的扩展有望对 2^{k+1} 数量的缓存实例实现进一步消减.

3.1 集包含关系的反链

序集是由一个同类元素构成的集合 L 及定义在 L 上的二元关系 R 构成, 形式化表示为<L;R>. 若 L 中的任意两个元素均满足关系 R, 序集就是链; 若 L 中的任意两个元素均不满足 R, 序集就是反链[23]. 序集<L;R>上必存在划分:

$$\{<L_i;R> \mid L_i \subseteq L, i \in [1,\tau], 1 \leqslant \tau \leqslant |L|\} \tag{9}$$

如任意子序集<L_i;R>服从式 (10), 则称为上界导向划分.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

$$\exists a \in L_i,\ \max_R L_i = a \tag{10}$$

其中, $\max_R$ 表示关系 R 约束的集合中的最大元素.

如任意子序集<L_i;R>服从式 (11), 则称为下界导向划分.

$$\exists a \in L_i,\ \min_R L_i = a \tag{11}$$

其中, $\min_R$ 表示关系 R 约束的集合中的最小元素.

上界导向划分中任一<L_i;R>存在一个上界 $\max_R$, 全部子序集中的上界构成的序集是一个反链, 且反链的基数不会超过$|L|$, 最理想情况是 1, 即式 (9) 所示划分仅包含一个子序集. 同理, 下界导向划分的全部子序集中的下界也可构成一个反链, 且反链的基数也不会超过$|L|$, 最理想情况也是 1.

基于上述理论, 子 CFG 中任一顶点处 (或程序控制点处) 计算的缓存实例集 L 与定义在 L 上的集包含关系$\subseteq$会构成序集<L;$\subseteq$>. 当于$\exists$hit 程序内存块中识别 AH 时, 可从<L;$\subseteq$>的上界导向划分计算反链, 为此给出了第 2.1 节的消减规则 1. 当于$\exists$miss 程序内存块中识别 AM 时, 可从<L;$\subseteq$>的下界导向划分计算反链, 为此给出了第 2.1 节的消减规则 2. 由于集包含关系满足的不易, 导致序集划分的基数可能呈现 NP 特征, 进而所计算反链的基数也呈现 NP 特征, 因此需对序集上的二元关系做扩展以进一步消减反链基数.

3.2 扩展的反链

图 2(b) 中的强连通分量被消除后, 表 2 中第 1 轮 DFA 即可精确识别内存块m_{2k+1}. 第 1 轮 DFA 中控制点 3 处缓存实例集为$\{\{m_{2k} \vee m_{2k-1}, m_{2k+1}\}\}$, 即包含两个缓存实例$\{m_{2k}, m_{2k+1}\}$和$\{m_{2k-1}, m_{2k+1}\}$. 虽然两个缓存实例因不满足$\subseteq$而无法消减, 可当控制点 3 到控制点 k+2 间待访问的程序内存块集能够被提前获知, 两个缓存实例有消减的可能. 消减理由为: 控制点 3 到控制点 k+2 间并没有出现$\{m_{2k}, m_{2k+1}\}$和$\{m_{2k-1}, m_{2k+1}\}$中的内存块, 沿控制点 3 到控制点 k+2 间任一分支计算自$\{m_{2k}, m_{2k+1}\}$和$\{m_{2k-1}, m_{2k+1}\}$的两个新的缓存实例仅是基数变大而基数间大小关系与$\{m_{2k}, m_{2k+1}\}$和$\{m_{2k-1}, m_{2k+1}\}$基数间大小关系始终保持一致, 那么依赖$\{m_{2k}, m_{2k+1}\}$和$\{m_{2k-1}, m_{2k+1}\}$基数间的大小关系可以确定消减对象. 更一般地, 给定G' (定义 2) 某程序控制点 v 处的两个缓存实例, 若两个缓存实例均包含与待识别内存块m^*相同程序内存块, 且借助 v 到v^*间待访问程序内存块集可得到事实“沿控制点 v 到控制点v^*间任一分支计算自两给定缓存实例的两个新缓存实例的基数间大小关系与两给定缓存实例基数间大小关系保持一致”, 那么控制点 v 处的两个缓存实例之一可被消减. 下面先给出计算各程序控制点后待访问程序内存块集的方法, 然后给出二元关系扩展方法.

