基于关联性启发的自定义指令选择算法①

薛　辉　周学海 (中国科学技术大学 计算机科学与技术系 安徽 合肥 230027;
中国科学技术大学苏州研究院 嵌入式系统重点实验室 江苏 苏州 215123)

摘　要： 面向特定应用的自定义指令可以减小可执行代码的长度，提高执行效率和降低系统功耗。候选指令选择在自动指令集扩展问题上占用相当重要的作用，它直接影响了指令扩展的性能和效率。已有的启发式选择算法虽然有较优的时间性能，但在时间性能和选择结果上还有改进的空间。由此，提出了一种基于扩展指令间关联性的启发式算法，实验结果证明，本算法能快速有效的找到比现有启发式算法更优的候选指令组合。

关键词： 启发式算法 关联性 指令集扩展 指令选择

Relevance Heuristic-Based Custom Instruction Selection Algorithms

XUE Hui, ZHOU Xue-Hai
(Deparment of computer Science and Technology, University of Science and Technology of China, Hefei 230027, China;Suzhou Instiute for Advanced Study, University of Science and Techonlogy of China, Suzhou 215123, China)

Abstract: Application-specific custom instructions can reduce the length of exectutable code, improve efficiency and reduce system power consumption. Instruction at the candidate selection plays an important role in automatice instruction set extension. It influences the performance and efficiency of the instruction extension directly. Although the existing heuristic algorithm has better time performance, it still has room for improvement in time performance and selection result. Thus, this paper proposes a heuristic algorithm based on the directive expansion of inter-relevance. Experimental results show that this algorithm can be quicker and more effective than the existing heuristic algorithm in finding better combinations of the candidate instructions.

Keywords: heuristic algorithm; relevance; instruction set extension; instruction selection;

1　引言

随着数字多媒体的普及，特别是带有视频，音频的应用是的面向特定应用处理器(Application Specific Instruction-set Processor，简称 ASIP)已不能满足多媒体处理需求。而设计专用的 ASIC(Application Specific Integrated Circuit)来满足需要往往需要太多的人力物力以及很长的上市周期(Time-to-Market，简称 TTM)，因而支持可扩展指令的 ASIP 处理器应运而生。

可扩展指令集 ASIP 处理器可以根据特定应用的具体特征将频繁执行的指令簇转化为少量的扩展指令，在自定义的处理逻辑单元上执行，从而实现性能的提升。目前已经有多种支持可扩展指令的 ASIP，如 Altera 的 NIOS[1]，Tensilica 的 Xtensa[2]等。指令集扩展算法的目标就是找出可作为自定义指令执行的指令簇。指令扩展算法一般被划分为两个子问题:(1)自定义指令的识别——找出可被映射为自定义指令的指令簇；(2)自定义指令的选择——从被识别的候选自定义

① 基金项目:安徽省自然科学基金(070412030),国家高技术研究发展计划(863)(2008AA01Z101)
收稿时间:2009-05-19

指令集合中选出符合资源约束或者是其他约束的最优子集。本文的研究工作主要针对后者进行。

自定义指令的选择问题在可扩展指令集处理器的研究中占有相当重要的地位，选择的自定义指令集合的好坏直接影响到最终的效率和性能提升。PanYu[3]等用启发式的算法来选择自定义指令集合,并给出了基于性价比、执行时间、加速比的三种启发函数。纪金松[4]指出已有的算法求解问题时忽视了指令本身和指令实例之间的关系，提出了结合贪心策略的启发式算法以及基于遗传进化的选择算法。但是纪金松[4]提出的算法忽视了已选取指令和待选取指令之间存在的联系。针对以上不足，本文改进了启发式算法，该算法与已有算法相比，可更快速地获得一个更优的结果。

