GISEES:面向嵌入式系统的扩展指令集自动产生方法

陈 虎,陈书明,陈胜刚,谷会涛,陈小文
(国防科学技术大学计算机学院,湖南长沙 410073)

摘 要: 面向应用的指令集处理器通过增加扩展指令可有效提升处理器的性能,满足上市时间要求.然而为嵌入式系统定制扩展指令需解决以下3个问题:设计空间随应用复杂度的增加指数增加,有限的片上资源限制了扩展指令的数量和复杂度,现有指令集扩展算法复杂度高难以在嵌入式系统上运行.本文提出了一种快速的指令集扩展方法GISEES.该方法以应用的典型操作为中心产生扩展指令以裁剪了设计空间,并采用基于最大公共等价子串的资源共享策略减少资源开销和插入的多路选择器的数量.实验结果表明,该方法具有线性复杂度,可产生效率更高的扩展指令,更适合为嵌入式系统定制高效的扩展指令.

关键词: 扩展指令;指令识别;指令选择;资源共享
中图分类号: TP314 **文献标识码:** A **文章编号:** 0372-2112 (2011) 09-2026-08

GISEES: Automatic Generation of Instruction-Set Extensions for Embedded Systems

CHEN Hu, CHEN Shu-ming, CHEN Sheng-gang, GU Hui-tao, CHEN Xiao-wen
(*Department of Computer Science, National University of Defense Technology, Changsha, Hunan* 410073, *China*)

Abstract: Application-specific instruction-set processors (ASIPs) with extended instructions (EIs) can effectively improve performance and meet time-to-market requirements. However, there are three problems to be solved when customizing EIs for embedded systems. First, design space increases exponentially as applications become more complex. Second, limited on-chip resources restrict the number and complexity of EIs. Third, current instruction-set extension (ISE) algorithms can hardly run on embedded systems due to their high complexity. We propose a fast automatic method called GISEES to address these problems. GISEES can effectively prune design space by enumerating EIs around typical operations of applications, and reduce resources overhead and the number of inserted multiplexers through sharing resources based on finding the maximal common equivalent substring (MCES). Experiment results show that, GISEES features linear complexity and can generate EIs with higher efficiency, which means it is more suitable for customizing power efficient instruction-set extension.

Key words: extended instruction; instruction enumeration; instruction selection; resource sharing

1 引言

90/10定律表明,针对关键代码段对处理器的指令集进行适当调整可获得可观的性能提升.因此,面向应用的处理器的性能很大程度上依赖于指令集与应用的匹配程度.由于从头开始设计处理器非常耗时,为处理器增加扩展指令形成ASIP(Application-Specific Instruction-set Processor)证明是一种提升处理器的性能和效率,满足上市时间要求的有效方式[1,2].但随着应用复杂度的增加,设计者难以再手工分析应用选择适当的扩展指令.因此近年来,很多学者在ISE(Instruction-Set Extension)领域做了大量工作[3~16].

EI(Extended Instruction)是在CFU(Custom Functional Unit)上实现的执行次数较多可用于同一应用的多个部分或同一领域的多个应用的计算模式(Computing Pattern).为嵌入式系统快速定制高效的扩展指令需解决以下3个问题.首先,设计空间随应用复杂度的增加指数增加,设计者难以手工分析应用并选择所需的扩展指令.第二,有限的资源限制了扩展指令的数量和复杂度,要求在相同的开销下获得最大的性能加速比.第三,现

收稿日期:2010-11-04;修回日期:2011-02-11
基金项目:国家高技术研究发展计划(863计划)课题(No.2007AA01Z108)

有指令集扩展算法复杂度高,难以在嵌入式处理器上运行使其根据执行情况自动调整指令集.

为解决这些问题,本文提出了一种快速的扩展指令集产生方法 GISEES(Generation of Instruction-Set Extensions for Embedded Systems).GISEES 以应用的典型操作为中心产生扩展指令,有效裁剪了设计空间,实现了设计空间的快速开发.同时,GISEES 采用基于最大公共等价子串 MCES(Maximal Common Equivalent Substring)的资源共享策略提高了资源利用率,减小了多路选择器产生的面积和延时开销.实验结果表明,该方法具有线性复杂度,可产生效率更高的扩展指令,更适合为嵌入式系统定制高效的扩展指令.

2　相关工作

文献[3]将尽可能多的操作包含在多输入单输出计算模式中以提高加速比.文献[4]采用穷举方式产生设计空间中所有的有效计算模式.文献[5]引入引导函数控制计算模式的生长以裁剪设计空间.文献[6]利用应用的结构化特性提出了一种可扩展的指令扩展算法.文献[7]通过将 DFG(Data Flow Graph)划分成多个区域极大地降低了算法复杂度.文献[8]允许在计算模式中复制操作并将匹配问题转换成最小面积逻辑覆盖问题.文献[9]允许存储操作包含在计算模式中.文献[10~13]通过提出新的硬件结构提高扩展指令的加速比.文献[14]在选择过程中采用最优子图覆盖的方法减小计算模式关键路径的延迟.文献[15]在估计开销时合并多个指令在同一阶段的相同操作以精确估计扩展的面积.文献[16]通过产生简单计算模式并选择性地组合出复杂计算模式以减小算法复杂度.

文献[17]通过在相同的单元上实现算数和逻辑操作实现资源共享.文献[18]首先在可重构系统的一个 CFU 上压缩多条高效的扩展指令以提高资源利用率并减少系统的重构时间.文献[19]采用基于最长公共子序列的启发式策略将指令有效地压缩在一个数据通路中.文献[20]基于文献[19]引入参数化的面积和延时信息以进一步减小数据通路的复杂度.文献[21]基于全局 DFG 匹配提出了一种资源共享算法旨在提高基于 ADL(Architecture Description Language)产生的 RTL 代码的面积效率.文献[22]通过将共享资源流水以避免多个指令共享资源时引起的结构冲突.文献[16]将简单计算模式视作多个复杂计算模式共享资源时的基本单元以提高资源利用率并减小算法的复杂度.