设G' (定义 2) 中任意一个顶点 v 后待访问的程序内存块集为 Δ_v, 则包含了 G'中每一个顶点后待访问的程序内存块集的集合就是$\Delta = \{\Delta_v | v \in V'\}$. 为了避免计算 Δ 时某些顶点后 Δ_v 的重复计算, 先以v^*为起点对子图中所有顶点做逆向拓扑排序, 然后从v^*出发采用顶点接力方式来逆向计算每一个 Δ_v, 具体见算法 5.

算法 5. 获知 Δ 的顶点接力算法

1) 以v^*为起点获得顶点逆向拓扑序 *tps*;
2) 取 *tps* 中一顶点 v, 把 v 出边集 E^{out} 中每一边的备注 M_e 并入 Δ_v;
3) 若 *tps* 中顶点未取尽, 则返回第 2) 步.

算法 5 第 1) 步通过遍历E'中所有边完成拓扑排序, 时间开销级别为$|E'|$; 接力过程 2) 到 3) 时间开销级别为$\sum_{v \in V'} (v.E^{\text{out}}) < |E'|$; 总时间开销不超$c|E'|$, c 是常量.

定义 3. 设新的二元关系为$\preccurlyeq$, G'中顶点 v 处任意两个缓存实例为 l_1 和 l_2, 且 l_1 和 l_2 均包含与m^*相同程序内存块, v 后待访问程序内存块集为 Δ_v. 若式 (12)–式 (14) 任一成立, 则 $l_1 \preccurlyeq l_2$ 成立, 表示$|l_1| \leqslant |l_2|$的关系会于计算自 l_1 和 l_2 的新缓存实例上得到保持, 含义为 l_2 比 l_1 更具计算基数更大缓存实例的潜力, 或 l_1 比 l_2 更具计算基数更小缓存实例的潜力.

$$l_1 \subseteq l_2 \tag{12}$$

$$|l_1| < |l_2| \wedge |l_1 \cup ((l_2 - l_1) \cap \Delta_v)| \leqslant |l_2| \tag{13}$$

$$|l_1| = |l_2| \wedge ((l_2 - l_1) \cap \Delta_v) = \{\} \tag{14}$$

为了证明 $l_1 \preccurlyeq l_2$ 成立, 可证明: 对于G'中任一分支 p, 其起始于定义 3 中的 v, 终止于G'中v^*, 其上边备注构成集合 M_p, $M_p \subseteq \Delta_v$, 由式 (12)–式 (14) 均可推导出$|l_1 \cup M_p| \leqslant |l_2 \cup M_p|$.

由式 (12) 证明: $l_1 \subseteq l_2$ 可知$|l_1| \leqslant |l_2|$, 同时有 $l_1 \cup M_p \subseteq l_2 \cup M_p$ 及进一步可知$|l_1 \cup M_p| \leqslant |l_2 \cup M_p|$, $l_1 \preccurlyeq l_2$ 成立. 证毕.

由式 (13) 证明: M_p 最多可划分出 4 个互不相交子集, 与 l_1 和 l_2 的交集均为空的 S_1, 同时包含于 l_1 和 l_2 的 S_2, 包含于 l_1 但与 l_2 交集为空的 S_3, 与 l_1 交集为空但包含于 l_2 的 S_4.

由$|l_1| \leqslant |l_2|$可分别推导出$|l_1 \cup S_1| \leqslant |l_2 \cup S_1|$, $|l_1 \cup S_2| \leqslant$

© 中国科学院软件研究所 http://www.c-s-a.org.cn

$|l_2 \cup S_2|$和$|l_1 \cup S_3| \leqslant |l_2 \cup S_3|$.

$(l_2-l_1)\cap\Delta_v$ 等于 S_4, 则由$|l_1 \cup ((l_2-l_1)\cap\Delta_v)| \leqslant |l_2|$可知$|l_1 \cup S_4| \leqslant |l_2| = |l_2 \cup S_4|$.

综上可得$|l_1 \cup S_1 \cup S_2 \cup S_3 \cup S_4| \leqslant |l_2 \cup S_1 \cup S_2 \cup S_3 \cup S_4|$, 即$|l_1 \cup M_p| \leqslant |l_2 \cup M_p|$. 证毕.

由式 (14) 证明: 同式 (13) 证明中的划分方法, M_p 被划分出 4 个互不相交子集 S_1、S_2、S_3 和 S_4.

由$|l_1|=|l_2|$可分别推导出$|l_1 \cup S_1|=|l_2 \cup S_1|$, $|l_1 \cup S_2|=|l_2 \cup S_2|$和$|l_1 \cup S_3| \leqslant |l_2 \cup S_3|$.

