一种针对位操作密集应用的扩展指令自动选择方法

张吉豫[1,2],刘先华[1,2],谭明星[1,2],程 旭[1,2],丛京生[2,3]
(1.微处理器及系统教育部工程研究中心,北京 100871;2.北京大学信息科学技术学院,北京 100871;
3.PKU-UCLA 理工联合研究所,北京 100871)

摘 要: 本文提出一种结合位操作分析和变换的扩展指令自动选择方法.该方法在数据流图中引入新的位操作中间表示结点,可精简地描述位访问操作.编译器可对程序数据流图进行选择性循环展开和位操作分析优化,并将其转换为带有直接表示位赋值操作结点的数据流图.实验结果表明,基于新的数据流图进行扩展指令选择可有效提升位操作密集型应用的性能.

关键词: 指令系统扩展;自动选择;位操作变换
中图分类号: TP314 **文献标识码:** A **文章编号:** 0372-2112 (2012)02-0209-06
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2012.02.001

Automatic Instruction-Set Extension for Bitwise Operation-Intensive Applications

ZHANG Ji-yu[1,2], LIU Xian-hua[1,2], TAN Ming-xing[1,2], CHENG Xu[1,2], CONG Jing-sheng[2,3]
(1. *Engineering Research Center of Microprocessor & System, Ministry of Education, Beijing* 100871, *China*;
2. *School of Electronics Engineering and Computer Science, Peking University, Beijing* 100871, *China*;
3. *UCLA-PKU Joint Research Institute in Science and Engineering, Beijing* 100871, *China*)

Abstract: This paper describes a new automatic instruction-set extension approach with bitwise operaton analysis and transformations. It introduces a new intermediate bit-manipulation operation to directly describe bitwise accesses. It performs bit-level analysis and optimization, and builds an extended data-flow graph with the new bit-manipulation operations. Experimental results show that extending instruction-set selection based on the new data-flow graph can effectively improve the performance for bitwise computation-intensive applications.

Key words: instruction-set extension; automatic selection; bitwise operation transformation

1 引言

近年来,面向特定应用的指令系统扩展技术日益受到学术界和工业界的关注.从高级语言编写的源程序出发自动选择和生成扩展指令,可有效降低指令系统扩展设计的复杂度,极大缩短专用指令系统处理器的开发周期,从而可在较低开销下显著提高应用程序核心代码的性能[1],因此得到越来越广泛的关注.

位操作密集型应用在许多领域中大量存在[2].但如 C++ 等典型高级编程语言中不支持直接对数据的某些位进行读写的描述,而是使用移位、按位与、按位或等操作的组合来实现.这使得在面向位操作密集型代码进行扩展指令自动选择时,一方面算法将难以进行准确的位操作相关面积和延迟估算,另一方面庞大的操作结点数将带来过长的运行时间,从而使算法无法取得更高效率和更优结果.

针对扩展指令自动选择算法应用于位操作密集应用时存在的上述问题,本文提出并实现了一种结合位操作变换的扩展指令自动选择方法.实验表明,对于位操作密集型应用,该方法可以更精简地表示位的读取和赋值,有效减小数据流图的规模,辅助扩展指令选择算法更精确地估算指令面积和延迟,从而使扩展指令选择算法可在更小的总面积约束下取得更高的加速比,并减少算法执行所需时间.

2 相关工作

扩展指令自动选择已成为近年来处理器设计的研究热点之一[3~6].最优扩展指令集合的选择是 NP 难的问题[5],相关工作通常采用迭代选取一条或几条当前最佳扩展指令等方法,并致力于提高算法效率.而近几

收稿日期:2011-01-15;修回日期:2011-05-16
基金项目:国家"核高基"重大科技专项课题(No.2009ZX01036-001-003,No.2009ZX01029-001-002)

年,针对扩展指令选择的编译技术日益受到关注[1,7~9].传统编译优化主要针对固定的指令系统来改进程序性能、功耗或代码大小.针对扩展指令自动选择引入新优化策略或对现有优化策略进行调整,可有效挖掘重要优化机会,从而提高扩展指令自动选择的效果.针对这一目标,Peymandoust 等[10]在编译器中引入特定算术运算变换将复杂的多项式表示进行分解.Bonzini 等[7]采用编译优化空间探索的方法确定 IF 转换和循环展开这两项技术是否应使用以及该如何进行参数配置.Bennett 等[8]和 Murray 等[1]提出了结合源代码级别程序变换的扩展指令自动选择的方法.

