面向嵌入式应用的指令集自动扩展

吕雅帅, 沈 立, 黄立波, 王志英
(国防科学技术大学计算机学院, 湖南长沙 410073)

摘 要: 面向特定应用扩展指令集, 并通过定制的硬件实现这些扩展指令, 能够大幅度提高嵌入式处理器的性能. 本文提出了一种全自动的面向特定应用的指令集扩展流程, 该流程能够较精确地估算扩展指令的性能加速比和硬件开销, 并高效完成指令模板匹配. 实验结果表明, 在给定的硬件开销限制下, 该方法产生的扩展指令能够显著提升嵌入式应用的性能.

关键词: 嵌入式应用; 扩展指令; 自动生成
中图分类号: TP314 文献标识码: A 文章编号: 0372-2112 (2008) 05-0985-04

Automatic Instruction Set Extension for Embedded Applications

LÜ Ya-shuai, SHEN Li, HUANG Li-Bo, WANG Zhi-ying
(*Department of Computer Science, National University of Defense Technology, Changsha, Hunan 410073, China*)

Abstract: Application-specific instructions play an important role in reducing code size and improving performance for embedded processors. Automating the design process of these instructions can shorten the time-to-market of the product. This paper proposes a fully automatic approach on instruction set extension for embedded applications and a new algorithm for instruction pattern mapping. Experimental results show great performance improvement on various embedded applications under given hardware constraints.

Key words: embedded application; instruction set extension; automatic generation

1 引言

在当前的嵌入式处理器设计中, ASIP[1] (Application Specific Instruction-set Processor) 结合了通用处理器和 ASIC 的优点, 受到了越来越多的研究设计人员及芯片厂商的青睐, 已成为当前嵌入式处理器设计的重要方法之一. 面向特定应用扩展指令集是 ASIP 设计的核心环节. 在以前的 ASIP 设计工作中, 设计人员往往根据自己的经验加入一些特殊指令, 然而, 仅凭经验设计人员无法准确回答这些扩展指令究竟能够带来多大的性能加速比等问题, 因此必须分析不同应用的具体特征. 而且, 随着嵌入式芯片处理能力的不断提高, 嵌入式应用的复杂度也越来越高, 若总是由芯片设计人员通过手工方式分析应用特征来决定应扩展哪些指令, 不但会大大分散设计人员设计芯片的精力, 还会大大延长芯片的设计周期. 所以, 设计并实现一套自动分析程序, 使之能够根据应用特征扩展原有的指令集, 对于 ASIP 的设计来说是不可或缺的.

本文在充分研究前人工作的基础上, 针对现有研究的不足之处, 提出并实现了一个全自动的指令集扩展系统以及一个新的指令模板匹配算法. 实验结果表明, 该方法效果良好.

2 相关工作

面向嵌入式应用扩展指令的研究在国外开展的比较早, 但早期研究[1, 2] 主要关注如何通过定制硬件高效地实现扩展指令. 随着嵌入式技术的飞速发展, 很多国外研究人员便开始思考如何自动地面向特定应用定制指令[3~13].

实现指令集自动扩展必须解决两个主要问题:

(1) 分析目标应用特征, 生成候选指令模板集.

目前比较流行的方法是在数据流图(DFG) 上搜索满足特定约束的子图, 所产生的子图集合就是候选指令模板集[5~13]. 虽然这种方法在最坏情况下的时间复杂度是 $O(2^n)$, 但通过采用启发式搜索以及剪枝策略, 在实际应用中可以得到相当快的搜索速度.

(2) 从候选指令集中选择指令作为扩展指令, 并重新编译应用程序.

实际上它包含了两个不可分割的方面: 如何从候选指令集中选择合适的扩展指令, 以及如何用选出的扩展指令替换原有 *DFG* 的相应部分. 而现有的研究存在以下几个主要问题:

收稿日期: 2006-11-13; 修回日期: 2007-12-26
基金项目: 国家 973 重点基础研究发展计划(No. 2007CB310901)

首先, 很多研究工作把候选指令选择与对 DFG 的匹配替换分开进行处理. 例如, Lee[7] 和 Yu[11] 把对候选指令选择归结为整数线性规划问题, 而 Cong[11] 则把该问题描述为 0- 1 背包问题. 这些工作都是在确定了扩展指令后再进行 DFG 匹配替换, 并且这两种方法都认为候选者之间不存在关联, 即选择一个候选者之后不会引起其他候选者的收益变化. 但那些在同一个 DFG 上存在相同节点的候选指令模板之间显然是存在关联的, 选择一个指令模板后必将导致其他候选模板无法选择.