本文划分为四个部分，第一部分为简介和相关工作介绍，第二部分给出了指令选择问题的形式化定义和实例解释，并提出 RelevanceHeur 算法，给出了该算法的描述，第三部分说明实验环境及实验结果，并与其他算法进行了分析比较，最后对所做工作进行了总结。

2 指令选择问题及算法描述

2.1 问题描述

令候选自定义指令为$C_1,\cdots,C_n$。在程序的代码序列中，不同位置出现的相同自定义指令称为自定义指令的实例。假设自定义指令C_i有n个实例，分别为$c_{i.1},\cdots,c_{i.n_i}$,每个实例在运行的过程中被执行的频率为$f_{i.j}$。令A_i是自定义指令C_i的面积需求(面积的意思为指令占用的硬件资源数)，P_i是选中C_i所获得的性能提高(将C_i使用硬件实现时，相比软件实现所获得的性能提升，以时钟数的形式给出)，A为面积上限，M为自定义指令数目上限;二进制变量$s_{i.j}$表示是否选中自定义指令的实例(如果在实例$c_{i.j}$上选中自定义指令，则$s_{i.j}$为1，否则为0；S_i为是否选中自定义指令C_i；自定义指令的选择问题可以抽象为如下多约束的最优化问题:

$$\max:\sum_{i=1}^{N}\sum_{j=1}^{n_i}(s_{i,j}\times P_i\times f_{i,j})$$

$$s.t.\quad \forall i,j\ \sum_{n=1}^{k}s_{i_{n\cdot j_n}}\le 1,\quad \text{iff}\ \bigcap_{n=1}^{k}c_{i_{n\cdot j_n}}\neq\phi$$

$$\sum_{i=1}^{N}(S_i\times A_i)\le A,\quad S_i=s_{i.0}\,|\,s_{i.1}\,|\cdots|\,s_{i.n_i};$$

$$\sum_{i=1}^{N}S_i\le M,\quad S_i=s_{i.0}\,|\,s_{i.1}\,|\cdots|\,s_{i.n_i};$$

选择的自定义指令除了必须符合面积和指令数目限制外，还必须需满足覆盖集约束。不同自定义指令实例可能包含相同代码序列位置的原指令实例，这样就存在自定义指令实例间的互斥，也就是这些自定义指令实例中只能有一个被选中。自定义指令实例的执行频率$f_{i.j}$可根据剖析信息中基本块的指令频率得到，自定义指令的面积需求A_i和性能提升P_i可以从相应的硬件库中抽取。A_i以包含的原指令所占用的加法器个数作为基本单位。P_i由分别通过软件和硬件执行自定义指令所需要的时间比值确定。

2.2 关联启发选择算法

假设待选自定义指令个数为 n，首先将自定义指令集合分为待选集合C和结果集合R。初始时$C:=\{C_1,C_2,...,C_n\}$，$R:=\{\Phi\}$。从待选集合中选取自定义指令时，R中的自定义指令与C中的自定义指令之间可能存在着互斥实例，互斥实例的存在将影响下一次选入的自定义指令对的性能提升，而[3, 7]中启发函数每次选取自定义指令时却忽视了这种关联性，仅仅基于待选集合C本身选取指令。

算法 1.为关联选择算法伪码描述，我们用数组$Perf[n]$记录C和R的关联性，数组的第 i 项$Perf[i]$代表自定义指令C_i对R所能产生的性能增益。二维数组$Penality[n][n]$保存指令间由于互斥实例造成的性能损失，例如$Penality[i][j]$表示同时选入C_i和C_j时，由于互斥的实例C_j对C_i造成的性能损失。初始时$Perf[n]$设置为每一个C_i所有实例的加速比之和。$GetBestPerfCI()$中的启发函数选取$Perf[n]$中最大值项并返回相应的加速值。然后将该指令加入结果集合，更新结果集合的面积，加速比，以及包含的自定义指令个数。