上述相关工作指出,扩展指令自动选择的效果与代码形态密切相关.针对扩展指令自动选择进行的编译优化工作往往与传统编译优化存在很大不同;且在特定的编译优化后再进行扩展指令自动选择可带来更多性能提升.目前配合扩展指令选择的位操作优化研究仍不充分.本文选择位操作密集应用作为实例开展研究,取得了一些有益的结论.

3 问题分析及数据流图扩展设计

3.1 问题分析

典型高级语言和编译器中位访问的表示方法给扩展指令自动选择带来了以下问题:

(1)过高的位操作延迟和面积估计可能导致扩展指令选择算法得不到最优解.

在扩展指令自动选择过程中,对指令使用面积和延迟的准确快速估算是取得优化结果和高性能的关键环节.对使用移位、与、或来描述位访问的数据流图进行扩展指令选择时,其数据流图表示不能反映其实现为硬件时的面积和延迟.这很可能导致对位操作的面积和延迟估计过高,从而不能选出实际上的最优指令.

(2)过多的结点数会严重增加扩展指令选择算法的运行时间.

扩展指令自动选择算法的时间复杂度往往是结点数的指数级函数,数据流图中结点的个数直接影响完成扩展指令选择的总时间.在常见的高级语言表示中,每一位的读取和赋值均需进行一系列移位、与、或操作,对 32 位宽的字进行置换所需的指令就超过 100 条.这严重影响扩展指令选择算法执行的时间.

(3)循环限制了扩展指令选择范围,而激进的循环展开会严重增加数据流图结点数.

许多高级语言程序中使用循环来描述对各位域进行的有规律的操作.不进行循环展开将使得扩展指令选择只能在循环内的基本块中进行,限制了扩展指令选择的效果.针对扩展指令自动选择的编译优化相关研究[1,7,8]提出更激进地使用循环展开,但会极大地增加基本块内指令数,严重增加算法运行时间.在扩展指令自动选择和程序代码优化效果之间应良好进行权衡.

3.2 位操作结点设计

针对上述问题,本文提出在数据流图中引入一种位操作结点 BM(Bit Manipulation).该操作结点可按如下形式表示:

$$z = BM.n(x, y, \boldsymbol{pmt\_vec}) \tag{1}$$

式(1)中 z 的位宽为 n,x 的位宽大于或等于 n,该结点的操作语义表示按照常数向量 ***pmt_vec*** 所指示的方式,使用操作数 y 中的若干位替换操作数 x 中的相应位,并输出结果到 z.式中 ***pmt_vec*** 是形为$\langle p_0, p_1, p_2, \cdots, p_{n-1}\rangle$的向量,若 $p_{i(0\leqslant i<n)}=0$,则将 x 的第 i 位赋给 z 的第 i 位,否则将 y 的第 p_i-1 位赋给 z 的第 i 位.图 1 展示了 $BM.n$ 操作语义的伪码描述.

```
for i from 0 to n - 1
    if (pmt_vec[i] = = 0)    z[i] = x[i];
    else                     z[i] = y[pmt_vec[i] - 1];
endfor
```

图 1 $BM.n$ 操作语义描述

BM 结点可简洁地表示单一位的读取和赋值、位收集(bit gather)和位散布(bit scatter)、位置换等常见的位操作.

4 结合位操作变换的扩展指令自动选择

本文提出一种结合位操作优化和变换的扩展指令自动选择方法.该方法令编译器在数据流图中可以使用 BM 结点来直接表示位操作,并对程序中的位操作进行优化,从而进一步改善扩展指令自动选择的效果.

算法流程如下:首先将仅含位操作的数据流图转换为位值流图来进行位操作优化;之后将其转换为使用 BM 结点来直接表示位操作的数据流图;接下来在新的数据流图上进行扩展指令选择;最后输出扩展指令集合并完成程序编译和扩展指令模拟模块自动生成.本节分别对算法各主要部分进行介绍.

4.1 基于位值流图的位操作优化

本文算法首先将程序中的循环进行选择性的循环展开.若一个循环体中包含移位或逻辑运算操作,且移位操作的移位量为常量或循环迭代变量与常量之间的运算,同时位操作相关结点占该循环体结点的 50% 以上,则尝试将该循环进行循环展开.接下来本文算法提取程序数据流图中仅包含逻辑运算及移位操作的子图,并构建位值流图(Bit-Flow Graph,BFG).位值流图是表示操作之间的位值依赖关系的有向图,其中每个结点和边表示的是单一位之间的数据相关[11].对每个仅

包含移位及逻辑操作的数据流图子图根结点,算法将建立一组位值流图结点指针,分别指向数据流图根结点中各个位所对应的位值流图结点.数据流图根结点所对应的位值流图结点的组织结构如图 2 所示.