$(l_2-l_1)\cap\Delta_v$ 等于 S_4 且为空, 则有$|l_1 \cup S_4|=|l_1|=|l_2|=|l_2 \cup S_4|$. 综上得$|l_1 \cup S_1 \cup S_2 \cup S_3 \cup S_4| \leqslant |l_2 \cup S_1 \cup S_2 \cup S_3 \cup S_4|$, 即$|l_1 \cup M_p| \leqslant |l_2 \cup M_p|$. 证毕.

至此, 基于上述定义的二元关系$\leqslant$可定义G'中任意顶点处的缓存实例序集以及求解序集反链的函数.

定义 4. 设G'中任一顶点 v 处的缓存实例集是 L_v, 结合二元关系$\leqslant$可形成序集$\mathcal{L}_v < L_v, \Delta_v; \leqslant >$.

当于∃hit 程序内存块中识别 AH 时, 待求解的反链$\overline{\mathcal{L}_{v\text{hit}}}$为$\overline{< L_v, \Delta_v; \leqslant >_{\text{hit}}}$. $\mathcal{L}_v$ 到$\overline{\mathcal{L}_{v\text{hit}}}$的求解函数为:

$$\begin{cases} \Gamma^n(\mathcal{L}_v) = \Gamma \cdots \Gamma(\mathcal{L}_v) = \overline{\mathcal{L}_{v\text{hit}}} \\ \Gamma(\mathcal{L}_v) = \begin{cases} \mathcal{L}_v - \{a\}, & \exists a, b \in \mathcal{L}_v, a \leqslant b \\ \mathcal{L}_v, & \text{otherwise} \end{cases} \end{cases} \tag{15}$$

当于∃miss 程序内存块中识别 AM 时, 待求解的反链$\overline{\mathcal{L}_{v\text{miss}}}$为$\overline{<L_v, \Delta_v; \leqslant>_{\text{miss}}}$, $\mathcal{L}_v$ 到$\overline{\mathcal{L}_{v\text{miss}}}$的求解函数为:

$$\begin{cases} \Gamma^n(\mathcal{L}_v) = \Gamma \cdots \Gamma(\mathcal{L}_v) = \overline{\mathcal{L}_{v\text{miss}}} \\ \Gamma(\mathcal{L}_v) = \begin{cases} \mathcal{L}_v - \{b\}, & \exists a, b \in \mathcal{L}_v, a \leqslant b \\ \mathcal{L}_v, & \text{otherwise} \end{cases} \end{cases} \tag{16}$$

4 第 3 阶段的重构

定义边上开始点到结束点的缓存实例更新函数.

定义 5. 取G'中任一条边 e <s, M_e, d>, 顶点 s 处的一个缓存实例为 $l_{e.s}$, 下标 $e.s$ 表示 e 的开始点, 经更新函数式 (17) 可得顶点 d 处的一个缓存实例 $l_{e.d}$, 下标 $e.d$ 表示 e 的结束点.

$$update : l_{e.s} \cup e.M_e \mapsto l_{e.d} \tag{17}$$

设开始点 s 处的缓存实例集为 $L_{e.s}$, 经 e 计算的结束点 d 处的缓存实例子集为 $L_{e.d}$, 边 e 上的更新函数可形式化为式 (18):

$$\begin{cases} Update : L_{e.s} \cup M_e \mapsto L_{e.d} \\ \forall l_{e.s} \in L_{e.s},\ update(l_{e.s}, M_e) \in L_{e.d} \end{cases} \tag{18}$$

更新函数与反链求解函数整合, 设计正向 DFA 分析方法, 如算法 6 所示.

算法 6. 正向 DFA 分析算法

1) 获得G'中顶点正向拓扑序 tps;
2) 顺序取 tps 中不属于 S 的顶点 v;
3) 取 v 入边集 E^{in} 中每一条边 e, 更新 $e.s$ 处的缓存实例集到 v 处, $L_v := Update(L_{e.s}, M_e) \cup L_v$;
4) 若于∃hit 程序内存块中识别 AH, 调用式 (15) 计算序集$\mathcal{L}_v$的反链, 若于∃miss 程序内存块中识别 AM, 调用式 (16) 计算序集$\mathcal{L}_v$的反链;
5) 若 tps 中顶点未取尽, 则返回第 2) 步.

