第 36 卷　第 20 期　　计　算　机　工　程　　2010 年 10 月
Vol.36　No.20　　Computer Engineering　　October 2010
·博士论文·　　文章编号：1000—3428(2010)20—0001—03　　文献标识码：A　　中图分类号：TP309.2

面向安全与加密应用的定制指令设计

薄　拾，葛　宁，林孝康
(清华大学电子工程系，北京 100084)

摘　要：为设计灵活高效的安全加密处理器件，提出一种多目标联合定制指令设计方法。该方法通过挖掘加密算法之间的公共频繁计算模式，提高定制指令的适应性。采用该方法针对 DES、AES、Blowfish、RC4 和 MD5 等主流加密算法进行定制指令设计实验，结果表明其可能有效地完成定制指令设计，指令集扩展后，5 种算法获得了 4841.9%~63.6%的性能提升。

关键词：安全；加密；定制指令；频繁计算模式；多目标联合设计

Custom Instruction Design for Security and Encryption Applications

BO Shi, GE Ning, LIN Xiao-kang
(Department of Electronic Engineering, Tsinghua University, Beijing 100084, China)

【Abstract】 In order to design flexible and efficient security and encryption processing device, this paper proposes a multi-objective joint custom instruction design method. By discovering common frequent computation patterns hiddening in encryption algorithms, the flexibility of custom instructions is raised. Custom instructions aimed aiming at several five encryption algorithms including DES, AES, Blowfish, RC4 and MD5 are generated by adopting the proposed method. Experimental result shows that the proposed method can fulfilled fulfill custom instruction design effectively. After instruction- set extension, forthe performance of the five algorithms DES, AES, Blowfish, RC4 and MD5, is performance improvementsd from by 4841.9% to ~63.6% are obtained respectively.

【Key words】 security; encryption; custom instruction; frequent computation pattern; multi-objective joint design

1　概述

随着信息网络的发展，特别是无线网络和电子商务的广泛应用，保障信息安全成为系统设计中的重要内容。近 20 年来，安全加密技术取得了长足进步，随着安全性的提高，信息安全处理也给通信设备带来越来越高的计算负担。另外，由于通信网络中多种协议、标准同时共存，从兼容性方面考虑，往往要求系统能够灵活支持多种安全加密技术。

不同于传统的专用硬件和通用可编程处理器，可配置处理器通过引入自定义的业务专用指令获得性能与灵活性之间更好的折中。此前，已有研究者[1-2]开展了安全加密专用指令的设计研究，这些工作大多依靠人工完成，一方面需要设计者具有较深的知识背景，另一方面其设计质量无法保证。国内外一部分研究者已经注意到定制指令设计自动化的重要性，并获得了一定进展成果[3]，但这些研究存在一个共同的问题：设计流程只能针对单个应用，无法实现面向多应用目标的协同设计。

本文提出一种基于频繁计算模式发现思想的定制指令自动设计方法，该方法能够完成面向多应用目标的定制指令设计。并然后针对著名的 DES、AES、Blowfish、MD5 和 RC4 算法在 MIPS32 指令集基础上进行指令集扩展。实验结果显示，本文方法能够有效完成定制指令设计，灵活实现多种安全加密算法的性能加速。

2　安全加密算法

系统往往需要满足三方面的信息安全要求：机密性，完整性和可用性。根据具体应用的不同需要，研究者提出了多种算法和解决方案。从安全机制上，这些方法可以分为 3 类：对称密钥加密，非对称密钥加密以及散列法。从操作模式上通常分为分组模式和流模式。从数学原理上，常见方法有 Feistel 及非 Feistel 密码、大整数因子分解系统、椭圆曲线离散对数系统等。从应用角度，完整的安全服务往往需要多种算法配合使用。例如 SSL 套件 SSL_RSA_WITH_RC4_128_MD5 就需要 RSA 算法完成密钥交换、RC4 算法负责数据加密/解密、MD5 算法保证信息完整性。

尽管安全加密算法看起来各不相同，但总体上这些算法都具有数据密集和计算密集的特点。本文对算法执行过程中的指令消耗分布进行了统计，从图 1 可以看出，算法基本都消耗了大量的逻辑运算、移位操作以及存储访问指令。

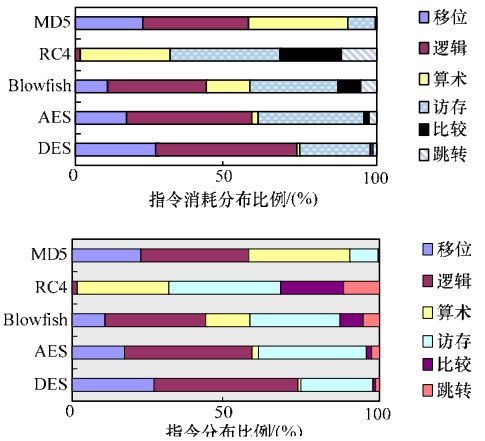


图 1　5 种算法的指令消耗分布

基金项目：国家“863”计划基金资助项目(2007AA01Z2b3)；国家“973”规划计划基金资助项目(2007CB310608)
作者简介：薄　拾(1982－)，男，博士研究生，主研方向：专用指令集设计，可重配置计算；葛　宁、林孝康，教授、博士生导师
收稿日期：2010-03-25　**E-mail：**boshi99@mails.tsinghua.edu.cn

