基于差分进化和贪心策略的自定义指令选择算法研究

周学海,纪金松,张 敏
(中国科学技术大学计算机系,安徽合肥 230027)

摘 要: 本文针对常见启发式算法中忽略指令与指令实例区别的问题,改进了一个已有启发式算法 GreedyHeur:根据指令实例的启发式函数值得出相应指令的权值,并根据指令的优先级关系以贪心策略进行指令实例选择.针对启发式算法无法找到最优解的问题,本文引入基于群体搜索的差分进化算法,并结合贪心策略,提出了 ISDE(Instruction Selection Based on Differential Evolution)算法. ISDE 算法通过简单的编码和高效的适应度评价机制,快速地迭代搜索最优指令组合.实验结果表明,GreedyHeur 和 ISDE 算法能快速有效地找到比已有启发式算法更优的候选指令组合.

关键词: 差分进化算法; 贪心策略; 指令集扩展; 指令选择
中图分类号: TP302 **文献标识码**: A **文章编号**: 0372-2112 (2009) 02-0372-05

Study on Differential Evolution and Greedy Strategy Based Custom Instruction Selection Algorithms

ZHOU Xue-hai, JI Jin-song, ZHANG Min
(Department of Computer Science, University of Science and Technology of China, Hefei, Anhui 230027, China)

Abstract: As heuristic algorithms usually omit the difference between instruction and instruction instance, we improved one existing heuristic algorithm to GreedyHeur algorithm. It calculates custom instructions' weights from their instruction instances, then select custom instruction instances with greedy strategy according to their instructions' weights. To find better custom instruction than heuristic algorithms, we introduced an algorithm (ISDE) integrating greedy strategy with differential evolution algorithm. Simple encoding and efficient fitness evaluation help ISDE find the best combination of custom instructions quickly. Experiments show that our algorithms can find better custom instruction candidates more quickly and efficiently than heuristic algorithm.

Key words: differential evolution algorithm; greedy strategy; instruction set extension; instruction selection

1 引言

随着消费类数字产品市场的日趋繁荣,面向特定应用的处理器(Application Specific Instruction-set Processor, ASIP)也越来越受到青睐.但是随着多媒体的引入,用户对 ASIP 的性能需求越来越高,通用指令集 ASIP 已不能满足多媒体处理需求.同时,设计一个专用的 ASIC(Application Specific Integrated Circuit)又需要太多的人力物力以及很长的上市周期.可扩展指令集的 ASIP 很好的解决了这一个问题.

可扩展指令集 ASIP 的设计一般是基于通用的精简指令集处理器核,由设计者根据特定应用的具体特征对其核心指令集进行扩展.设计者通过使用测试程序对特定应用进行剖析,抽取出其中频繁执行的代码段,并使用硬件直接实现成扩展指令,从而实现性能上的提升.具体的流程是:设计者将测试集在仿真器上或实际硬件上得到的剖析信息反馈到指令集扩展算法上,由指令集扩展算法从数据流图(Data Flow Graph, DFG)上找出适合的模式,然后再使用选择算法,从这些候选模式中选择出扩展指令.整个过程可以根据扩展指令的需求和实际性能进行多次迭代.这种设计方式,只需要在原有芯片的基础上进行少量的修改就可以设计出满足性能需求的面向特定应用的新芯片,大大缩短了芯片开发周期和上市时间.目前已经有多种支持可扩展指令的 ASIP,如 Altera 的 NIOS II和 Tensilica 的 Xtensa 等.