其次, 对候选指令模板进行选择意味着首先要对其进行评估, 而在先前的研究中没有给出一种具体且准确的自动评估方法. 而本文给出了一个候选指令模板自动评估方法, 完善了先前研究在这方面的不足之处.

再次, 候选指令模板的匹配问题也没有得到较好的解决. 为了使目标应用的性能得到最大提升, Cong[10]、Brisk[6] 和 Clark[9,12] 分别探讨了 DFG 的最优匹配问题. 他们工作的共同之处在生成候选指令之后, 用图形匹配库把指令模板与各个 DFG 进行同构匹配以求得最优解. 这种方法存在的主要问题在于: (1) 图的同构匹配是 NP 完全问题, 匹配耗费的时间随着问题的规模呈指数级增长趋势; (2) 这种方式虽然能够求出最优解, 但根本没有考虑硬件实现开销. 当硬件设计者希望将指令扩展的硬件开销限制在一个范围内时, 按照这种方式甚至无法求得一个解.

3 指令集扩展的自动实现

3.1 候选指令模板的性能估算

先前的很多研究工作只是提出了应以延迟、面积作为参数从候选指令模板集中挑选扩展指令, 却没有给出如何自动地计算这些参数的方法. 本文给出了一个比较精确的自动估算指令模板性能的方法.

(1) 评估参数 本文通过三个参数, 周期(Cycle)、面积(Area) 和性能收益(Gain) 评估候选指令模板的优劣. Cycle 是该指令模板作为一条扩展指令实现时所需的时钟周期数, Area 是该指令模板在硬件实现上所需的面积, Gain 是该指令模板可以节省的处理器时钟周期数.

(2) 硬件信息库 硬件信息库中存放了在特定工艺下, 基本指令以及部分硬件上已实现的扩展指令的延迟与面积信息. 这些信息是候选指令模板性能估算的基础. 信息库的构成如表 1 所示:

表 1 硬件信息库的构成

基本指令			扩展指令		
操作	面积(mm^2)	延迟(ns)	操作	面积(mm^2)	延迟(ns)
ADD	0.01	1.3	MUL- ADD	0.15	3.5
MUL	0.12	2.9	ADD- ADD	0.012	2.0
≫	0.007	0.8	≪- ADD	0.017	2.1

(3) 单个模板的计算方法 对于由基本指令 i_1、i_2、…、i_n 构成的指令模板 p, 前述三个参数的基本计算公式如下:

$$Cycle(p) = Ceil((G_{delay}(i_{c1}) + G_{delay}(i_{c2}) + \cdots + G_{delay}(i_{cn})) * Clock_f) \quad (1)$$

$$Area(p) = Area(i_1) + Area(i_2) + \cdots + Area(i_n) \quad (2)$$

$$Gain(p) = Frequency(p) * (Cycle(i_{c1}) + Cycle(i_{c2}) + \cdots + Cycle(i_{cn}) - Cycle(p)) \quad (3)$$

其中, $Ceil(n)$ 表示不小于 n 的最小整数, $G_{delay}(i_c)$ 是基本指令 i_c 硬件实现的门延迟, i_c 表示位于指令模板关键路径(Critical Path) 上的指令, $Clock_f$ 是处理器的时钟频率, $Area(i)$ 表示指令 i 的面积, $Frequence(p)$ 是指令模板 p 的执行频率. 公式(1)、(2)、(3) 只是基本的计算公式, 系统还要依据硬件信息库中扩展指令的硬件信息对该模板的评估参数进行更精确的计算.

(4) 功能单元面积的计算 最终所有的扩展指令是要放到功能单元(Function Unit) 中实现的, 当多个扩展指令放到同一个功能单元中实现时, 不同的扩展指令之间可以部分共享硬件, 也就是说功能单元的硬件实现面积要小于多个扩展指令的面积之和.