在计算模式上，安全加密算法也存在着一些共性，如算法大多包含模加/减、有限域乘法、有限域加法、换位盒及换字盒。计算模式共性的发掘有助于设计灵活普适的定制指令。

3 定制指令设计方法

图 2 给出了本文提出的定制指令自动设计方法的基本框架。整个框架分为 3 个主要部分：(1)算法分析模块，实现算法程序代码自动分析、热点提取和计算核心库构建。(2)频繁计算模式发现模块，实现从计算核心库中挖掘有限数量的频繁计算模式。(3)指令硬件及目标代码生成模块，分别完成定制指令硬件综合和目标代码生成。

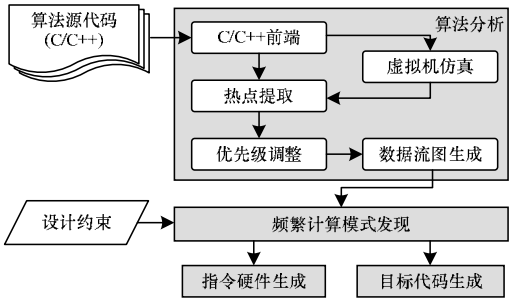


图 2 定制指令设计流程

3.1 算法分析

算法分析模块的核心任务是自动提取目标算法程序的计算热点。首先，算法程序的 C/C++源代码经由编译前端 llvm-gcc[4]转化为机器架构无关的中间表达形式；接着通过 lli[4]虚拟机执行中间表达形式，获得代码的动态信息；根据各代码段的调度频率信息，提取频繁调度的基本块作为程序的计算热点。需要指出的是，在进行多目标定制指令设计时，还需要对多个算法程序的热点基本块进行优先级加权调整。为便于后续处理，最后将热点基本块的中间表达形式转化为数据流图，这些带有频率信息的数据流图最终组成计算核心库。

3.1.1 热点提取

本文中的计算热点是指程序执行过程中消耗大量指令周期的基本块代码。热点提取的依据为基本块所消耗的指令周期及其与程序总的指令周期消耗的相对关系。

对于程序 app_i 的任意基本块 $b_{i,j}$，其内部包含的指令个数为 $I_{i,j}$，执行次数为 $f_{i,j}$，于是，程序执行期间基本块 $b_{i,j}$ 消耗的指令周期 $C_{i,j}$ 为：

$$C_{i,j} = I_{i,j} f_{i,j} \tag{1}$$

程序 app_i 消耗的总指令周期 C_i 为其所有基本块消耗的指令周期数之和：

$$C_i = \sum_{j=1}^{m} C_{i,j} \tag{2}$$

为提取其中的主要成分，可对程序 app_i 的所有基本块进行排序，即得到 $C_{i,1} \ge C_{i,2} \ge \cdots \ge C_{i,m}$。这里引入了影响因子 $\alpha \in (0,1)$，通常取 $\alpha = 0.9$，。满足下式的排名最前的 x 个基本块即 app_i 的计算热点：

$$\begin{aligned} &\min \ \ x \\ &\text{s.t.} \ \ \sum_{j=1}^{x} C_{i,j} \ge \alpha \times C_i \end{aligned} \tag{3}$$

3.1.2 优先级调整

算法程序的动态信息是在程序测试执行中得到的，。由于测试向量规模不同，各程序所消耗的时间也不同，有时甚至相差很大。之后的频繁计算模式发现模块的工作原理决定了代码消耗的指令数越多其受关注的程度就越高。因此，当执行多目标定制指令设计时，必须调整目标程序间的优先级关系。这种调整可由设计者根据实际需要人为指定，也可由工具根据特定规则自动完成。

进行自动调整时，平均是最常见的规则。假设现有 n 个算法程序 $\{app_i \mid i = 1,2,\cdots,n\}$，其中，任意 app_i 消耗的总指令周期数为 C_i，引入加权因子 β_i，平均原则下 β_i 可用下式的计算公式为 β_i：

$$\beta_i = \frac{\max\limits_{1 \le i \le n} \{C_i\}}{C_i} \tag{4}$$

进而对 app_i 中各基本块的频率进行加权：

$$f'_{i,j} = \beta_i f_{i,j} \tag{5}$$

可以发现，平均化加权处理后各算法程序将具有相同的指令周期消耗，然而各程序内部基本块间的相对关系仍保持不变。优先级调整后，来自各算法程序的热点基本块将被转化为数据流图的形式，并构成计算核心库。