现有算法虽然可获得可观的性能提升或面积压缩比,但不适合为嵌入式系统快速定制高效的扩展指令.首先,现有算法[3~16]都将 DFG 视为常规有向无回路图(Directed Acyclic Graph)并采用图形理论中的方法开发设计空间,而没有向算法提供应用的相关信息.由于算法计算复杂度随图中节点数的增加指数增加,现有算法不得不采用各种经验策略以裁剪庞大的设计空间.这样做可能产生不能体现应用典型特征的局部最优结果.第二,现有算法[16~22]都是为了在指令选定后减少指令的面积开销.这种将指令选择和面积压缩分开进行的做法无法获得最大的性能加速比.事实上,由于总加速比取决于指令的数量和每条指令的加速比,选择和压缩是不可分割的,只有在选择过程中同时考虑指令间资源共享才能得出获得最大加速比的最优解.第三,现有工作[16~22]都保持指令的数据流且不考虑插入的多路选择器对数据通路的面积,延时和可测试性的影响.我们发现,适当改变指令的数据流结构可减少多路选择器的数量,从而减小多路选择器带来的影响.例如,若路径 $B(+,+,\hat{})$中的 ADD 和 XOR 操作间插入操作 SHR 且 SHR 的第二个操作数置为 0 形成路径 $A(-,+,+,>>,\hat{})$的子串 B^*,则 A 和 B 共享资源时可节省 1 个多路选择器.B^* 称为 A 和 B 的 MCES.

3　问题定义

程序的基本块 BB(Basic Block)以一个或多个有向无回路图 $G(V,E)$表示.$G(V,E)$是基本块的连接子图.集合 V 中的节点代表操作;集合 E 中的边 $e(u,v)$表示节点 u 产生的数据被节点 v 消费.$\mathrm{loglev}(v)$表示节点 v 在 $G(V,E)$中所处的逻辑级数.每个 $G(V,E)$与一个扩展图 $G^+(V\cup V^+,E\cup E^+P)$关联,$V^+$中的节点表示 $G(V,E)$的输入或输出.E^+中的边连接 V 和 V^+中的节点.V_{typical}表示应用的典型操作集合.计算模式 S 是 $G(V,E)$的子图,是扩展指令的最初表示形式.$A(S)$为 S 中所有节点的面积之和.$C(S)$为 S 的关键路径上的延时.$\mathrm{Occur}(S)$表示 S 在程序中被执行的次数.$T_{\mathrm{bef}}(S)$和 $T_{\mathrm{aft}}(S)$分别表示 S 以指令形式实现之前和之后的执行时间.增益函数 $M(S)$定义为 $\mathrm{Occur}(S)\times(T_{\mathrm{bef}}(S)/T_{\mathrm{aft}}(S)-1)$.

$N_{\mathrm{in}}(S)$和 $N_{\mathrm{out}}(S)$表示 S 的输入和输出数.N_{IN}和 N_{OUT}表示 S 中允许的最大输入和输出数.A_{MAX}和 C_{MAX}表示 S 允许的最大面积和最大延时.对于一个基本块,指令扩展问题可表述为:

问题 1　给定扩展图 $G^+(V\cup V^+,E\cup E^+)$,在下列约束下,找出一个组计算模式 $S_1,S_2,\cdots,S_N$,使得增益之和取得最大值:

(1) $N_{\mathrm{in}}(S_j)\leqslant N_{\mathrm{IN}}$, $N_{\mathrm{out}}(S_j)\leqslant N_{\mathrm{OUT}}$

(2) $A(S_j)\leqslant A_{\mathrm{MAX}}$, $C(S_j)\leqslant C_{\mathrm{MAX}}$

(3) S_j is convex,且 $\exists\, v\in V_{\mathrm{typical}}$ in S_j

上述问题定义基于文献[4]并作了必要修改.con

vex 条件要求 S 产生的结果不被 S 外的操作处理之后作为 S 的输入,是保证编译器能够正确调度单周期指令的必要前提.本文针对多周期指令同样引入 convex 约束以降低问题的复杂度.本文中,计算模式的面积和延时作为约束条件而不是引导函数.

4 算法实现

4.1 扩展指令识别

4.1.1 指令识别算法

在基本块的连接子图 G^+ 中,每个操作 p 在图中出现的位置用逻辑级数 loglev(p)表示.G^+ 中的典型操作作为初始搜索中心 center.与搜索中心有直接或间接数据相关性的节点称为生产者或消费者.搜索中心的数据相关图 DDG(Data Dependency Graph)由 3 种节点构成:生产者,消费者,及位于生产者和消费者之间与生产者和消费者直接或间接数据相关的节点.由于 DDG 之外的节点通常不在含搜索中心的关键路径上,搜索空间仅限于 DDG 内.搜索空间的边界由常数 L 界定.DDG 中节点的逻辑级数介于 loglev(center) − L 和 loglev(center) + L 之间.不难看出,最坏情况下 DDG 中的节点数随 L 的增加指数增加.

```
s←Φ, N_cut←0; violation←1;
if(first_step);
    S←S∪all nodes in search space;
else
    S←S∪output pattern from previous step;
while((violation)and(N_cut< # N of nodes in DDG));
    n←N_cut; N_cut←N_cut+1;
    set←Φ; S*←S; S_max←Φ;
    Comp_n←find all combinations that cut n nodes;
    ∀ i∈(1,Comb_n);
        V_cut←select n nodes from S*;
        S*←S*−V_cut;
        violation←CHECK(S*);
        if(! violation);
            set←set∪S*;
    ∀ S_i∈set;
        if(MERIT(S_i)≥MERIT(S_max));
            S_max←S_i;
if(S_max≠Φ);
        violation←0;
        return(S_max,MERIT(S_max));
else;
    violation←1;
```