以图 1 为例, 这三个扩展指令第二步 ADD 操作的硬件实现是可以共享的, 因而该功能单元面积应该是 $2*Area(ADD)+Area(MUL)+Area(\ll)$, 而不是这三个扩展指令的面积之和. 本文实现的系统在计算功能单元面积时, 在不影响功能单元关键路径的前提下, 合并多个扩展指令在同一阶段的相同操作, 以这种方式能更加精确地计算功能单元的面积.

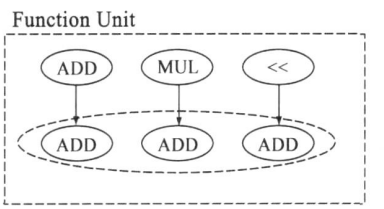

图 1 功能单元的面积计算

3.2 候选指令的选择匹配算法

候选指令的选择与匹配是密不可分的两个过程, 这一问题可以描述为: 从给定的候选指令模板集 P_c 中选取一组指令模板 p_1、p_2、…、p_n, 匹配并替换这些 DFG 中与给出模板同构的子图, 使得由这些指令模板构成的功能单元 FU 在满足 $Area(FU) \leqslant AREA_{max}$ 的条件下, 性能加速比最大.

不难看出, 这是一个 NP 难的问题, 要想求得最优解只能采用时间复杂度为指数级的算法. 本文提出的算法基于以下三点考虑: (1) 各个指令模板的性能收益并不是平均分布的, 性能收益大的模板往往只占其中一小部分, 在那些对程序性能影响不大的模板之间搜索是没有意义的; (2) 模板之间存在很强的关联关系, 选择一个模板匹配后, 往往会影响另外模板的性能加速比, 所以模板选择应该是一个迭代的过程比较合适, 而且每选出

一个模板后, 需要重新计算其它候选模板的性能收益; (3) 一个好的可以在实际中应用的算法应该是能够在时间复杂度与结果优劣之间进行折衷的算法.

以下是该算法框架, 其中 P_c 是候选指令模板集合, P_{sel} 是当前选择出的指令模板集合, P_{best} 记录了性能提升最好的一组指令模板, $AREA_{max}$ 是面积约束.

```
Algorithm PatternMatching( PatternSet Pc, PatternSet Psel, double AREAmax)
  begin
    AREAfu= ComputeFUArea(Psel);
    if AREAfu> AREAmax then return;
    if(Gain(Psel)> Gain(Pbest)) Pbest= Psel;//记录性能最好的一组解
    Pattern_Gain_ReCalculate(Pc);//重新计算各个模板的性能收益
    Pattern_Sort(Pc);//按照新的收益值对候选模板排序
    repeat
      Pc= Pc- {p1};   //每次从候选集中选取一个收益最好的模板
      Psel= Psel+ {p1};//将该模板加入到当前选择出的模板集合中
      for each G(v, e) inprogram do
        SubGraphMatch(G(v, e), p1);                    ①
      PatternMatching(P, Psel, AREAmax);               ②
      for each G(v, e) inprogram do
        SubGraphMatchRestore(G(v, e), p1);             ③
      Psel= Psel- {p1};
    until exp(- K/Gain(p1))> random(1) or IsEmpty(P);  ④
  end;
```

该算法第一步计算由当前选择出的指令模板所构成的功能单元的面积是否超出限制; 第二步把当前解的性能收益与之前搜索到的解比较, 记录下最好的一个; 第三步重新计算候选模板集合中各个模板的性能加速比, 并按性能加速比由高至低排序; 第四步是一个循环, 每次从候选指令模板集合 P 中选取性能加速比最大的一个模板(即 p_1)加入 P_{sel}, 语句①用该模板在所有 DFG 上进行匹配替换, 语句②在此基础上进一步搜索新的模板, 语句③把用 p_1 匹配过的 DFG 恢复到原有状态.

该算法的优点在于一方面它避免了大量的对程序性能影响不大的搜索, 另一方面又避免了贪婪算法结果单一的缺点, 并且可以通过调整参数 K, 在搜索时间长短与结果好坏之间进行折衷.

4　性能测试与结果分析

我们以 TTA 结构[14] 做为 ASIP 的基准体系结构, 选择了 8 个典型的嵌入式应用作为基准程序进行测试. 基准程序包括数字信号处理领域中常见的两个典型算法快速傅立叶变换(fft)和离散余弦变换(idct), 嵌入式应用基准测试包 MiBench 中的 6 个程序(susan, djpeg, dijkstra, rawdaudio, sha, blowfish), 以及 H.264 和 AVS 两种视频编解码标准的解码程序. 硬件开销信息来自于 Design Ware 综合 IP 库. 基本 TTA 处理器核的配置为: 4 个整数运算单元, 2 个浮点运算单元, 2 个比较运算单元, 2 个存储操作功能单元, 64 个通用整数寄存器和 64 个浮点寄存器.