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.国内外很多学者在这方面做了不少研究[1~6]. Arnold[1]等采用动态规划来选择没有约束的候选指令,但是没有约束的指令在实际应用中比较少见. Cheung[2]等提出先对面积约束使用迭代的贪心算法来选择候选指令集,然后再通过仿真计算出相应的加速比,最后选择出最大值. Pan[3]等则定义了性价比,执行时间,加速比等启发函数,使用启发式的算法来选出候选集合. Lee[4]等也提出了一个启发式算法,并和

收稿日期:2008-01-21;修回日期:2008-05-19
基金项目:安徽省自然科学基金(No. 070412030);高等学校博士学科点专项科研基金(No. 20050358040)

ILP 算法进行了比较. Clark[5]等则将该问题对应到 0 - 1 背包问题上,使用了结合启发式的贪心选择算法进行选择. 这些算法在他们各自的指令集扩展流程中有效的实现了指令选择功能,但是它们仍存在不少问题. 如动态规划和 ILP 算法在候选指令数较多或约束较多时计算量大[3],效率低,一般只适合小规模指令集合(～15 节点)的计算[6];贪心算法一般无法保证选择到最优指令,启发式算法往往局限于启发函数,也无法保证最优解. 差分进化算法[7]是基于群体搜索的启发式算法,通过遗传迭代能实现全局最优解的搜索. 针对已有指令选择算法存在的问题,本文分析改进了文[3]的启发式算法,并提出结合启发函数和贪心策略的差分进化算法,快速有效的解决了最优指令的选择问题.

2　指令选择问题及算法描述

2.1　问题描述

令自定义候选指令为 $C_1, \cdots, C_n$,它们在程序中出现的 n_i 个实例分别为 $c_{i.1}, \cdots, c_{i,n_i}$,每个实例的执行频率记为 $f_{i,j}$. A_i 是自定义指令 C_i 的面积需求, P_i 是选中 C_i 所获得的性能提高(将 C_i 使用硬件实现,相比用软件实现的性能提升,以时钟数的形式表示),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);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抽象为多约束最优化问题:

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

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

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

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

2.2　贪心启发式算法 GreedyHeur

常见的启发式算法一般都在启发式函数上进行尝试和改进,却往往忽视了算法本身的问题. 文[3]中使用三种不同的启发式函数,但是这些函数都仅仅对指令实例起作用. 而且算法在处理覆盖集约束时,强化了约束,不允许选中任何包含相同指令的指令实例. 但问题本身在保证被选中的指令实例不互相覆盖的情况下是允许包含相同指令的. 针对这些问题,我们尝试在启发式函数中包含指令与指令实例的关系信息:先根据指令实例的启发式函数值得出相应指令的权值,然后再根据指令的优先级关系,以贪心策略进行指令实例选择;处理覆盖集约束时,仅删除互斥的指令实例. 算法(GreedyHeur)的描述如算法 1 所示.

算法 1　GreedyHeur 算法

步骤 1　对自定义指令实例集合 c 中的所有指令实例,计算其启发式函数权值;并计算出相应指令集合 C 中各指令的权值.

步骤 2　如候选指令实例集合 X 为空或不满足指令数目约束,结束. 否则从 X 中选择并删除具有最高指令权值的指令实例 c_{i,j_0}.

步骤 3　如指令实例 c_{i,j_0} 所对应指令 C_i 不满足资源约束,则删除 X 中所有 C_i 的实例,否则选中该指令:$S_i = 1, A = A - A_i$ iff $A_i \leq A$,并依次处理该指令的实例;转步骤 2. 其中对所有 C_i 的实例 $c_{i,j}$ 处理步骤为:

步骤 3.1　选中指令实例 $c_{i,j}$,累计总体性能提升值 $s_{i,j} = 1, P = P + P_i \times f_{i,j}$

步骤 3.2　处理覆盖集约束,删除 X 中相应指令实例,使得

$$\forall i',j', s_{i,j} + s_{i',j'} \leq 1 \quad \text{iff} \quad c_{i,j} \cap c_{i',j'} \neq \phi$$

2.3　差分进化指令选择算法 ISDE

由于指令选择中存在覆盖集约束,且每一个指令实例的互斥集合各不相同,大小也各异. 如果仅仅只是贪心的选择最高优先级的指令实例,可能会导致删除与其互斥的大量相对较低优先级的指令实例,从而使最终选择的指令组合总体性能提升反而不高. 针对这个问题,我们引入基于群体搜索的差分进化算法,在整个搜索空间里进行迭代的启发式搜索,使选择出的指令的总体性能尽可能达到最优.