图 1 以典型操作 p 为中心的搜索过程的一个搜索步骤

图 1 示出了以典型操作 p 为中心的计算模式识别过程的一个搜索步骤.每个步骤检查 DDG 中所有有效计算模式并从中找出增益值最大的计算模式.由于增益随计算模式复杂度的增加而增加,计算模式的初始值设为包含 DDG 中的所有节点可减少针对复杂 DFG 的搜索步骤.如果初始值违反约束,则从初始值中逐渐删除节点,直到剩余节点构成的子图满足约束.若当前步骤找到增益值最大的计算模式或 DFG 中的节点逐渐删除后只剩搜索中心,则一个搜索步骤结束.每个步骤产生的结果称为 p 的备选计算模式,并用一个复合节点替换作为下一步骤的搜索中心.每个搜索步骤结束后,当搜索中心无法再继续生长,即上一步骤的输出与当前步骤的输出相同时,以典型操作 p 为中心的搜索过程结束.最后一个搜索步骤的输出作为 p 的输出计算模式.

针对一个连接子图的搜索过程的结束条件为:图中所有节点均已被之前搜索步骤产生的输出计算模式覆盖,且针对图中所包含的每个类型的典型操作都至少进行了一个搜索步骤.结束条件有助于减少 DFG 中节点被以不同典型操作为中心的多个搜索步骤重复访问的次数,避免遗漏以执行时间较少的典型操作为中心的常用计算模式,避免产生局部最优结果.与一个连接子图相关的搜索步骤结束后,若某个节点被包含在输出计算模式集合中的多个计算模式中,则相交性检测过程仅将该节点被保留在增益值最大的计算模式中.

4.1.2 复杂度分析

假设 DDG 中节点的连接系数为 n,即每个节点最多有 n 个输入和 n 个输出.本文中 n 设为 2.图 3 中,while 循环的最大执行次数为 N_{while}.检查子图有效性的 CHECK 函数的复杂度计为 C_{check}.C_{check} 约为 5000 周期且近似为常数.最坏情况下除搜索中心外的所有节点均被删除.因此,最坏情况下一个搜索步骤的计算复杂度 C_{step} 为:

$$C_{\text{step}} = C_{\text{check}} \sum_{i=1}^{N_{\text{while}}} \left((1 - i/(2^{L+1}+1)) C_2^{i_{L+1}} + \sum_{j=1}^{i/(2^L-1)} \left(C_2^{j_L} C_{2^{L+1}-2j}^{i-j(2^L-1)} \right) \right) \quad (1)$$

则当 $L=2$ 且 $n=2$ 时,最坏情况下一个搜索步骤的计算复杂度约为:

$$C_{\text{step}} = C_{\text{check}} \sum_{i=1}^{11} \left((1 - i/9) C_8^i + \sum_{j=1}^{i/3} \left(C_4^j C_{8-3j}^{i-3j} \right) \right) = 3,115,000 \quad (2)$$

设每个连接子图中典型操作与所有操作的比例 r 为常数,操作总数为 N_{op},以每个典型操作为中心的搜索过程最多有 T 个搜索步骤,T 的值为 MIN($L_{\max}/2$, $(N_{\text{IN}}-1)/2$).则最坏情况下与一个连接子图中的所有典型操作相关的搜索过程的计算复杂度为:

$$C_{\text{worst}} = N_{\text{op}} \times r \times T \times C_{\text{step}} \quad (3)$$

GISEES 的扩展指令识别算法将计算模式限定在以典型操作为中心的数据相关图中,并采用步进式的搜索方式控制计算模式的生长方向有效裁地剪了设计空间.式(2)和(3)表明,指令识别算法的复杂度随 DFG 中节点数的增加线性增加.

4.2　扩展指令选择

4.2.1　预处理

指令识别过程的输出集合中存在图形同构或子图同构的计算模式需合并形成新的计算模式.由于匹配两个子图的复杂度随图中节点数的增加指数增加,图形同构和子图同构都是 NP 完全问题[23].本文基于 VF 算法[24]考虑序列的等价性以检测计算模式间的同构.虽然最坏情况下 VF 算法也具有指数复杂度,但由于计算模式的数量有限且复杂度较小(各种约束限制了计算模式的复杂度),相对于 4.1 中的识别算法和本节中的选择算法的开销,修改后的 VF 算法的开销可忽略不计.

假设合并前的集合中包含 3 个计算模式 S_1, S_2 和 S_3.同构计算模式的合并基于以下 3 个原则:(1) S_3 只与 S_1 同构. S_3 与 S_1 直接合并,且新计算模式的增益等于 S_3 与 S_1 的增益之和;(2) S_3 与 S_1 均同构, S_1 与 S_2 不同构,且 S_3 与 S_1 的增益之和大于 S_3 与 S_2 的增益之和. S_3 将与 S_1 合并, S_2 保持不变;(3) S_1, S_2 和 S_3 相互同构.若三者的最大公共子图(Maximal Common Subgraph)的增益大于任意 2 个计算模式的增益之和,则最大公共子图作为新的计算模式,否则合并增益值最大的 2 个计算模式.