(1) 面积限制与性能提升　图 2 给出了不同面积限制下各个应用的性能提升曲线. 对于 fft、idct、djpeg、H.264 和 AVS 来说, 性能提升随着面积限制的增大呈稳步增长趋势, 面积限制超过 0.4 时基本达到性能提升的上限; 而 sha 和 blowfish 这两个加密领域的应用在面积限制为 0.2 时, 性能提升就达到了上限, 这是因为这两个应用中多为 XOR、AND 这样占用面积很小的位操作, 所以其性能提升曲线比较陡峭; susan、adpcm 和 dijkstra 属于分支密集的应用, 可扩展指令很少, 性能提升有限.

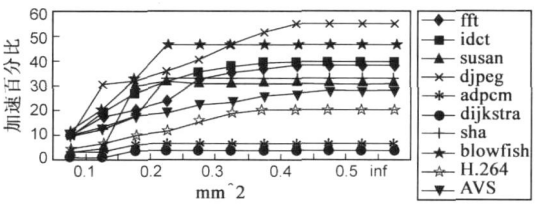

图 2　不同面积限制下的性能提升

(2) 模板匹配算法中参数 K 的影响　图 5 和表 3 给出了模板匹配算法中参数 K 分别取 0.01、0.001 和 0.0001 这三个值时, 性能与算法执行时间的统计. 其中当 K 取 0.01 时, repeat 循环只执行一次, 此时该算法等价于贪婪算法.

从图 3 和表 3 中可以看出对于分支密集型的程序而言, 由于候选指令模板很少, 所以 K 在取不同值时, 无论是性能提升还是算法执行时间都差不多. 从表 3 中还可以看出, 贪婪算法的速度是非常快的, 但其性能提升不如 K 取值为 0.001 和 0.0001 时好.

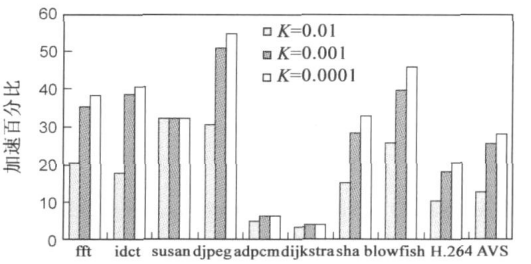

图 3　优化参数 K 取不同值时, 性能提升对比

表 2　参数 K 取不同值时, 匹配算法的执行时间(单位: 秒)

	fft	idct	susan	djpeg	adpcm	dijkstra	sha	blowfish	H.264
K= 0.01	0.13	0.09	0.07	0.21	0.11	0.10	0.26	0.22	0.46
K= 0.001	2.78	1.96	0.12	3.56	0.15	0.11	4.08	3.96	8.93
K= 0.0001	13.87	10.34	0.141	7.63	0.17	0.11	21.45	19.82	35.47

(3) 测试结果的结论　通过对各个应用最后测试结果的观察, 还可以得出以下几点结论:

(a) 指令扩展对于计算密集型应用性能提升明显, 而对于分支密集的应用效果不好.

从上面可以看出,除了 adpcm 和 dijkstra 之外,其余应用的性能提升基本都在30%以上,由于H.264和AVS程序比较大,我们仅测试了几帧视频,实际运行时性能提升应该可以达到 30% 左右.对于 adpcm 和 dijkstra 而言,分支操作占据了绝大部分运行时间,扩展指令对程序性能提升的效果十分有限.

(*b*)相同领域应用的扩展指令存在一定相似性.测试结果显示出,扩展指令相似性最明显的是H.264 decoder 和AVS decoder 这两个高清晰视频解码标准的实现程序. ADD→MIN→MAX→ADD、ADD→ADD、ADD→MIN 和MUL→ADD 都是对这两个程序性能提升影响最大的指令.值得一提的是,在与图像处理相关的程序 djpeg、H.264 和 AVS 中,扩展3操作数加三法指令后能够有效提高它们的性能,其作用超过了通常的嵌入式芯片指令集设计中凭经验加入的乘累加指令.