3.2 频繁计算模式发现

定制指令模式的提取主要采用对计算热点的数据流图先分解再重新覆盖的方式实现[3]，这类方法存在 2 个问题：(1)流图分解将产生大量候选模式，时间和空间复杂度很高。(2)流图覆盖是一个 NP 完全问题，求解困难。

本文采用图数据挖掘的思想来解决定制指令模式提取问题。把程序计算核心库作为一个图数据库，频繁计算模式发现模块将从其中发现的感兴趣的计算模式作为定制指令的行为级描述，这些模式实际上是计算核心库中流图的子图。

本文感兴趣的计算模式应具有较高较多的调用次数和合适的规模，由此构造启发函数：

$$h(p) = \left(I_p - 1\right) f_p \tag{6}$$

其中，I_p 为模式 p 内部包含的指令节点个数；f_p 为其出现的频率。$h(p)$ 显示了计算模式 p 作为定制指令而具有的指令周期削减能力。

与一般频繁子图挖掘[5]不同，模式 p 的频率 f_p 定义为：

$$f_p = \sum_{b \in GDB} S_{p,b} f'_b \tag{7}$$

其中，b 为计算核心库 GDB 中的任意数据流图；$S_{p,b}$ 为存在于 b 中且与 p 同构的子图数量；f'_b 为 b 自身的调用频率。

利用启发函数 $h(p)$ 可以构造频繁计算模式发现流程，其步骤如下：(1)以 GDB 中各图的单个节点为初始模式。(2)对每个模式进行各种可能的扩展，每次扩展只向现有模式添加一个新节点，若现有模式都无法扩展，则跳至(5)。(3)对扩展所得的新模式重新归类，计算每个新模式的 $h(p)$ 作为其分值，并记录本轮扩展产生的分值最高的新模式。(4)按照分值，保留分值最高的 $beamwidth$ 个新模式供下轮扩展，跳至(2)。(5)输出各轮扩展中最高分的模式。

这里采用了集束搜索的思想，由于每轮模式扩展会产生大量的新模式，为减轻计算和存储负担，每次只允许有限数量(即 $beamwidth$)的最优模式参与下一轮扩展。类似方法也曾被其他数据挖掘算法[5]所采纳。尽管集束搜索体现了局部最优的思想，但实践证明，对于大规模问题它确实是一种快速且有效的方法。

从式(6)(6)可以看出，受 I_p 和 f_p 的共同影响，在启发函

数 $h(p)$ 的引导下，模式发现过程将趋于挖掘那些包含节点数较多且出现频率较高的计算模式。由于模式的频率是在整个计算核心库上统计得到的，因此那些隐藏在多个算法程序之中的同构计算模式将脱颖而出。这种善于挖掘程序之间共性的特质正非常适合完成面向多目标程序的定制指令设计工作。

4 安全加密定制指令设计

为满足现代通信设备对多业务、多协议支持的需要，安全加密专用处理器必须具有灵活和快速处理多种加密算法的能力。本文选择分组加密算法(DES、AES、Blowfish)、流加密算法(RC4)以及安全散列算法(MD5)为作为优化目标，在MIPS32 指令集的基础上进行指令集扩展。

首先将上述 5 种算法程序送入虚拟机进行仿真运行，并根据运行过程记录的动态信息进行热点提取，此时影响因子取 $\alpha = 0.9$，热点提取结果见表 1。对于来自各个算法的计算热点数据流图，这里采用平均加权思路，即 $\beta_i = 0.2$。

表 1 热点提取结果

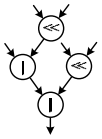
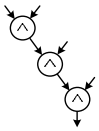
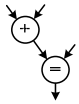
目标算法	全部代码		热点代码	
	基本块	代码量	基本块	代码量
DES	18	564	3	346
AES	94	3 117	4	1 142
Blowfish	43	829	5	436
RC4	16	103	3	52
MD5	32	1 191	1	927

从实际设计的角度考虑，定制指令必须满足一定的设计约束。，其中，包括：(1)定制指令数量 N_{CI} ：，由于指令编码空间和硬件面积的限制，只能增加有限数量的定制指令。(2)数据输入端口数 IP 和输出端口数 OP ：，由于片内通信能力的限制，每个定制指令只能拥有限定数量的数据端口。(3)指令时延 D ：，定制指令与基本指令一样，其计算时延不可过长，否则将影响指令流水。由于定制指令本质上是基本运算操作的聚类，因此时延由其内部最长计算路径所决定。(4)硬件面积 A ：，延时 D 虽然限制了定制指令的内部深度，但没有限制其规模，本文要求每条定制指令只能占用有限的硬件面积。