4.2.2　指令选择算法

指令选择算法基于 MCES 的资源共享策略提高资源利用率,使给定资源上实现的扩展指令获得最大的性能提升.算法的输入是预处理过程的输出计算模式集合 $\Gamma=(S_1,S_2,\cdots,S_K)$. Γ 中每个计算模式 S_i 与采用深度优先搜索算法 DFS[25](Depth-First Search)构建的路径集合 P_i 关联,集合 $P_\Gamma=(P_1,P_2,\cdots,P_K)$ 表示集合 Γ 中的所有路径.给定面积和延时约束,从集合 Γ 中找出取得最大增益值的最优子集的问题可表示为:

问题 2　给定面积和延时约束 A_{MAX} 和 C_{MAX},计算模式集合 $\Gamma=(S_1,S_2,\cdots,S_K)$ 及路径集合 $P_\Gamma=(P_1,P_2,\cdots,P_K)$,找出具有 N_τ 个元素的子集 τ,使增益之和取得最大值.其中, $N_\alpha\leqslant N_\tau\leqslant N_\beta$.

N_β 和 N_α 分别表示子集 τ 中计算模式数的最大和最小值.经验参数 β 和 α 分别表示 τ 中计算模式间共享资源时可达到的面积压缩比的最大和最小值.其中,面积压缩比定义为 $(A_{\mathrm{bef}}-A_{\mathrm{aft}})/A_{\mathrm{aft}}$.本文中 α 和 β 的值分别设为 0.3 和 1.0. N_α 和 N_β 计算如下:

$$\sum_{j=1}^{N_\alpha}A(S_j)\leqslant\alpha A_{\max}<\sum_{j=1}^{N_\alpha+1}A(S_j)\tag{4}$$

$$\sum_{j=1}^{N_\beta}A(S_{K-j+1})\leqslant\beta A_{\max}<\sum_{j=1}^{N_\beta+1}A(S_{K-j+1})\tag{5}$$

指令选择算法的实现过程如下.首先,从集合 Γ 中选出前 $i(i\in(N_\alpha,N_\beta))$ 个增益值最大的计算模式形成子集 τ 及其关联的路径集合 P_τ.接着,对 τ 中计算模式的面积进行压缩使资源在计算模式间充分共享.压缩过程的输出结果为由 1 个或多个复合计算模式组成的集合 τ^*.如果集合 τ^* 中所有计算模式的面积和大于 A_{MAX},则丢弃集合 τ^*,否则将其包含在计算模式子集的集合 ψ 中.最后,从集合 ψ 中选出 1 个取得最大增益值的子集 $\tau^*_{\max}$作为指令选择算法的最终输出.

图 2(a)示出了计算模式子集 τ 的面积压缩过程.首先,从 P_τ 中找出所有路径的最大公共等价子串 mces.包含 mces 的计算模式的集合记为 τ_{mces},其关联的路径集合为 P_{mces}.接着,将 P_τ 中包含 mces 的路径压缩到复合计算模式 S_{comp}中并插入必要的多路选择器. τ_{mces}和 P_{mces}分别从 τ 和 P_τ 中删除,之后 mces 从 P_{mces}中删除. P_{mces}中的剩余路径采用相同的方式压缩.压缩 P_{mces}的过程一直进行直到 S_{comp} 违反延时约束或 P_{mces}中的路径间没有再共享资源的可能.然后,针对更新后的 τ 和 P_τ,寻找 mces 和压缩 P_{mces}的过程重复进行直到 τ 为空集.

两条路径的 MCES 寻找过程分为 2 个阶段.首先,找出路径的所有等价路径.接着,采用广义后缀树[26](Generalized Suffix Tree)方法寻找等价路径的 MCES.本文中,每条路径中最多允许插入 1 个操作.图 2(b)示出了路径 $A(-,\&,+,+,>>,\hat{},>>)$和 $B(+,\hat{},+,+,\hat{},<<)$的等价路径的寻找过程.首先,找出两条路径的插入操作集合 V_{insert}.

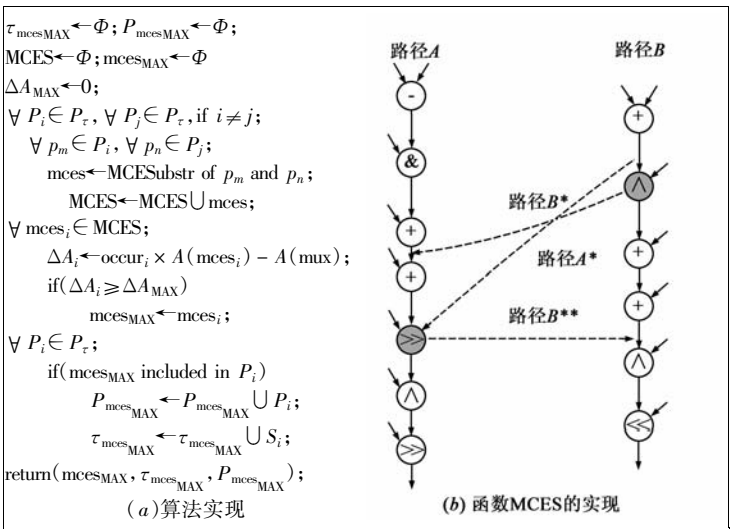

图 2　考虑资源共享的指令选择算法

接着,将 V_{insert}中的操作插入到对方路径形成等价路径. V_{insert}中的操作需满足下列 3 个等价条件:(1)面积开销小于 4:1 多路选择器;(2)不处于路径的开始或结尾处;(3)插入到对方路径后具有相同的上下文. 如图 2(b)中,V_{insert}中包含操作 $SHR_{(0,4)}$和 $XOR_{(1,1)}$,插入到对方路径对应位置后形成等价路径集合 $P_{E1}(A, A^*)$和 $P_{E2}(B, B^*, B^{**})$. 不难看出 P_{E1}和 P_{E2}中路径的所有组合中,组合(A, B^{**})的 MCES 最长,为(+,+,>>,^).