根据算法 6 计算的v^*处的缓存实例序集$\mathcal{L}_{v^*}$的反链可精确判定待识别内存块m^*的类型, 判定规则如下.

当m^*是∃hit 程序内存块时, 若$\overline{\mathcal{L}_{v\text{hit}}}$和$m^*$满足约束式 (1), 则$m^*$为 AH, 若$\overline{\mathcal{L}_{v\text{hit}}}$和$m^*$满足约束式 (3), 则$m^*$为确定 NC; 当$m^*$是∃miss 程序内存块时, 若$\overline{\mathcal{L}_{v\text{miss}}}$和$m^*$满足约束式 (5), 则$m^*$为 AM, 若$\overline{\mathcal{L}_{v\text{miss}}}$和$m^*$满足约束式 (7), 则$m^*$为确定 NC.

最后于图 3 中给出重构的第 3 阶段分析框架.

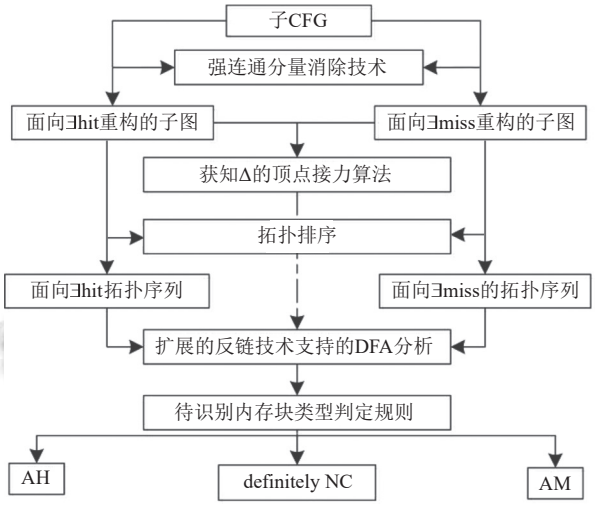

图 3 重构的第 3 阶段分析

该框架首先根据待识别程序内存块m^*的类别来确定强连通分量消除方法, 若是∃hit 则采用算法 1–算法 4 来合并强连通分量, 若是∃miss 则断开回边来消除强连通分量; 其次是采用算法 5 来计算消除强连通分量 (子 CFG) 中各控制点后待访问程序内存块集, 为求解各控制点处的缓存实例序集反链创造条件; 第 3 步是采用扩展的反链技术支持的算法 6 来计算反链; 最后依据所求反链和判定规则对m^*做进一步的精确分类.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

5 实验评价

实验目的: 证实强连通分量消除技术和扩展的反链技术对消减精确缓存分类分析时间开销 NP 特征有效.

验证工具: Chronos[8]是一款免费开源工具, 实现了 LRU 高速缓存建模、程序 CFG 构建和 Must-May 分析.

程序基准: 用于实验评价的 49 个程序基准取自 MRTC[24]基准集和 TACLeBench[25]基准集, 既涵盖程序规模较小且程序结构不太复杂的基准也涵盖程序规模较大且程序结构复杂的基准. 程序基准均被编译为 SimpleScalar[26]体系结构支持的目标代码.

硬软件平台: 运行工具的硬件平台为 Intel Core i7 处理器+4 GB 主存+8 GB 虚存, 软件平台为 Ubuntu 18.04+GCC 7.4.0.

用于实验比较的精确缓存分类分析[12]的第 2 阶段和第 3 阶段被构建, 提出的强连通分量消除技术和扩展的反链技术支持的第 3 阶段被构建. 鉴于两种第 3 阶段精确性相同, 仅时间开销存在差异, 因此实验聚焦两种分析的时间开销比较. 在时间开销比较过程中, 如所取程序基准的缓存分类能够被前两个阶段精确实现, 这些基准将不再出现在第 3 阶段的时间开销比较中. 在第 3 阶段时间开销比较中, 已有的第 3 阶段分析[12]被标记为集包含反链, 重新设计的第 3 阶段被标记为消除强连通分量+扩展的反链.

5.1 容量较大高速缓存上的实验比较

当高速缓存总容量为 8 KB, 缓存集数为 32, 集合关联度为 8 时, 49 个基准中的 39 个由前两阶段精确分类, 分类结果如表 3 所示, 其余 10 个基准由第 3 阶段完成最终精确分类, 第 3 阶段的时间开销比较如图 4 所示, 实际测量时间开销最高为 15 477 936 ms.