2.3.1　编码机制

采用二进制 N 维向量 $s = \{s_{0.1}, \cdots, s_{0.k_0}, \cdots, s_{n.1}, \cdots, s_{n,k_n}\}$ 对指令实例进行编码,最终的候选指令可由相应的实例编码译码获得. 译码式为 $S_i = s_{i.0} | s_{i.1} | \cdots | s_{i.n_i}$.

2.3.2　适应度评价

算法 2　个体适应度评价算法

```
ind.y=0; currentindex=-1; numInsts=0;// 初始化各变量
tmpind=ind;
// 拷贝个体用于修复,不改变原有个体以保证群体多样性
for i = 0 to nbin do
    if tmpind.x[i]==0 then
        continue;// 不选中指令实例
    end
    insdindex=tmpind.x[i]'s instruction index;
    // 译码取得染色体对应指令的编号
    if currentindex!=insindex then
        if numInsts ≥ M then
            tmpind.x[i..nbin]=0; break;// 指令数目检查
        end
        if Candidates[i].R > R then
            tmpind.x[i]=0; continue;// 资源约束检查
        end
        R-=Candidates[i].R;// 选中指令
        currentindex=insindex; numInsts++;
    end
    ind.y+=Candidates[i].P;
    // 选中该指令实例,累计性能提升值
    clearmutexbits(tmpind,i);
    // 并修复相应位以符合覆盖集约束
end
if y_best<ind.y then
    y_best=ind.y;bestind=tmpind;
    // 设成群体最优值和最优个体
end
```

使用指令实例编码时,对应的适应度评价函数如式(1)所示. 由于存在覆盖集约束,经过交叉、变异的新个体,并不一定包含在可行的解空间中. 因此,需要根据覆盖集约束对新个体进行修复. 简单的随机修复算法会在整个搜索空间里面进行搜索,收敛较慢. 我们采取贪心的启发式修复来加快收敛速度:先使用启发函数对所有指令进行评估,优先修复高优先级指令所对应的指令实例的染色体. 为了减小修复时的复杂度,我们在编码之前先对指令实例进行启发式函数的评估,然后根据指令实例所对应的指令的权值从高到低顺序编码. 在适应度评估时则按编码顺序对染色体进行修复. 个体评估算法的描述如算法 2 所示.

2.3.3 算法描述

设编码长度为 N,群体规模为 M,差分进化指令选择算法 ISDE 的形式化描述如算法 3 所示.

算法 3 ISDE 算法

步骤 1 初始化并评估群体 P_0, $t=0$.

步骤 2 离散差分产生下一代 Q_t,对于第 i 个个体 $Q_{t,i}$,产生方式具体如下:

$$Q_{t,i}(j)=\begin{cases} P_{t,k}(j), & u\leq 0.5,\ k=\mathrm{rnd}(1,M) \\ P_{i,j}(j), & u>0.5 \end{cases}$$

其中 u 是区间 $[0,1]$ 上的均匀随机数, $P_{t,k}(j)$ 以概率 $5/N$ 进行变异. $1\leq j\leq N$

步骤 3 评估群体 Q_t,计算个体的适应度值.

步骤 4 选择,取 P_t 和 Q_t 中对应位置适应度值大的个体进入下一代 P_{t+1}.

步骤 5 判断终止条件是否满足,若满足则结束. 否则, $t=t+1$,转步骤 2.