4.2.3 复杂度分析

设 Γ 中每个计算模式最多有 p_{max}条路径,每条路径最多包含 l_{max}个节点. 对于子集 τ,外层 while 循环的循环次数最多为 τ 中计算模式的个数 i,内层 while 循环最多循环 $i \times p_{max}$次. 寻找两条路径的 mces 的复杂度近似为 $O(m+n)$,其中 m 和 n 分别为两条路径中的节点数. 最坏情况下,寻找 $i \times p_{max}$条路径的 mces 的复杂度近似为$(i \times p_{max})2 \times O(l_{max})$. 因此,最坏情况下从计算模式集合 Γ 中寻找最优子集 τ 的复杂度为:

$$\sum_{i=N_\alpha}^{N_\beta} (i \times (i \times p_{max})^3 \times O(l_{max})) \approx O(N_\beta^4 p_{max}^3 l_{max}) \quad (6)$$

设 K、N_β、p_{max}、l_{max}的值分别为 32、16、4、8,则最坏情况下扩展指令选择算法的计算复杂度约为 33,554,432. 从式(2)、式(3)和式(6)可以看出,相对于扩展指令识别算法,扩展指令选择算法的复杂度相对较小,因而 GISEES 方法的计算复杂度取决于扩展指令识别算法的计算复杂度.

5 实验结果

5.1 实验设置

本文中,测试程序 wcdma 采用 Michigan 大学的 W-CDMA 测试程序[27],其它测试程序选自 mediabench[28]. 本文对测试程序做了适当的优化以减少基本块的数量. 对于每个测试程序,典型操作集合包含占总执行时间 95%的所有 ALU 操作类型,输入基本块是指占关键函数的总执行时间 95%的包含典型操作的基本块,关键函数是指占总执行时间 95%的所有函数. 应用的复杂度用基本块中的平均节点数表示. 表 1 示出了不同测试程序的典型特征. 其中,操作"->"代表符号转换或数据类型转换.

本文的工作嵌入在 Trimaran 环境[29]中. 应用的 DFG 基于 Trimaran 产生的经编译优化但未经指令调度和寄存器分配的中间表示 IR(Intermediate Representation)构建. 本文对 Trimaran 做了适当扩展使得扩展指令支持 8 输入和 4 输出. 扩展指令的输入、输出限定为寄存器型. 设通用寄存器的数量为 32,cache 的命中率为 100%,操作具有固定的面积和延时开销. 表 2 列出了常用操作在 Xilinx V5 FPGA 上实现时的面积和延时信息.

表 1 不同测试程序的典型特征

程序	复杂度	输入块	典型操作
cjpeg	21.3	14	+,<<,cmp,>>,-,*
djpeg	24.0	18	+,>>,*,<<,cmp
mpeg2enc	8.1	32	+,||,*,cmp,>>,&
mpeg2dec	5.5	28	+,cmp,-,>>
gsmenc	16.1	26	+,*,<<,-,cmp,->
gsmdec	8.4	21	+,cmp,->,>>,*
wcdma	11.1	33	+,*,cmp,<<,^,-

表 2 常用操作的面积和延时信息

操作	位宽	面积(LUT)	延时(ns)
+	32	32	2.24
*	18	280	6.67
>>	32	16	2.20
^	32	7	0.90
MUX 4:1	32	32	0.90
MUX 8:1	32	64	1.08

5.2 实验结果

表 3 列出了 N_{IN}和 N_{OUT}为 8 和 2,面积和延时约束为 840 LUT 和 30ns 时,GISEES 针对不同测试程序的执行信息,如 MSTN(Maximal Steps for a Typical Nodes),SMB(Steps for Maximal BB),TNMB(Typical Nodes in Maximal BB),NMB(Nodes in Maximal BB)和 TN(Typical Nodes)等. 表 3 显示,对于不同程序,最大基本块中典型操作的比例在 0.57~0.85 之间,输入基本块中典型操作的比例在 0.43~0.68. 对于所有程序,总搜索步骤与输入基本块中典型操作的数量近似成线性关系并小于典型操作的数量. 以一个典型操作为中心的搜索过程的最大搜索步骤数均为 3,与应用的复杂度无关,从而避免了算法的复杂度随应用复杂度的增加指数增加.

表 3 针对不同的测试程序 GISEES 的执行情况

程序	Steps	MSTN	SMB	TNMB	TN	NMB	Nodes
cjpeg	102	3	42	57	194	76	298
djpeg	115	3	51	66	184	92	432
mpeg2enc	42	3	14	28	127	33	259
mpeg2dec	20	3	3	11	101	17	154
gsmenc	106	3	54	160	285	202	418
gsmdec	45	3	4	9	114	12	176
wcdma	59	3	12	34	163	60	367

表 4 列出了 GISEES 与 scalable 算法[7]的运行结果的比较. 设输入、输出约束分别为 8 和 2,面积和延时约束分别为 840 LUT 和 30ns,GISEES 方法 scalable 算法均不考虑分支和存储操作. 对于所有测试程序,由于 GISEES 方法允许指令间共享资源,scalable 算法输出的扩展指令集 Γ_1 是 GISEES 产生的扩展指令集 Γ_2 的子集. 表 4 表明,GISEES 的执行时间最大约为 200M 周期,

GISEES的效率是 scalable 算法的 2.49x ~ 32.50x.并且,由于 GISEES 可以显著减少产生有效计算模式的数量和 DFG 中节点被重复访问的次数,对于基本块较大的程序(如 cjpeg,djpeg)GISEES 的执行效率更高.