(*c*) 通过增加不同的寻址方式,可以提升程序性能.基本TTA核中只支持立即数寻址,即:地址→LOAD→结果,与(地址,内容)→STORE.但是测试时中发现,地址计算→LOAD 和(地址计算,内容)→STORE 这两个扩展指令对各个测试程序的性能均有不同程度提升,这两个扩展指令实际上是基址寻址.在 fft、jpeg、adpcm、sha 和H.264 这四个应用中,基址寻址方式可以带来较大的性能加速比.从本质上讲,这是由于频繁的数组访问造成的.因此,增强寻址方式不但方便程序的设计,更重要的是提升程序的性能.

5 结束语

本文重点讨论了面向嵌入式应用的指令集自动扩展问题.并针对先前研究工作的不足之处,提出并实现了一个指令集自动扩展系统.该系统对先前的研究进行了完善与改进,实现了整个指令集扩展过程的自动化,并提出了一个候选指令模板的估算方法和指令模板的选择匹配算法.

参考文献:

[1] P M Athanas, H S Silverman. Processor reconfiguration through instruction set metamorphosis[J]. IEEE Computer, 1993, 26(3): 11-18.

[2] J R Hauser, J Wawrzynek, Garp. A MIPS processor with a reconfigurable coprocessor[A]. IEEE Symposium on FPGAs for Custom Computing Machines[C]. Los Alamitos: IEEE Computer Society Press, 1997. 24-33.

[3] M Arnold, H Corporaal. Designing domain specific processors [A]. International Conference on Hardware Software Codesign [C]. New York: ACM Press, 2001. 61-66.

[4] M Gschwind. Instruction set selection for ASIP design[A]. International Conference on Hardware Software Codesign[C]. New York: ACM Press, 1999. 7-11.

[5] Kubilay Atasu, Laura Pozzi, Paolo Ienne. Automatic application-specific instruction-set extensions under microarchitectural constraints[J]. International Journal of Parallel Programming, 2003, 31(6): 411-428.

[6] Philip Brisk, Adam Kaplan, Ryan Kastner, Majid Sarrafzadeh. Instruction generation and regularity extraction for reconfigurable processors[A]. Proceedings of CASES[C]. New York: ACM Press, 2002. 262-269.

[7] J-E Lee, K Choi, N Dutt. Efficient instruction encoding for automatic instruction set design of configurable ASIPs[A]. Proc ICCAD[C]. New York: ACM Press, 2002. 649-654.

[8] A Peymandoust, et al. Automatic instruction set extension and utilization for embedded processors[A]. IEEE International Conference on Application-Specific Systems, Architectures and Processors[C]. Los Alamitos: IEEE Computer Society Press, 2003. 108-118.

[9] N Clark, H Zhong, S Mahlke. Processor acceleration through automated instruction set customization[A]. Proceedings of the 36th Annual International Symposium on Microarchitecture [C]. New York: ACM Press, 2003. 129-140.

[10] Jason Cong, Yiping Fan, Guoling Han, Zhiru Zhang. Application-specific instruction generation for configurable processor architectures[A]. International Symposium on Field Programmable Gate Arrays[C]. New York: ACM Press, 2004. 183-189.

[11] Pan Yu, Tulika. Characterizing embedded applications for instruction-set extensible processors[A]. Design Automation Conference[C]. ACM Press, 2004. 723-728.

[12] Nathan T Clark, Hongtao Zhong, Scott A. Mahlke. Automated custom instruction generation for domain-specific processor acceleration[J]. IEEE Transactions on Computers, 2005, 54(10): 1258-1270.

[13] N Clark et al. An architecture framework for transparent instruction set customization in embedded processors[A]. Proc of the 32nd Annual International Symposium on Computer Architecture[C]. New York: ACM Press, 2005. 272-283.

[14] Jan Hoogerbrugge, Henk Corporaal. Transport-triggering vs. operation-triggering[A]. Proceedings of the International Conference on Compiler Construction[C]. London: Springer-Verlag, 1994. 435-449.

作者简介:

吕雅帅 男,1981年生于河北邯郸,国防科学技术大学计算机科学与技术学院博士.研究方向为系统结构及相关编译技术.

E-mail: freelancer_lys@163.com