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. $T_{GreedyHeur}$ 主要包括启发式函数计算时间 $T_{H\text{-}cal}$、选择最高权值指令时间 $T_{H\text{-}sel}$、选中指令的操作时间 $T_{H\text{-}op}$. 其中 $T_{H\text{-}cal}$ 和 $T_{H\text{-}sel}$ 复杂度均为 $O(n)$. 选中指令后主要包括参数计算和覆盖集约束处理. 参数计算可在 $O(1)$ 时间内完成,约束检查则可通过预处理先计算出所有指令实例的互斥集合,然后根据集合进行判定,设互斥集合最大为 $|M|$,则约束检查的复杂度为 $O(|M|)$. 一般情况下 $|M|$ 是常数,且 $M<m$. 故 $T_{GreedyHeur}$ 的时间复杂度是 $O(m\times n)$. T_{ISDE} 主要包括初始编码时间 $T_{D\text{-}init}$ 和种群迭代进化时间 $T_{D\text{-}iter}$. $T_{D\text{-}init}$ 中主要进行指令实例和指令的权值计算,复杂度为 $O(n)$. 设算法进化 I_D 代,种群规模为 N,种群中个体完成遗传操作的时间为 $T_{D\text{-}OP}$,适应度计算时间 $T_{D\text{-}fit}$,则种群迭代进化时间是 $T_{D\text{-}iter}\approx I_D\times N\times(T_{D\text{-}op}+T_{D\text{-}fit})$. 其中 $T_{D\text{-}op}$ 的复杂度为 $O(n)$,个体适应度评价复杂度为 $O(n\times|M|)$. 故 $T_{ISDE}\approx O(I_D\times N\times m\times n)$. 其中 N 为常数,而 I_D 和收敛快慢有关,在快速收敛的小规模种群上 I_D 和 N 相对 n 比较小, T_{ISDE} 时间复杂度约为 $O(m\times n)$.

3.2 实验及结果分析

3.2.1 实验环境

为了验证和比较 3 个算法的性能,我们在 GNU 工具链的基础上实现了如图 1 所示的实验框架. 实验选择 Altera 的 NIOSII 指令集为基本指令集,MiBench[8] 及 NetBench[9] 作为测试程序.

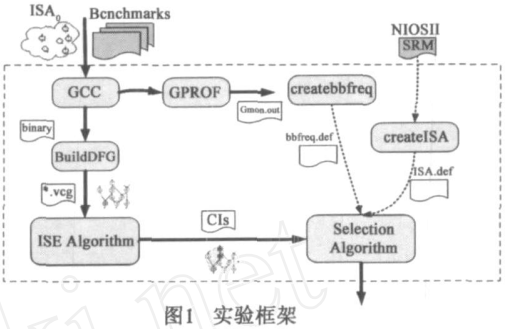

图 1 实验框架

我们先从程序中选出最耗时的几个热点函数,使它们的运行时间和占总运行时间 90 %以上;然后执行指令扩展算法 MIMOGen[3,10]. 算法得出的候选指令实例的数目累计均在 600 左右,其规模在基于二进制编码的差分进化算法的应用范围之内.

3.2.2 实验数据及分析

我们选择 cjpeg 的数据来分析和说明算法的性能. cjpeg 程序含有 5 个热点函数,指令集扩展生成 506 条非平凡候选指令实例,基本块执行频率从 327 到 198452 不等. 所有算法中的启发式函数均采用性能提升/面积(代价)比;差分进化算法中初始种群数为 20,重组概率 0.5,变异概率 $5/M$. 实验中差分进化算法都在 150 代内收敛,所以我们将迭代代数设置为 150.

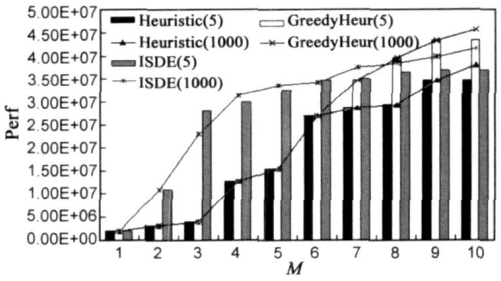

图 2 算法选择结果比较

图 2 是各算法选择的结果比较,横坐标表示指令数目约束,纵坐标表示候选指令累计性能提升值. 其中 ISDE 算法取的是随机运行 20 次的中间值,图例中的括号中的参数是面积约束 A. 从图中可知,ISDE 算法在指令数目约束较强时($M<7$)能选择出比启发式算法都好的指令组合. 而 GreedyHeur 算法在指令约束较强时和原启

发式算法差不多,但在指令数目约束较弱时($M > 7$)则能选择出比其他算法都好的指令组合. 图 3 则是 ISDE 算法在不同的参数(M,A)下运行 20 次的结果分布图. 从图中可知,在面积约束较强($M = 10$,$A = 5$)或指令数目约束较强($M = 1$)时,算法的选择结果波动较小;而在面积约束较弱时($M = 5$,$A = 1000$),算法的选择结果波动较大. 表 1 前半部分给出 $A = 1000$ 时性能提升值的最小值、中值和最大值. 这些数据也表明算法选择结果在 M 较小时波动较大,但总体波动范围不大.