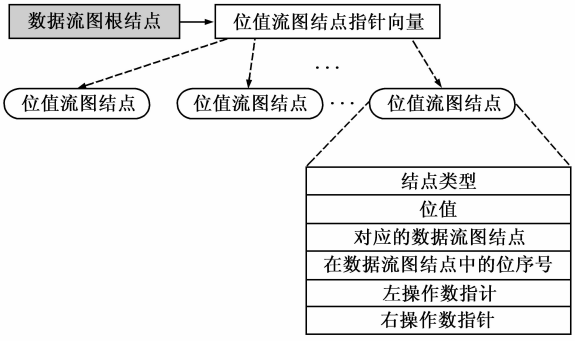

图2　位值流图结点的组织及结构

在位值流图构建完成之后,进行位级别的常数传播和运算强度降低等方法将位值流图进行优化.

4.2　位值流图转换为扩展的数据流图

在建立并优化位值流图后,本文算法将位值流图转换为带有 BM 结点的数据流图.为使转换后的数据流图更精简,算法设计中进行了如下优化考虑:所有具有固定数值的位值流图结点可组成一个数值,作为 BM 结点的第一个源操作数;所有使用同一数据流图结点中的不同位对输出进行赋值的操作应在同一个 BM 结点的操作语义中进行.

算法 1　将位值流图转为扩展数据流图的算法
Algorithm: Transform_BFG_to_DFG_with_BM

```
Input: vRoot — a vector of BFG nodes for a DFG root
Output: a pointer to the root of the generated DFG
Trans_to_DFG
Begin
   nv ← const_nodes_in_vector(vRoot);
   //vRoot 中常量结点构成 nv 向量
   rest_vRoot ← vRoot − nv;   //从 vRoot 中去掉 nv 中结点得到 rest_vRoot
   pRoot ← gen_node (CONST, value_of(nv));
   //生成常量数据流图结点 pRoot,若 nv 为空则值设为 0
while ! empty( rest_vRoot) do //遍历处理 BFG 中结点
   tmp_nv ← nodes_of_same_type(rest_vRoot);
   //选择所有具有相同类型或来自同一变量的位结点
   tmpNode(Part_trans_to_DFG(tmp_nv);//产生子数据流图
   pmt_vec(gen_pmt(vRoot, tmp_nv);//产生位置向量
   pRoot ← gen_node(BM, pRoot, tmpNode, pmt_vec);
   //产生 BM 结点来进行位赋值
   rest_vRoot ← rest_vRoot − tmp_nv;   //去掉已处理过的 BFG 结点
   endwhile
   return pRoot;
end
```

算法 1 用伪码进行了算法描述.每个仅含位操作的数据流图子图的根结点中的每个位都对应一个 BFG 结点,这些结点构成了一个 BFG 根结点向量并作为参数传入.算法从根结点向量 ***vRoot*** 开始遍历整个 BFG.对于每个 BFG 结点向量,首先将其中的常量结点合并成一个常量 DFG 结点;接下来依次选择所有具有相同操作类型的 BFG 结点或对应同一变量中若干位的 BFG 结点,调用 *Part_trans_to_DFG* 生成子数据流图以及相应的 BM 结点.其中 *Part_trans_to_DFG* 函数的输入为类型相同的 BFG 结点构成的向量,该函数根据给定的 BFG 结点的运算类型,首先处理给定的 BFG 结点向量中所有结点的左操作数结点所构成的向量,递归调用自身生成代表左操作数的数据流图;若存在右操作数,则同样调用自身生成代表右操作数的数据流图;最后生成代表相应操作的数据流图结点.

4.3　扩展指令自动选择及模拟实现

完成位级别分析及转换后,本文算法采用 Pozzi 等提出的迭代选择算法[4](以下简称为 PAI 迭代算法)进行扩展指令选择.在程序数据流图中搜索每个基本块内的最佳扩展指令,在这些候选扩展指令中选出收益最大者加入扩展指令集合,之后使用选出的扩展指令更新数据流图,并在相应的基本块中重新选择扩展指令加入到候选集合中.重复这一过程,直至达到扩展指令数限制或总面积限制或没有更多的扩展指令可以选出为止.