表 4　两种指令识别算法产生结果的比较

程序	方法	有效计算模式	输出计算模式	执行周期	加速比
cjpeg	GISEES	342	13	143,677,506	32.50
	scalable	819	9	4,668,842,815	
djpeg	GISEES	361	18	206,910,760	27.67
	scalable	962	11	5,725,656,330	
mpeg2enc	GISEES	124	12	32,445,840	11.83
	scalable	234	9	383,672,058	
mpeg2dec	GISEES	82	10	12,160,960	2.49
	scalable	120	7	30,326,394	
gsm2enc	GISEES	205	20	120,189,160	15.47
	scalable	437	15	1,859,696,118	
gsm2dec	GISEES	105	14	19,848,780	2.65
	scalable	152	10	52,616,088	
wcdma	GISEES	263	24	59,730,125	20.74
	scalable	504	17	1,238,711,208	

图 3 示出了对于不同测试程序 GISEES 的计算复杂度与测试程序的复杂度之间的关系.图 3 表明,测试程序的 ASTN(Average Steps for a Typical Node)和 ASTNMB(Average Steps for a Typical Node in Maximal BB)均小于 1,且随测试程序复杂度增加近似成线性增加.CDNMB(Cycles Divided by Nodes in Maximal BB)随测试程序的基本块的规模变化不大,CDANB(Cycles Divided by Average Nodes in BB)随着基本块的规模的增加近似成线性增加.表 4 和图 3 表明,GISEES 方法具有线性复杂度,从根本上解决了现有算法的计算复杂度随着应用复杂度的增加指数增加的问题.

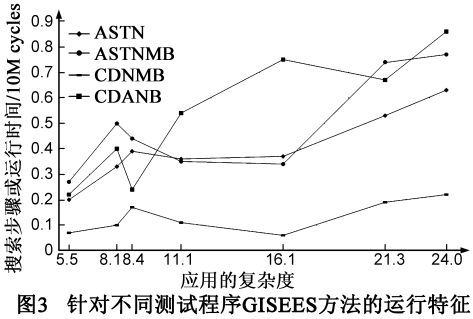

图3　针对不同测试程序GISEES方法的运行特征

图 4 示出了输入操作数约束对程序获得的加速比的影响.由于 N_{OUT}大于 1 时其值对加速比的影响较小,N_{OUT}固定设为 2.A_{MAX}和 C_{MAX}分别为 840 LUT 和 30ns.图 4 显示,当输入操作数小于 6 时,每个程序获得的加速比都随输入操作数的增加而增加,基本块较大的程序(如 cjpeg,djpeg)或基本块的逻辑级数相对较多的程序(如 gsmenc)的加速比随输入操作数的增加增幅较大.

当输入操作数为 3 时,所有测试程序均可获得显著的性能加速比(17% ~ 48%),基本块相对较小且扩展指令数相对较多的程序(如 gsmenc)获得的性能加速比最大.因此,3 输入的扩展指令能够用较少的资源获得较大的性能提升,因而效率相对较高.当输入操作数为 4 时,所有测试程序的性能加速比的增幅开始下降,基本块较小的程序(如 mpeg2dec,gsmdec)的加速比的增幅下降更明显并趋于饱和.因此,对于基本块较小的测试程序,4 输入操作数的扩展指令可以充分加速应用的执行.当输入操作数达到 6 时,cjpeg 和 djpeg 的加速比随输入操作数的增加继续增加,但增幅明显减小,其它程序的加速比均达到饱和.因此,6 输入操作数足以满足通过增加扩展指令获得性能提升的需要.

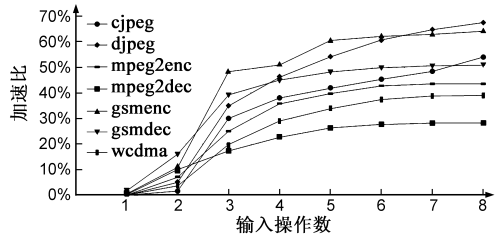

图4　输入操作数约束对加速比的影响

图 5 示出了面积约束对测试程序的加速比的影响.N_{IN}和 N_{OUT}分别为 8 和 2,延时约束 C_{MAX}为 30ns.图 5 中虚线(实线)表示扩展指令间允许(不允许) 共享资源时不同测试程序获得的性能加速比.图 5 中实线显示,扩展指令间不允许共享资源时,所有测试程序的加速比随着面积的增加近似成线性缓慢增加,当面积约束 A_{MAX}值为 1120 LUT 时,除 mpeg2dec 外其它测试程序的加速比均未达到饱和.图 5 中虚线显示,当面积约束 A_{MAX} 的值为 840 LUT 时,cjpeg, mpeg2enc, mpeg2dec, gsmdec 的加速比接近饱和,其它程序的加速比随面积的增加增幅明显下降.A_{MAX}的值为 1120 LUT 时,所有程序的加速比接近饱和.

扩展指令间允许共享资源时,所有测试程序的加速比随着面积的增加明显增加,且当面积约束 A_{MAX}的值在 280 LUT-560 LUT 之间时,除 mpeg2dec 外,其它测试程序的加速比显著增加.这是因为除 mpeg2dec 外的其它程序均包含了大量的乘法操作,当 A_{MAX}的值大于 280

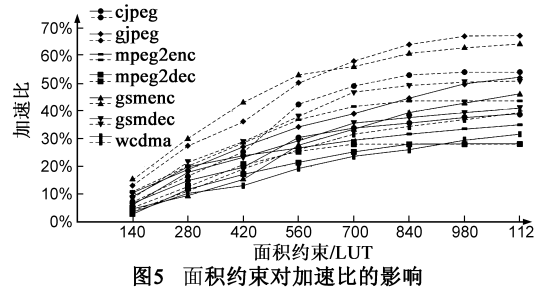

图5　面积约束对加速比的影响

LUT时,面积开销较大的乘法操作可以被包含在扩展指令中,并且乘法操作在扩展指令间的共享显著增加了GISEES的资源共享策略达到的面积压缩比.比较图5中的虚线和实线可以看出,由于扩展指令间共享资源可以使得在同样的面积上实现更多的扩展指令,GISEES的资源共享策略可以显著提高测试程序获得的性能加速比.例如,当 A_{MAX}的值为840 LUT时,图5中虚线所示的各程序的加速比与实线所示的各程序的加速比的比值平均为1.35,最大为1.52.