每次C_k加入R后，C中与C_k互斥指令实例都需要被删除，$Perf[n]$数组也需做相应的更新。这些更新都由函数$UpdateEffet()$完成。算法 2 为$UpdateEffet()$函数的伪码描述。其中对于$Perf[n]$和$Penality[n][n]$的动态更新是本算法的核心部分，由于评价信息的动态更新，省去了每次重新选择最优指令所需的大量重复计算。当从待选集合C中选择一条自定义指令C_i加入结果

```
selection(A_max, M_max)
begin
  //已选集合初始化
  R:={Φ}
  //数组初始化
  init array Perf,Penality
  //当选入指令超出面积约束时退出
  loop(until M_r < M_max)
  begin
    //得到当前加速比最高的指令
    <perf,i>:=GetBestPerfCI()
    //i=-1表示待选集合为空
    if i=-1 then
      candidates set empty and quit loop
    end if
    //选入当前指令是否违反面积约束
    if A_r < A_i then
      //从待选集合中排除该指令
      Perf[i]=-1
      next loop
    end if
    //将该条指令加入已选指令集合
    R:={R}∪{C_i}
    //更新已选集合的占用面积，指令书加速比
    A_r-=A_i;M_r+=M_r+1;P_r+=perf;
    //更新信息
    UpdateEffect(i)
  end loop
end
```

算法 1　关联选择算法

集合 R 后，就需要重新检查 C，将其中所有与 C_i 互斥的实例删除。由于被删除的实例可能同时与其他自定义指令的实例互斥，所以当该实例被删除后，与其互斥的指令实例就减少了一个互斥对象(一个指令实例可能会与多个指令实例互斥，或者多个指令实例共同互斥)。假设自定义指令 C_j 中存在与自定义指令 C_i 互斥的实例 c_{jk}。c_{jk} 与 C_m 中实例 c_{ml} 互斥，当 c_{jk} 被删除后，则 C_m 与 C_j 互斥的指令实例使 C_m 的性能损失就减少了，所以更新数组项 $Penality[m][j]$。最后更新每一个 C_i 对 R 的性能提升值 $Perf[j]$，这只需要减去 C_j 中与 C_i 互斥的实例加速比，其值保存在数组项 $Penality[j][i]$ 中。

```
UpdateEffect(i)
begin
  //遍历所有待选指令
  foreach(unselected custom instruction C_j)
    //如果C_j指令实例为空
    if all C_j's instance set is empty then
      next unselected custom instruction
    end if
    //C_j中与C_i互斥的指令实例放入CII
    CII ⇐ all C_j's insatnces mutex with C_i
    //检查CII中所有互斥实例c_jk
    foreach(instance c_jk in CII)
      //在C_i中删除与c_jk互斥实例c_ik
      Delete c_ik in C_i
      //检查其他待选指令C_m中与c_jk互斥的实例
      foreach(c_mn ∈ C_m ∧ c_mn mutex with c_jk)
        //减少指令C_m由于与c_jk互斥造成的损失
        Penality[m][j]-=f_mn·p_m
      end for
    end for
    //更新加速比
    Perf[j]-=Penality[j][i]
  end for
end
```

算法 2　关联更新

3　算法分析及实验结果

3.1 算法复杂度分析

设候选指令实例数目为 n，候选指令数目为 m。函数 $GetBestPefCI()$ 选择使当前集合性能提升最高的自定义指令，其时间复杂度为 T_{sel}。函数 $UpdateEffect()$ 中删除候选指令集合中与结果集合互斥的实例，并更新 $Perf[n]$、$Penality[n][n]$，其时间复杂度为 T_{update}。$GetBestPefCI()$ 中搜索一遍 $Perf[n]$ 选中最高加速比的指令，所以 T_{sel} 时间复杂度均为 $O(n)$。$UpdateEffect()$ 通过预先的处理，计算出候选集合中与当前选入集合指令的互斥实例集合，然后删除互斥实例并更新 $Penality[n][n]$。假设指令 i 为当前选入结果集合的指令，与候选集合中指令 j 互斥的平均概率为 α (经验表明 $\alpha < 0.2$)，i 与 j 互斥集合的大小为 $|M_1|$。实例 $k \in M_1$，候选集合中与实例 k 互斥的实例个数为 $|M_2|$。