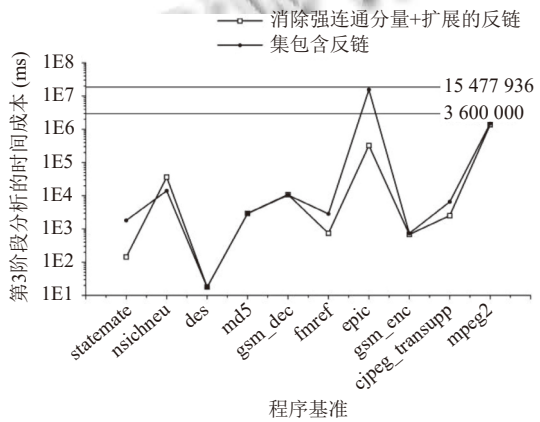

图 4 8 KB 高速缓存上时间开销比较

由图 4 显示的时间成本比较可知, 消除强连通分量+扩展的反链可使 7 个程序基准的分类分析时间开销下降, 3 个程序基准的分类分析时间开销略微上浮, 具体情况如下.

表 3 前两个阶段精确分类的基准 (8 KB)

基准	Must-May分析和∃hit+∃miss分析		
	AH	AM	definitely NC
adpcm	2 548	212	49
expint	131	27	0
edn	1 364	140	1
cover	642	111	70
prime	150	21	0
bsort100	96	15	0
fir	114	20	0
ndes	1 446	122	4
qurt	468	46	19
sqrt	69	17	0
jfdcint	597	90	0
cnt	215	28	0
st	363	46	0
ns	287	14	0
ludcmp	496	49	0
matmult	473	31	0
fdct	426	71	0
compress	766	117	131
fft	1 519	82	40
lms	992	87	40
ud	605	62	10
select	315	32	0
insertsort	88	16	0
janne_complex	29	16	0
fibcall	23	6	0
duff	100	21	0
crc	391	40	4
bs	41	9	0
fac	51	9	0
lcdnum	125	20	0
minver	789	82	34
whet	540	87	5
g723_enc	2 243	228	305
petrinet	1 460	247	213
sha	2 201	181	283
h264_dec	11 669	189	2 944
huff_dec	2 350	84	54
susan	11 294	1 073	1 998
anagram	2 286	181	431

分析取自 TACLeBench 的 epic 基准, 消除强连通分量+扩展的反链的时间开销为 324 466 ms, 不超 6 min, 集包含反链的时间开销为 15 477 936 ms, 约 4 h 30 min, 下降量为 7 个实验基准中最大的, 下降约 4 h 24 min. 分析其余 6 个基准的时间开销下降量在 100 ms–2 min 之间.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

分析取自 MRTC 的 nsichneu 基准, 消除强连通分量+扩展的反链的时间开销为 36 272 ms, 约 36 s, 集包含反链的时间开销为 14 063 ms, 约 14 s, 上浮量为 3 个实验基准中最大的, 上浮约 22 s. 分析其余 2 个基准的时间开销上浮量分别为 25 ms 和 350 ms.

5.2 容量居中高速缓存上的实验比较

当高速缓存总容量为 4 KB, 缓存集数为 16, 集合关联度为 8 时, 49 个基准中的 33 个由前两阶段精确分类, 分类结果如表 4 所示, 其余 16 个基准由第 3 阶段完成最终精确分类, 第 3 阶段的时间开销比较如图 5 所示. 实际测量时间开销最高为 5 191 619 ms.

由图 5 显示的时间开销比较可知, 消除强连通分量+扩展的反链可使 9 个程序基准的分类分析时间开销下降, 7 个程序基准的分类分析时间开销略微上浮, 具体情况如下.

表 4 前两个阶段精确分类的基准 (4 KB)

基准	Must-May分析和∃hit+∃miss分析		
	AH	AM	definitely NC
expint	127	27	4
edn	1 362	140	3
cover	626	111	86
prime	148	21	2
bsort100	96	15	0
fir	111	20	3
ndes	1 411	122	39
qurt	431	46	56
sqrt	69	17	0
jfdcint	597	90	0
cnt	214	28	1
st	363	46	0
ns	287	14	0
ludcmp	496	49	0
matmult	473	31	0
fdct	426	71	0
compress	764	117	133
fft	1 505	82	54
lms	981	87	51
select	286	32	29
insertsort	88	16	0
janne_complex	29	16	0
fibcall	23	6	0
duff	100	21	0
crc	370	40	25
bs	41	9	0
des	6 444	1 949	11
fac	51	9	0
lcdnum	122	29	3
whet	536	87	9
md5	35 031	10 772	799
petrinet	1 392	247	281
sha	2 149	181	335