在完成扩展指令选择之后,本文方法将自动生成扩展指令模板,并根据该模板自动生成模拟器用于快

Template Description: ExtensionInstr.templ

```
int cost_inst_i = … //扩展指令所用周期数
int reg_use_inst_i = … //扩展指令的输入寄存器个数
// 获得目标寄存器号
int get_dst_inst_i (unsigned int instruction) {…}
// 获得输入操作数 1 的寄存器号,如无,则返回 − 1.下同
int get_input1_inst_i (unsigned int instruction) {…}
int get_input2_inst_i (unsigned int instruction) {…}
int get_input3_inst_i (unsigned int instruction) {…}
void exec_inst_i (unsigned int instruction)
{                    // 扩展指令的执行语义描述
   int input1, input2, input3;
   input1 = Register[get_input1_inst_i (instruction)];
   input2 = Register[get_input2_inst_i (instruction)];
   input3 = Register[get_input3_inst_i (instruction)];
   result = … //用于描述指令具体操作语义的代码
   Register[get_dst_inst_1(instruction)] = result;
}
…
```

图 3　扩展指令模板文件构成

速反馈评价.本文方法对扩展指令的功能模拟实现进行了划分,将扩展指令的取指和初步译码在模拟器核心代码中模拟实现,其余部分在由扩展指令选择过程生成的扩展指令描述文件中实现.该文件同时提供在周期级精确模拟时需要的指令执行周期数及输入输出寄存器号.根据该模板可自动生成扩展指令的模拟模块,以用于快速代码功能测试及性能评测.该模板由如图 3 所示的三部分组成:扩展指令的执行周期、扩展指令的输入输出寄存器号以及扩展指令的具体操作语义.此处所用周期数为扩展指令选择算法估算的结果,即最长路径上各结点的延迟之和.模拟器通过指令解析,根据操作码调用相应的解释执行函数来模拟该指令的功能,i 为扩展指令序号.

5 性能评测与结果分析

本文算法基于 LLVM 编译框架[12]实现.实验中采用 UniCore-2 处理器[13]作为基准处理器.UniCore-2 是一个 32 位的类 RISC 处理器,具有 32 个整点寄存器和 32 个单精浮点寄存器,寄存器堆具有 3 个读口和 1 个写口,经 TSMC 65nm 工艺流片的频率为 1.0-1.3GHz.在本文的实验中设置扩展指令数上限为 16 条.实验使用的硬件信息库参数见表 1,来源于 TSMC 65nm 工艺下的真实综合结果.本文的所有评测实验均在一台配有 2.0GHz AMD Opteron 处理器和 4GB 内存的机器上进行.所采用的评测程序选自 Mibench[14]中位操作比重超过 50%的应用或模块,并增补了常见的 3DES 加解密程序.评测程序列表及概要在表 2 中进行了介绍.

表 1 硬件信息库参数

操作(32 位)	面积(μm^2)	延迟(ns)	操作(32 位)	面积(μm^2)	延迟(ns)
加	1277	0.2	或	460	0.012
减	1524	0.2	非	46	0.005
乘	28124	1.87	异或	675	0.019
与	399	0.012	移位	1235	0.12

表 2 评测程序

评测程序	描述
RevBits32	将一个 32 位的字中的值倒转.
GwaveUnPK	GSM 06.10 解码中的波解包模块.
GwavePK	GSM 06.10 编码中的波打包模块.
BitCNT2	统计给定输入中值为 1 的位的个数.
Blowfish	一种数据加密算法.
3DES	数据加密标准,一种数据加密算法.
SHA	安全散列算法,一种数据加密算法.

5.1 不同的总面积约束下的性能比较

本文首先使用 GCC 4.2 编译器用"-O2"标准优化选项对各评测程序进行优化,之后将使用本文算法(用 BTO+PAI 表示)得到的结果与 PAI 迭代算法(用 PAI 表示)以及选择性循环展开加 PAI 迭代算法(用 Unroll+PAI 表示)所得到的结果进行了比较.为使评测更接近真实情况,本文对其它方法最终得到的带有位操作的指令也进行了位操作分析,从而更准确地分析其实际延迟.