$|M_1| < n/m$，$|M_2| < m$，而每次调用 $UpdateEffect()$ 外层循环的执行次数是递减的，所以 T_{update} 的时间复杂度 $\le O(\alpha \times m \times n)$。

3.2 实验环境

实验环境构建在 Linux 平台下，选择 Altera 的 NIOS II 指令集为基本指令集，Mediabench[5]作为测试程序,所有的 GNU 工具链均交叉编译到目标体系结构上。实验机器配置为 AMD Athlon(tm)1GHz，384M 内存，编译器采用 gcc version 3.4.1 (Altera Nios II 7.2b152),编译时一律采用-O3 优化选项。

首先用 GCC 编译出测试程序可执行代码,然后通过 BuildDFG(Perl 脚本)标记基本块、分析数据流，得到可执行代码的 DFG 图文件,文件基于 VCG[6]格式表述指令流的数据依赖关系；然后在 DFG 图上运行候选指令扩展算法[7]，生成包含原指令个数大于 1 的非平凡候选指令实例集合。我们使用 GPROF 剖析测试程序，然后使用脚本得出测试程序中各个基本块的执行频率，以基本块的指令频率作为候选指令实例的执行频率。选择算法所需的各指令的占用面积，软硬件执行延时等从硬件库中抽取得到。

3.3 实验数据及分析

我们用 Mediabench[5]中的 cjpeg 作为测试程序分析比较算法的结果和时间性能。cjpeg 包含有 5 个热点函数，执行指令集扩展算法[7]后，得到 70 条非平凡候选指令，506 条非平凡候选指令实例，基本块执行频率从 327 到 198452 不等。

图 1，图 2 和图 3 是几种选择算法运行结果的比较。图中横坐标为选取候选指令的个数约束，纵坐标为选取结果的累计性能提升值。

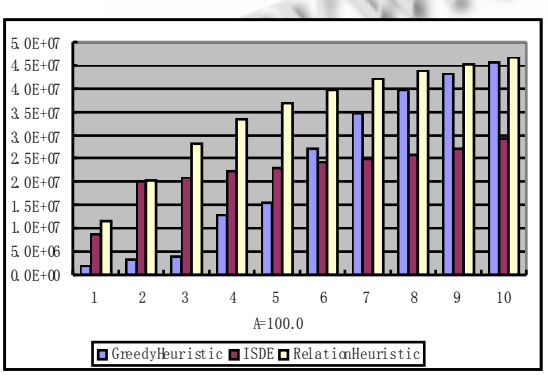

图 1 选取结果对比(A=100)

图 1 为面积约束 A=100，ISDE 算法取的是随机运行 20 次的平均值。可以看到在宽松的面积约束下 RelevanceHeur 算法的性能优于 ISDE 算法和 GreedyHeur 启发式算法。图 1 反映了在指令数目约束较强时的实验结果。在选取候选指令个数约束 M<5 时，ISDE 算法能够选择出的结果加速比远远大于 GreedyHeur 选取的结果，但是当 M>7 时其选取结果的加速比劣于 GreedyHeur 算法。而 RelevanceHeur 算法对于选取候选指令个数约束的强弱并不敏感。

图 2 和图 3 分别为在强面积约束和一般面积约束下三种算法的表现。从中也可以看出这两种情况下，RelevanceHeur 算法选取的结果加速比优于其他两种算法。