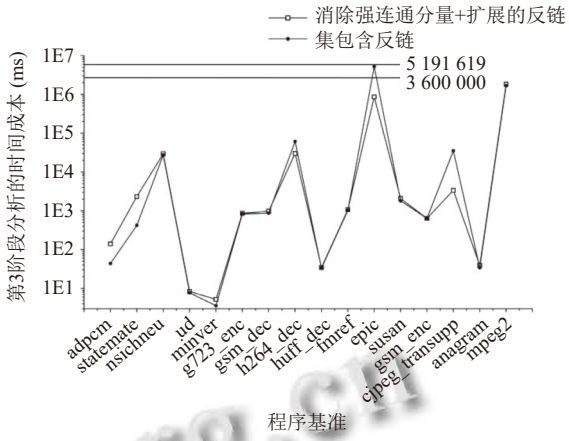

图 5 4 KB 高速缓存上时间开销比较

分析取自 TACLeBench 的 epic 基准, 消除强连通分量+扩展的反链的时间开销为 859 723 ms, 约 14 min, 集包含反链的时间开销为 5 191 619 ms, 约 1 h 40 min, 下降量为 9 个实验基准中最大的, 下降约 1 h 26 min. 分析其余 8 个基准的时间开销下降量在 1 ms–32 s 之间.

分析 TACLeBench 中 mpeg2 基准, 消除强连通分量+扩展的反链时间开销为 1 824 830 ms, 约 30 min, 集包含反链的时间成本为 1 653 520 ms, 约 27 min, 上浮量为 7 个实验基准中最大, 上浮约 3 min. 分析其余 6 个基准的时间开销上浮量在 2 ms–2 s 之间.

5.3 容量较小高速缓存上的实验比较

当高速缓存总容量为 2 KB, 缓存集数为 8, 集合关联度为 8 时, 49 个基准中的 29 个由前两阶段精确分类, 分类结果如表 5 所示, 其余 20 个基准由第 3 阶段完成最终精确分类, 第 3 阶段的时间开销比较如图 6 所示. 实际测量时间开销最高为 208 194 ms.

由图 6 显示的时间开销比较可知, 消除强连通分量+扩展的反链可使 18 个程序基准的分类分析时间开销下降, 2 个程序基准的时间开销略微上浮, 具体情况如下.

分析取自 TACLeBench 的 mpeg2 基准, 消除强连通分量+扩展的反链的时间开销为 164 759 ms, 约 2 min 44 s, 集包含反链付出的时间成本为 172 889 ms, 约 2 min 52 s, 下降量为 18 个实验基准中最大的, 下降约 8 s. 分析其余 17 个基准的时间开销下降量在 1 ms–2 s 之间.

分析取自 TACLeBench 的 anagram 基准, 消除强连通分量+扩展的反链的时间开销为 172 ms, 集包含反链的时间开销为 84 ms, 上浮量为 2 个实验基准中最大的, 上浮 88 ms. 分析另一个基准的时间开销上浮量为 1 ms.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

表 5 前两个阶段精确分类的基准 (2 KB)

基准	Must-May分析和∃hit+∃miss分析		
	AH	AM	definitely NC
expint	124	27	7
edn	1 361	141	3
cover	618	111	94
prime	141	21	9
bsort100	93	15	3
fir	107	20	7
ndes	1 406	124	42
qurt	425	46	62
sqrt	67	17	2
jfdcint	596	91	0
cnt	214	28	1
st	363	46	0
ns	286	14	1
ludcmp	495	49	1
matmult	473	31	0
fdct	426	71	0
nsichneu	6 935	2 349	104
select	268	32	47
insertsort	85	16	3
janne_complex	29	16	0
fibcall	23	6	0
duff	100	21	0
crc	361	40	33
bs	41	9	0
des	6 437	1 953	14
fac	51	9	0
lcdnum	112	29	13
minver	756	82	67
whet	536	87	9

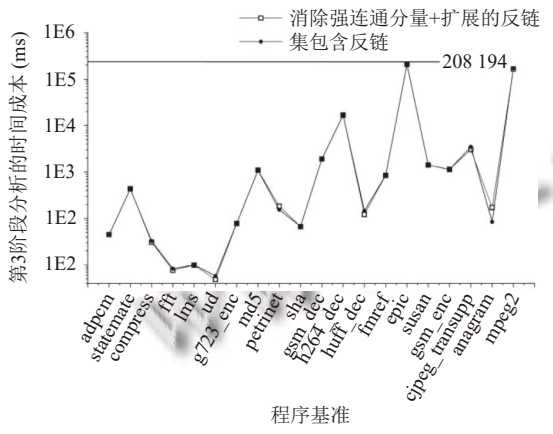

图 6 2 KB 高速缓存上时间开销比较

综合上述实验结果, 可得出如下结论.