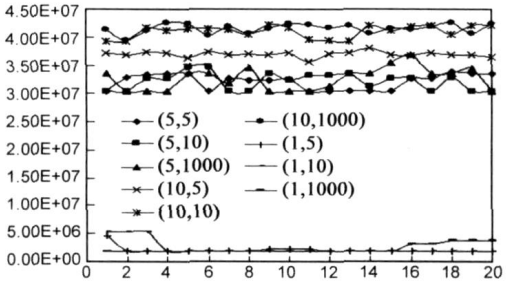

图3　ISDE结果分布图

表 1　ISDE 算法结果和时间分布

M	Perf. (1 + e07 cycles)			Elapse Time(s)		
	P_{min}	P_{mid}	P_{max}	T_{min}	T_{max}	T_{ave}
1	0.20	0.20	0.39	0.294	0.305	0.297
2	0.89	1.08	2.00	0.292	0.347	0.325
3	1.35	2.30	2.81	0.336	0.512	0.443
4	2.50	3.14	3.31	0.495	0.679	0.583
5	3.04	3.35	3.68	0.566	0.804	0.648
6	3.16	3.43	3.80	0.622	0.806	0.705
7	3.43	3.75	4.00	0.682	0.874	0.792
8	3.52	3.83	4.06	0.727	0.930	0.819
9	3.87	3.98	4.12	0.801	0.953	0.891
10	3.95	4.16	4.26	0.896	1.012	0.955

从运行时间来看,GreedyHeur 算法最快,ISDE 算法最慢. 表 1 后半部分给出了 ISDE 算法运行 20 次的时间详细数据. 表 2 则给出了三种算法的结果和运行时间的详细数据($A = 1000$). 从这些表中可得,在 M 较小时,GreedyHeur 能以比原有启发式算法快近 10 倍的速度求得相同的结果. 而 ISDE 算法虽然比其他算法稍微慢一些,但其最坏情况仍在指令扩展应用的可承受范围之内($\leq$100s),而它在指令约束较强时选出的指令组合的性能提升值却能远远超过其他算法.

实验数据表明 GreedyHeur 算法和 ISDE 算法均优于原有启发式算法. GreedyHeur 算法适合应用于对算法运行时间要求比较强或者指令数目约束较弱的情况;而 ISDE 算法则适合应用于指令数目约束较强的场景. 在 ASIP 的指令集扩展应用中,需要扩展的指令数目往往不会非常多(M 通常在 1 到 10 之间[11]),运行算法运行时间要求也不是非常严格,但是对算法所能选出的指令组合的性能提升值却非常重要. 所以,ISDE 算法恰能很好的适应于 ASIP 的指令集扩展应用中.

4　总结与展望

自定义指令的选择算法从候选集合里面选出性能提升较高的候选指令组合,它直接影响 ASIP 指令扩展的效率和质量. 虽然目前已有多种启发式算法和 ILP 算法等,但这些算法在性能和效率上各有缺陷. 本文提出了一种包含指令与指令实例关系信息的贪心启发式算法 GreedyHeur 和一种结合贪心策略和差分进化算法的 ISDE 算法. 实验数据表明,GreedyHeur 算法能快速的选择比原有启发式算法更优的候选指令集合,而 ISDE 算法在指令数目约束较强时能在可承受的运行时间内选出性能提升值远远超过其他算法的候选指令组合,特别适用于支持自定义指令扩展的 ASIP 应用中. 后续工作将进行更深入的实验,调整、选择或设计针对该问题的专用遗传算子,使该算法能更快速高效的选择到更好的候选指令组合.