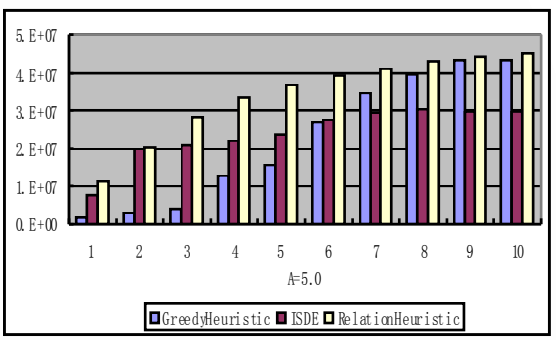

图 2 选取结果对比(A=5)

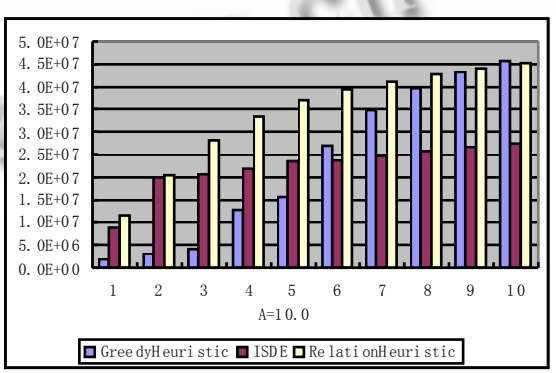

图 3 选取结果对比(A=10)

表明 RelevanceHeur 算法在运行的效率上大大优于 GreedyHeur 和 ISDE 算法。并且从表中可以很明显的看出，RelevanceHeur 算法相对于 GreedyHeur 和 ISDE 算法,其时间复杂度随选取指令的个数变化不明显。

表 1　算法时间效率比较

① P以 1e+07 周期为单位,Time 以秒为单位

M	GreedyHeur		DiffEvolution		RelevanceHeur	
	P	Time	P_{mid}	Time	P	Time
1	0.20	0.0129	0.89	0.1220	1.15	0.00306
2	0.31	0.0177	1.99	0.1675	2.03	0.00306
3	0.40	0.0219	2.08	0.1642	2.81	0.00312
4	1.28	0.0453	2.20	0.1668	3.33	0.00326
5	1.55	0.0714	2.35	0.1985	3.69	0.00329
6	2.70	0.1036	2.38	0.1637	3.93	0.00318
7	3.48	0.1246	2.47	0.1904	4.11	0.00332
8	3.96	0.1298	2.58	0.1983	4.28	0.00341
9	4.32	0.1485	2.67	0.2025	4.40	0.00338
10	4.56	0.1570	2.74	0.2211	4.51	0.00344

4　总结

自定义指令的选择算法在自定义指令算法研究中占有重要的位置,选择算法的好坏直接影响着 ASIP 指令扩展的效率和质量。已有的这一研究领域的算法在性能和效率上都有一定的缺陷。本文提出了一种包含指令与指令实例关系信息并且考虑已选和待选集合关联性的 RelevanceHeur 算法。实验数据对比表明,RelevanceHeur 算法能快速地选择出比原有启发式算法更优的候选指令集合,并且时间性能大大优于已有的算法,能够很好的应用于支持自定义指令扩展的 ASIP 应用中。

参考文献

1 Altera. Nios embedded processor system development. http://www.altera.com/products/ip/processors/nios2/ni2-index.html

2 Tensilica. http://www.tensilica.com/products/xtensa

3 Pan Y, Mitra T. Characterizing embedded applications for instruction-set extensible processors. Proc. of the 2004 International Conference on Compilers.ACM, 2004.67－78.

4 纪金松,周学海,张敏.基于差分进化和贪心策略的自定义指令选择算法研究.电子学报, 2009,37(2):372－376.

5 MediaBench.http://euler.sluedu/~fritts/mediabench/mbl

6 Sander,G.VCG Visualization of Compiler Graphs. Technical Report, Report 1996.

7 Pan Y, Tulika M. Scalable custom instructions identification for instruction-set extensible processors. Proc. of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems. ACM ,2004.