由于频繁计算模式发现模块工作时会产生大量候选模式，为节省设计时间，每个模块的 D 值和 A 值通过估算得到：首先给出每个原子操作的时延和面积估计，如定义与操作的 $D_{AND} = 1$、$A_{AND} = 1$，定义加法运算的 $D_{ADD} = 2$、$A_{ADD} = 2$。进而估计每个计算模式的 D 值和 A 值。

根据实际需要，这里设计约束被设定为：$N_{CI} = 10$，$IP = 6$，$OP = 3$，$D = 4$，$A = 6$。将 5 种目标算法的源码及设计约束送入定制指令设计流程，依次选出 10 个最优的定制指令模式。表 2 列出了这些定制指令的典型代表。可以发现，这些指令往往可被几种算法共同调用，说明定制指令具有一定的灵活性和普适性。

表 2 典型的定制指令

指令名称	计算模式	目标算法
SHL_OR_2		DES、MD5
XOR_3		RC4、MD5
ADD_CMP		AES、Blowfish、RC4、MD5

定制指令的引入带来了处理性能的显著提高。本文对几种 5 种算法进行了指令集仿真，并比较了指令集扩展前后每种算法所消耗的指令周期数。表 3 显示，定制指令的使用大大降低了算法对指令周期的消耗，这是由于大部分计算任务都由定制指令所承担。从指令周期消耗的角度看，5 种算法分别获得了 4841.9%~63.6%的性能提升。从表 3 中还可以发现，每种算法都可以利用多条定制指令来获得性能的提高。这正是多目标联合设计的优势所在。

表 3 指令集扩展前后的性能对比

目标算法	指令周期消耗		可用定制指令	压缩率/(%)
	扩展前	扩展后		
DES	1.41×10^{10}	7.21×10^{9}	4	48.9
AES	2.62×10^{8}	1.15×10^{8}	4	56.0
Blowfish	2.91×10^{8}	1.69×10^{8}	3	41.9
RC4	8.29×10^{9}	3.01×10^{9}	5	63.6
MD5	3.70×10^{10}	1.39×10^{10}	6	62.5

从表 3 中还可以发现，每种算法都可以利用多条定制指令来获得性能的提高。这正是多目标联合设计的优势所在。进一步地，将多目标联合设计与各目标算法独立设计进行比较。在进行独立设计时，每种算法均根据自身需要独立提出 2 个定制指令模式，因此，定制指令模式总数量仍是 10 个。2 种设计方式带来的性能优化效果如图 3 所示，很明显，由多目标联合设计方式产生的定制指令能够节省更多的指令周期。原因在于，频繁计算模式发现过程会关注和发掘各算法计算模式的共性，因此，各目标算法可共享更多的定制指令，性能提升更为显著。

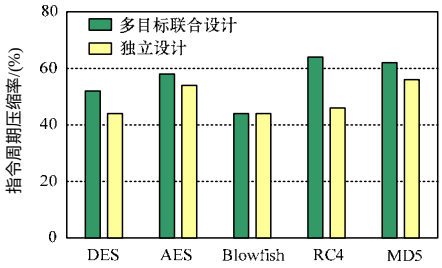


图 3 多目标联合设计与独立设计效果对比

5 结束语

针对现代通信与网络系统不断提高的安全需求，为使安全加密处理器能够灵活、快速处理多种安全加密算法，本文提出了一种多目标联合定制指令设计方法，并采用该方法针对 5 种典型的安全加密算法(DES、AES、Blowfish、RC4 和 MD5)进行定制指令设计。实验结果显示，由该方法设计产生的定制指令能够灵活支持和加速这 5 种算法的处理。通过挖掘安全加密算法之间计算模式的共性，多目标联合设计方式方法能够提高定制指令的灵活性和适应性，达到获得更好的性能优化效果。

参考文献

[1] Tillich S, Großschädl J. Accelerating AES Using Instruction Set Extensions for Elliptic Curve Cryptography[C]//Proc. of 2005

International Workshop on Information Security & and Hiding. Singapore: [s. n.]., 2005.

[2] Dai Z Bibin, Li Wei, Yang X Hiaohui, et al. The Research and Implementation of Reconfigurable Processor Architecture for Block Cipher Processing[C]//Proc. of ICESS'08. Hong Kong, China: [s. n.]., 2008.

[3] Atasu K, Pozzi L, Ienne P. 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.

[4] Lattner C, Adve V. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation[C]//Proc. of 2004 International Symposium on Code Generation and Optimization. Palo Alto, California, USA: [s. n.], 2004.

[5] Ketkar N S, Holder L B, Cook D J. Subdue: Compression-based Frequent Pattern Discovery in Graph Data[C]//Proc. of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations. Chicago, Illinois, USA: [s. n.]., 2005.

编辑　张　帆