表 2　算法效率比较

M	Heuristic[3]		GreedyHeur		ISDE	
	P	Time	P	Time	P_{mid}	Time
1	0.20	0.2728	0.20	0.0237	0.20	0.2974
2	0.32	0.2704	0.32	0.0354	1.08	0.3254
3	0.40	0.2712	0.40	0.0445	2.30	0.4431
4	1.28	0.2674	1.28	0.0908	3.14	0.5828
5	1.55	0.2709	1.55	0.1004	3.35	0.6475
6	2.70	0.2766	2.70	0.1535	3.43	0.7048
7	2.87	0.2684	3.47	0.1818	3.75	0.7922
8	2.94	0.2741	3.96	0.1936	3.83	0.8188
9	3.47	0.2713	4.32	0.2202	3.98	0.8912
10	3.80	0.2704	4.56	0.2321	4.16	0.9551

注:P 以 1e + 07 为单位,Time 以秒为单位,ISDE 取 20 次运行的均值

参考文献:

[1] Marnix A, Henk C. Designing domain-specific processors[A]. CODES '01: Proceedings of the Ninth International Symposium on Hardware/Software Codesign[C]. Copenhagen, Denmark: ACM, 2001. 61 - 66.

[2] Newton C, et al. Rapid Configuration and instruction selection for an ASIP: A casestudy[A]. Proceedings of the Conference on Design, Automation and Test in Europe[C]. Washington DC, USA: IEEE Computer Society, 2003. 10802.

[3] Pan Y, Tulika M. Scalable custom instructions identification for instruction-set extensible processors[A]. Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems[C]. Washington DC, USA:

ACM ,2004. 69 - 78.
[4] Lee J ,et al. Efficient instruction encoding for automatic instruction set design of configurable ASIPs [A]. Proceedings of the 2002 IEEE/ ACM International Conference on Computer-aided design[C]. San Jose ,California ,USA :ACM ,2002. 649 - 654.
[5] Clark N , et al. Processor acceleration through automated instruction set customization[A]. Proceedings of the 36th Annual IEEE/ ACM International Symposium on Microarchitecture [C]. Washington DC , USA : IEEE Computer Society , 2003. 129.
[6] Atasu K,et al. Automatic application-specific instruction-set extensions under microarchitectural constraints [J]. International Journal of Parallel Programming ,2003 ,31(6) :411 - 428.
[7] Storm R , Price K. Differential evolution-A simple and efficient heuristic for global optimization over continuous spaces [J]. Journal of Global Optimization ,1997 ,11(4) :341 - 359.
[8] Guthaus M ,et al. MiBench :A free ,commercially representative embedded benchmark suite [A]. Proceedings of the Workload Characterization [C]. Washington DC , USA : IEEE Computer Society ,2001. 3 - 14.
[9] Memik G , et al. NetBench : A benchmarking suite for network processors [A]. Proceedings of the 2001 IEEE/ ACM International Conference on Computer-aided Design [C]. San Jose , California ,USA. IEEE Press ,2001. 39 - 42.
[10] Pan Y, Tulika M. Efficient custom instruction identification with exact enumeration[R]. Singapore :National University of Singapore ,2007.
[11] Micheael F S , et al. Characterization of Repeating Dynamic Code Fragments [R]. Urbana , USA : University of Illinois , Center for Reliable and High-Performance Computing ,2002.

作者简介:

周学海 男,1966 年生于安徽淮南,中国科学技术大学教授,博士生导师. 主要研究方向:嵌入式系统开发、计算机系统结构、可重构计算.
E-mail :xhzhou @ustc. edu. cn

纪金松 男,1981 年生于福建漳州,中国科学技术大学计算机科学技术系博士研究生. 主要研究方向:指令集扩展、可重构计算.
E-mail :jsji @mail. ustc. edu. cn

张 敏 男,1981 年生于安徽肥东,中国科学技术大学计算机系博士研究生. 主要研究方向:遗传算法,进化计算. E-mail :zhangmin @ mail. ustc. edu. cn