图 4 给出了在同样面积约束下三种方法所分别取得的加速比.在此,我们选择了在本文算法取得最终结果时所用的面积作为代表性面积约束.应用本文方法选出的扩展指令对这些评测程序进行优化,可以获得 22.7% 至 98.9%的性能提升,对于所有的评测程序,相对其他方法,本文算法都可以在更小的面积约束下完成扩展指令选择.在特定面积的约束条件下,单纯使用 PAI 迭代算法只能获得 2.1%至 64.3%的性能提升,而本文算法所取得的性能可达到其 1.4 倍至 21 倍.从使用选择性循环展开之后的结果来看,循环展开只能对 RevBits32 和 3DES 等少数特征性较强的程序带来优化潜力,对其它测试程序则几乎没有影响.在面积约束下,利用选择性循环展开结合 PAI 算法仅将 RevBits32 的性能提升了 37.1%,而本文算法取得的性能加速可以达到 PAI 迭代算法相应效果的 2 倍.

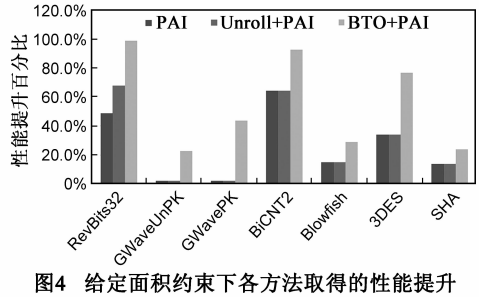

图4 给定面积约束下各方法取得的性能提升

在不加面积约束的情况下,本文算法所取得的性能提升的百分比与其它两种方法的对比如图 5 所示.对于 RevBits32 和 3DES,本文算法取得的性能提升分别是迭代法的 2.0 倍和 2.2 倍,其中一个重要原因是本文方法在编译阶段采用的选择性循环展开可以将这两个应用中含较多位操作的关键循环进行展开和分析优化.而单纯使用选择性循环展开,对于 RevBits32 可以获得与本文算法相同的性能提升,但对于 3DES 则由于结点数过多导致无法在合理时间范围内得到结果.由于本文算法对扩展指令延迟进行了更加准确的估算指导,从而可以得到更优的选择结果,对于 GWaveUnPK、GWavePK 和 SHA 等程序,本文算法取得的性能提升分别是 PAI 迭代算法的 1.08 倍、1.09 倍和 1.76 倍.对于 BitCNT2 和 Blowfish 等程序,本文算法也可以取得不低于与其它算法的效果.

性能评测的结果表明,引入 BM 结点后的位操作变换策略可以更好地支持面向位操作密集型应用的扩展

指令选择工作.本文算法应用于位操作密集程序时,可以在更小的总面积约束下取得比其它两种方法更高的加速比.在不加总面积约束的情况下,本文算法通过选择性循环展开、对位操作的优化和变换以及更精确的指令延迟估算,可以取得比其它两种方法更高的加速比.

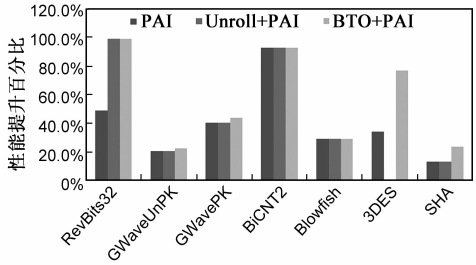

图5　不加面积约束时各方法取得的性能提升

5.2　数据流图结点数及扩展指令选择所用时间比较

数据流图结点数直接影响到算法的运行效率.本文对各方法中在正式进行扩展指令选择之前的数据流图结点数进行了统计.图中展示了归一化到 PAI 方法的评测结果,即三种方法中数据流图结点数与 PAI 方法中的比值.可以看到,Unroll + PAI 方法虽然可以带来性能上的一定改进,但对于 ReverseBits32 和 3DES 等程序,会严重增加数据流图中的结点数.本文的 BTO + PAI 的方法对于大部分程序都可以有效减少数据流图结点数,整体相较 PAI 方法可平均减少约 10%的结点数.

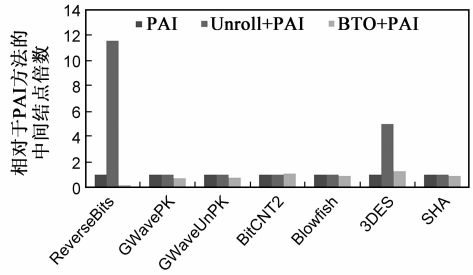

图6　数据流图结点数比较

采用三种算法进行数据流图优化及扩展指令选择所需的总体时间比较情况如图所示,反映了各方法的运行时间归一到 PAI 方法后的情况.对于 Unroll + PAI 方法,ReverseBits 和 3DES 程序的扩展指令选择的运行时间会明显增加,这是由于数据流图结点的严重增加所导致的.整体而言,相对于标准 PAI 方法,本文方法可以在提升性能的同时,平均减少约 11.4%的算法运行时间.