表5列出了GISEES的MCES资源共享策略与文献[19]中的基于最长公共子序列(LCS)的路径压缩策略的比较结果.列2~5中,前者表示获得的面积压缩比,后者表示插入的多路选择器在扩展指令的关键路径上产生的延时开销.从表5可以看出,面积约束 A_{MAX}的值较小时,对于不同的测试程序两种策略获得的面积压缩比相当;A_{MAX}的值较大时(如 A_{MAX} = 840 LUT),对于所有测试程序,GISEES获得的面积压缩比均大于或等于LCS策略获得的面积压缩比.对于基本块较大的测试程序(如cjpeg,djpeg)或扩展指令数相对较多且指令的逻辑级数相对较大的测试程序(如gsmenc),GISEES获得的面积压缩比明显高于LCS策略获得的面积压缩比.

表5 两种资源共享策略的压缩比及其产生的延时开销

程序	方法	面积约束(LUT)			
		280	560	840	1120
cjpeg	GISEES	0.55,1.80ns	0.77,2.88ns	0.84,3.96ns	0.85,4.50ns
	LCS	0.57,1.98ns	0.75,3.06ns	0.74,4.14ns	0.72,4.86ns
djpeg	GISEES	0.66,1.80ns	0.82,2.88ns	0.90,4.14ns	0.87,4.68ns
	LCS	0.70,2.16ns	0.78,3.60ns	0.82,4.68ns	0.80,5.04ns
mpeg2enc	GISEES	0.32,0.90ns	0.37,1.80ns	0.43,2.88ns	0.46,2.88ns
	LCS	0.40,1.08ns	0.42,2.70ns	0.43,2.88ns	0.42,3.60ns
mpeg2dec	GISEES	0.47,0.90ns	0.48,1.80ns	0.52,2.16ns	0.51,2.70ns
	LCS	0.43,0.90ns	0.48,2.70ns	0.50,3.06ns	0.50,3.06ns
gsm2enc	GISEES	0.49,2.16ns	0.53,2.70ns	0.72,3.60ns	0.74,3.60ns
	LCS	0.46,2.88ns	0.55,3.06ns	0.66,3.96ns	0.67,3.96ns
gsm2dec	GISEES	0.36,1.98ns	0.45,2.88ns	0.62,3.06ns	0.64,3.06ns
	LCS	0.38,2.16ns	0.53,2.88ns	0.60,3.06ns	0.61,3.78ns
wcdma	GISEES	0.53,2.16ns	0.65,3.06ns	0.66,3.60ns	0.69,3.96ns
	LCS	0.57,2.70ns	0.68,3.78ns	0.65,4.14ns	0.63,4.50ns

同时表5显示,A_{MAX}取任意值时,对于所有的测试程序GISEES的资源共享策略在扩展指令的关键路径上产生的延时开销均小于LCS策略产生的开销.这是因为GISEES的资源共享策略允许在路径上插入简单的操作形成等价公共子串,从而减少了插入的多路选择器的数量,减少了多路选择器带来的延时和面积开销.

6 结论

本文提出了一种快速的扩展指令集自动产生方法GISEES.该方法中的以应用特征为中心的快速扩展指令识别算法可有效裁剪设计空间,降低算法的计算复杂度,使算法的复杂度与应用的复杂度成线性关系,实现了设计空间的快速开发.同时,该方法中的扩展指令选择算法采用基于最大公共等价子串的资源共享策略,允许改变扩展指令的数据流结构,提高了资源利用率,减小了插入的多路选择器产生的面积和延时开销,可使给定资源上实现的指令获得最大的性能提升.实验结果表明,该方法更适合在嵌入式系统上运行,为嵌入式系统定制高效的扩展指令集.

由于Trimaran环境的限制,GISEES方法产生的扩展指令的执行情况无法进行周期精确的模拟,无法精确模拟cache的命中率和寄存器文件带来的影响.同时,本文假定添加扩展指令前后处理器的指令流出数均为1,简化了编译调度对扩展指令获得的性能加速比的影响.因此,在下一步工作中,我们将克服Trimaran的环境限制或在其它平台上研究存储层次和编译调度对GISEES方法产生的扩展指令所获得的性能提升的影响.此外,为嵌入式系统定制面向应用的高性能低功耗的扩展指令还需要解决如扩展指令的编/解码,扩展指令的数据供给,扩展指令与基本指令间的数据交互等重要问题,这些也将是我们下一步工作的内容.

参考文献

[1] Ricardo E. Gonzalez. Xtensa: A configurable and extensible processor[J]. IEEE Micro, 2002, 20(2): 60 – 70.

[2] Kurt Keutzer, et al. From ASIC to ASIP: the next design discontinuity[A]. Proceedings of ICCD'02[C]. Washington: IEEE Computer Society Press, 2002. 43 – 57.

[3] Cesare Alippi, et al. A DAG-based design approach for reconfigurable VLIW processors[A]. Proceedings of DATE'99[C]. New York: ACM Press, 1999. 778 – 779.

[4] Kubilay Atasu, et al. Automatic application-specific instruction-set extensions under micro-architectural constraints[A]. Proceedings of DAC'03[C]. New York: ACM Press, 2003. 256 – 261.

[5] Nathan Clark, et al. Processor acceleration through automated instruction set customization[A]. Proceedings of MICRO'03[C]. Washington: IEEE Computer Society Press, 2003. 129 – 140.