首先, 49 个基准上 147 次前两阶段实验中的 101 次实现精确分类, 46 次需要启动第 3 阶段完成最终精确分类. 也就是说约 31% 的实验需启动第 3 阶段分析来完成精确分类. 说明第 3 阶段分析可较高概率地被用到.

其次, 在 46 对第 3 阶段时间开销比较实验中, 有 34 对 (约占 74%), 消除强连通分量+扩展的反链的时间开销低于集包含反链的时间开销, 且最大下降量达到约 4 h 30 min; 约占 26% 的其余 12 对时间开销比较实验, 消除强连通分量+扩展的反链的时间开销略高于集包含反链的时间开销, 且最大上浮量不超过 3 min. 说明以强连通分量消除和反链扩展付出的时间代价来换取精确缓存分类分析时间开销 NP 特征的消减是可行的, 有助于改善分类分析工具时间性能.

最后, 实现强连通分量消除和反链扩展所付出的时间代价虽并不一定总是换来第 3 阶段时间开销的下降, 如因待第 3 阶段识别的程序内存块身处结构并不复杂的子 CFG, 也因定义的二元关系$\preccurlyeq$在求反链时不能得到满足, 或者是上述原因的混合, 可正如前文分析的, 不论是消减强连通分量的时间开销还是求解扩展反链的时间开销均是子 CFG 顶点和边表示的多项式级, 以多项式级时间开销换取 NP 特征进一步消减是划算的, 这一点从实验结果中可以观察到.

6 结论与展望

缓存分类分析是实时嵌入式系统研究领域中用于预测高速缓存行为的重要方法. 该方法不仅被广泛用于实时内存任务最坏执行时间估计, 也将会应用到高速缓存侧信道攻击[22]防御中. 鉴于精确分类程序内存块的 NP 特征, 研究以更少的分析时间开销实现程序内存块的精确分类成为当前的难点.

本文分析了精确缓存分类分析时间开销呈现非多项式特征的可能原因, 即强连通分量的存在, 集包含关系求反链的局限, 提出了强连通分量消除技术和扩展的反链技术来进一步消减精确缓存分类分析时间开销的 NP 特征. 经实验验证, 提出技术的额外时间代价在换取精确缓存分类分析时间开销 NP 特征进一步消减中占有优势, 证实了方法的可行性, 对改善分类分析工具时间性能有帮助.

在未来的工作中, 一是拟将这些技术应用到多层高速缓存分类中, 在保证分类精确性前提下进一步提升多层高速缓存分类分析的时间性能; 二是拟将这些技术应用到高速缓存侧信道攻击[27]防御中, 为高速缓存信息泄露预测研究提供有价值参考信息. 至于所定义的用于求反链的二元关系$\preccurlyeq$的进一步完备, 只能寄希望于 NP=P 被证明.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

参考文献

1 何立民. 物联网时代的嵌入式系统机遇. 单片机与嵌入式系统应用, 2011(3): 1–3. [doi: 10.3969/j.issn.1009-623X.2011.03.001]

2 郭斌, 刘思聪, 刘琰, 等. 智能物联网: 概念、体系架构与关键技术. 计算机学报, 2023, 46(11): 2259–2278. [doi: 10.11897/SP.J.1016.2023.02259]

3 Iyer R, De V, Illikkal R, *et al*. Advances in microprocessor cache architectures over the last 25 years. IEEE Micro, 2021, 41(6): 78–88. [doi: 10.1109/MM.2021.3114903]

4 王颖洁, 周宽久, 李明楚. 实时嵌入式系统的 WCET 分析与预测研究综述. 计算机科学, 2019, 46(6A): 16–22.

5 Lv MS, Guan N, Reineke J, *et al*. A survey on static cache analysis for real-time systems. Leibniz Transactions on Embedded Systems, 2016, 3(5): 5.