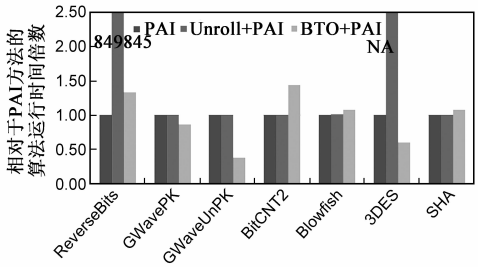

图7　编译所用时间比较

6　结论

充分挖掘特定应用程序的特性,从中抽取并设计特定运算结点对数据流图中间表示进行扩展,结合编译策略可以更好地进行支持指令选择工作.本文针对当前扩展指令选择算法应用于位操作密集型程序时存在的问题,提出并实现了一种结合位级别分析和变换的扩展指令自动选择方法.实验结果表明,本文算法应用于位操作密集程序时,可以在更小的总面积约束下取得更高的加速比.

参考文献

[1] Murray A C, et al. Code transformation and instruction set extension[J]. ACM Transactions on Embedded Computing Systems, 2009, 8(4): 1 – 31.

[2] Hilewitz Y, Lee R B. Fast Bit Gather, Bit scatter and bit permutation instructions for commodity microprocessors[J]. Journal of Signal Processing Systems, 2008, 53(1 – 2): 145 – 169.

[3] Cong J, et al. Application-specific instruction generation for configurable processor architectures[A]. International Symposium on Field Programmable Gate Arrays[C]. Monterey: ACM Press, 2004. 183 – 189.

[4] Pozzi L, et al. Exact and approximate algorithms for the extension of embedded processor instruction sets[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2006, 25(7): 1209 – 1229.

[5] 吕雅帅,沈立,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.
Lv Yashuai, Shen Li, et al. Automatic instruction set extension for embedded applications[J]. Acta Electronica Sinica, 2008, 36(5): 985 – 988. (in Chinese)

[6] Verma A K, et al. Fast, nearly optimal ISE identification with I/O serialization through maximal clique enumeration[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2010, 29(3): 341 – 354.

[7] Bonzini P, Pozzi L. Code transformation strategies for extensible embedded processors[A]. International Conference on Compilers, Architecture and Synthesis for Embedded Systems[C]. Seoul: ACM Press, 2006. 242 – 252.

[8] Bennett R V, et al. Combining Source-to-source Transformations and Processor Instruction Set Extensions for the Automated Design-space Exploration of Embedded Systems[A]. ACM Conference on Languages, Compilers, and Tools for Embedded Systems[C]. San Diego: ACM Press, 2007. 83 – 92.

[9] Kawahito M, et al. A New Idiom Recognition Framework for Exploiting Hardware-assist Instructions[A]. International Conference on Architectural Support for Programming Languages and Operating Systems[C]. San Jose: ACM Press, 2006. 382 – 393.

[10] Peymandoust A, et al. Automatic Instruction Set Extension and Utilization for Embedded Processors[A]. International Conference on Application-Specific Systems, Architectures and Processors[C]. Hague: ACM Press, 2003. 108 – 118.

[11] Zhang J, et al. Bit-Level Optimization for High-Level Synthesis and FPGA-Based Acceleration[A]. International Symposium on Field Programmable Gate Arrays[C]. Monterey: ACM Press, 2010.

[12] Lattner C, Adve V. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation[A]. International Symposium on Code Generation and Optimization[C]. Palo Alto: IEEE Computer Society Press, 2004. 75 – 88.

[13] Cheng X, et al. Research progress of UniCore CPUs and PKUnity SoCs[J]. Journal of Computer Science and Technology, 2010, 25(2): 200 – 213.

[14] Guthaus M R, et al. MiBench: a free, commercially representative embedded benchmark suite[A]. IEEE International Workshop of Workload Characterization[C]. Austin: IEEE Computer Society, 2001. 3 – 14.

作者简介

张吉豫 女,1982 年生于吉林长春,北京大学信息科学技术学院博士生. 主要研究领域为软硬件协同设计及编译优化.

刘先华(通信作者) 男,北京大学信息科学技术学院讲师,主要研究领域为处理器设计及编译优化.

E-mail: lxh@mprc.pku.edu.cn