[6] Fei Sun, et al. A scalable application-specific processor synthesis methodology[A]. Proceedings of ICCAD'03[C]. Washington: IEEE Computer Society Press, 2003. 283 – 290.

[7] Pan Yu, Tulika Mitra. Scalable custom instructions identification for instruction set extensible processors[A]. Proceedings of CASES'04[C]. New York: ACM Press, 2004. 69 – 78.

[8] Jason Cong, et al. Application-specific instruction generation for configurable processor architectures[A]. Proceedings of FPGA'

04[C]. New York: ACM Press, 2004. 183 – 189.
[9] Partha Biswas, et al. Introduction of local memory elements in instruction set extensions[A]. Proceedings of DAC'04[C]. New York: ACM Press, 2004. 729 – 734.
[10] Laura Pozzi, et al. Exploiting pipelining to relax register-file port constraints of instruction-set extensions[A]. Proceedings of CASES'05[C]. New York: ACM Press, 2005. 2 – 10.
[11] Jason Cong, et al. Instruction set extension with shadow registers for configurable processors[A]. Proceedings of FPGA'05[C]. New York: ACM Press, 2005. 99 – 106.
[12] Theo Kluter, et al. Speculative DMA for architecturally visible storage in instruction set extensions[A]. Proceedings of CODES + ISSS'08[C]. New York: ACM Press, 2008. 243 – 248.
[13] Ramkumar Jayaseelan, et al. Exploiting forwarding to improve data bandwidth of instruction-set extensions[A]. Proceedings of DAC'06[C]. New York: ACM Press, 2006. 43 – 48.
[14] Ya-shuai Lü, et al. Customizing computation accelerators for extensible multi-issue processors with effective optimization techniques[A]. Proceedings of DAC'08[C]. New York: ACM Press, 2008.
[15] 吕雅帅,等. 面向嵌入式应用的指令集自动扩展[J]. 电子学报, 2008, 36(5): 985 – 988.
Lü Ya-shuai, et al. Automatic instruction set extension for embedded applications[J]. Acta Electronica Sinica, 2008, 36(5): 985 – 988. (in Chinese)
[16] Quang Dinh, et al. Efficient ASIP design for configurable processors with fine-grained resource sharing[A]. Proceedings of FPGA'08[C]. New York: ACM Press, 2008. 99 – 106.
[17] Alauddin Alomary, et al. An ASIP instruction set optimization algorithm with functional module sharing constraint[A]. Proceedings of ICCAD'93[C]. Los Alamitos: IEEE Computer Society Press, 1993. 526 – 532.
[18] Bernardo Kastrup, et al. ConCISe: a compiler-driven CPLD-based instruction set accelerator. [A]. Proceedings of FCCM'99[C]. Washington: IEEE Computer Society Press, 1999. 92 – 101.
[19] Philip Brisk, et al. Area-efficient instruction set synthesis for reconfigurable System-on-Chip designs[A]. Proceedings of DAC'04[C]. New York: ACM Press, 2004. 395 – 400.
[20] Marcela Zuluaga, Nigel Topham. Resource sharing in custom instruction set extensions[A]. Proceedings of IEEE Symposium on Application Specific Processors[C]. Anaheim: IEEE Computer Society Press, 2008. 7 – 13.
[21] E M Witte, et al. Applying resource sharing algorithms to ADL-driven automatic ASIP implementation[A]. Proceedings of ICCD'05[C]. Washington: IEEE Computer Society Press, 2005. 193 – 199.
[22] Hai Lin, Yunsi Fei. Resource sharing of pipelined custom hardware extension for energy-efficient application-specific instruction set processor design[A]. Proceedings of ICCD'09[C]. Piscataway: IEEE Press, 2009. 158 – 165.
[23] David Eppstein. Subgraph isomorphism in planar graphs and related problems[A]. Proceedings of 6th annual ACM-SIAM symposium on Discrete algorithms[C]. Philadelphia: Industrial and Applied Mathematics Society, 1995. 632 – 640.
[24] Luigi P. Cordella, et al. A (sub)graph isomorphism algorithm for matching large graphs[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2004, 26(10): 1367 – 1372.
[25] Robert Tarjan. Depth-first search and linear graph algorithms[A]. Proceedings of 12th Annual Symposium on Switching and Automata Theory[C]. Washington: IEEE Computer Society Press, 1971. 114 – 121.
[26] E Ukkonen. On-line construction of suffix trees[J]. Algorithmica, 1995, 14(3): 249 – 260.
[27] Mark Woh, et al. The next generation challenge for software defined radio[A]. Proceedings of International Symposium on Systems, Architectures, Modeling and Simulation[C]. New York: Springer-Verlag Berlin Heidelberg, 2007. 343 – 357.
[28] Chunho Lee, et al. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems[A]. Proceedings of MICRO'93[C]. Washington: IEEE Computer Society Press, 1993. 330 – 335.
[29] Bhuvan Middha, et al. A Trimaran based framework for exploring the design space of VLIW ASIPs with coarse grain functional units[A]. Proceedings of 15th international symposium on System Synthesis[C]. New York: ACM Press, 2002. 2 – 7.

作者简介

陈　虎　男,1983 年 11 月生于安徽省淮南市. 现为国防科技大学计算机学院博士研究生. 主要研究方向为面向应用的处理器体系结构及其相关软/硬件协同设计技术.
E-mail: chenhu@ieee.org

陈书明　男,1961 年 4 月生于安徽省六安市. 1993 年毕业于国防科技大学计算机系,获工学博士学位. 现为国防科技大学计算机学院教授,博士生导师. 主要研究方向为高性能微处理器设计和超深亚微米 VLSI 设计理论与技术.　E-mail: smchen@nudt.edu.cn