6 Monniaux D, Touzeau V. On the complexity of cache analysis for different replacement policies. Journal of the ACM (JACM), 2019, 66(6): 41.

7 Touzeau V, Maiza C, Monniaux D, *et al*. Fast and exact analysis for LRU caches. Proceedings of the ACM on Programming Languages, 2019, 3(POPL): 54.

8 Chattopadhyay S, Roychoudhury A. Scalable and precise refinement of cache timing analysis via path-sensitive verification. Real-time Systems, 2013, 49(4): 517–562. [doi: 10.1007/s11241-013-9178-0]

9 Yan Q, Li Y, Wu Y, *et al*. DFlow: A data flow analysis tool for C/C++. IEEJ Transactions on Electrical and Electronic Engineering, 2021, 16(12): 1635–1641. [doi: 10.1002/tee.23467]

10 Debnath P, Konwar N, Radenović S. Metric fixed point theory: Applications in science, engineering and behavioural sciences. Singapore: Springer, 2021. 3–20.

11 Mueller F, Whalley DB, Harmon M. Predicting instruction cache behavior. Proceedings of the 1994 ACM SIGPLAN Workshop on Language, Compiler, and Tool Support for Real-time Systems. ACM, 1994. 1–10.

12 Mueller F, Whalley DB. Fast instruction cache analysis via static cache simulation. Proceedings of the 1995 Simulation Symposium. Phoenix: IEEE, 1995. 105–114.

13 Mueller F. Generalizing timing predictions to set-associative caches. Proceedings of the 9th EuroMicro Workshop on Real Time Systems. Toledo: IEEE, 1997. 64–71.

14 Mueller F. Timing analysis for instruction caches. Real-time Systems, 2000, 18(2–3): 217–247. [doi: 10.1023/A:1008145215849]

15 Ferdinand C, Wilhelm R. Efficient and precise cache behavior prediction for real-time systems. Real-time Systems, 1999, 17(2–3): 131–181. [doi: 10.1023/A:1008186323068]

16 Ferdinand C, Martin F, Wilhelm R, *et al*. Cache behavior prediction by abstract interpretation. Science of Computer Programming, 1999, 35(2–3): 163–189. [doi: 10.1016/S0167-6423(99)00010-6]

17 Touzeau V, Maiza C, Monniaux D, *et al*. Ascertaining uncertainty for efficient exact cache analysis. Proceedings of the 29th International Conference on Computer Aided Verification. Heidelberg: Springer, 2017. 22–40.

18 喻垚慎, 黄志球, 沈国华, 等. 基于抽象解释的嵌入式软件模块化 Cache 行为分析框架. 计算机学报, 2019, 42(10): 2251–2266. [doi: 10.11897/SP.J.1016.2019.02251]

19 Stock G, Hahn S, Reineke J. Cache persistence analysis: Finally exact. Proceedings of the 2019 IEEE Real-time Systems Symposium. Hong Kong: IEEE, 2019. 481–494.

20 Li XF, Liang Y, Mitra T, *et al*. Chronos: A timing analyzer for embedded software. Science of Computer Programming, 2007, 69(1–3): 56–67. [doi: 10.1016/j.scico.2007.01.014]

21 崔勇, 张小平. 图论与代数结构. 北京: 清华大学出版社, 2022. 16–20.

22 吴昊, 梁艳春, 李雄飞, 等. 离散数学. 北京: 清华大学出版社, 2022. 16–17.

23 姚卫, 路玲霞. 序与格论基础. 北京: 清华大学出版社, 2023. 1–6.

24 WCET. Benchmarks. http://www.mrtc.mdh.se/projects/wcet/benchmarks.html. [2025-06-15].

25 Falk H, Altmeyer S, Hellinckx P, *et al*. TACLeBench: A benchmark collection to support worst-case execution time research. Proceedings of the 16th International Workshop on Worst-case Execution Time Analysis. Toulouse: WCET, 2016. 2.

26 Burger D, Austin TM. The simplescalar tool set, version 2.0. ACM SIGARCH Computer Architecture News, 1997, 25(3): 13–25. [doi: 10.1145/268806.268810]

27 李志峰, 高玉琢. 基于高速缓存的侧信道攻击方法研究. 网络安全技术与应用, 2021(9): 4–7. [doi: 10.3969/j.issn.1009-6833.2021.09.003]

(校对责编: 张重毅)

© 中国科学院软件研究所 http://www.c-s-a.org.cn