面向密码流体系结构的超长指令字可重构研究

严迎建 王寿成* 徐进辉 陈 韬
(解放军信息工程大学 郑州 450001)

摘 要：可重构密码流体系结构是一种面向密码运算的新型体系结构，但存在着超长指令字(VLIW)代码稀疏和Kernel 体积过大的问题。该文以可重构密码流处理架构 S-RCCPA 为研究平台，通过大量密码算法在 S-RCCPA 架构上的适配分析，提出了 VLIW 可重构技术，并设计了 Kernel 级指令集、VLIW 可重构算法及指令可重构单元。实验证明，该技术能够有效提高 VLIW 的指令密度，同时降低了 VLIW 的指令宽度，使得整个 Kernel 体积减小了约 33.3%，并将微码存储器的容量由 96 kB 降为 64 kB，有效降低芯片整体面积和系统功耗。

关键词：密码流处理器；Kernel 级指令；超长指令字；可重构；指令密度
中图分类号： TP302.2 **文献标识码**：A **文章编号**：1009-5896(2017)01-0206-07
DOI: 10.11999/JEIT160213

Research of Reconfigurable Very Large Instruction Word on Cipher Stream Architecture

YAN Yingjian WANG Shoucheng XU Jinhui CHEN Tao
(*PLA Information Engineering University*, *Zhengzhou* 450001, *China*)

Abstract: Reconfigurable cipher stream architecture is a newly proposed architecture for cipher processing, but poor Very Large Instruction Word (VLIW) code density and huge Kernel level code cubage are always serious problems on this architecture. Through analyzing the characteristics of a series of cryptographic algorithms on Stream based Reconfiguable Clustered block Cipher Processing Array (S-RCCPA) architecture, a reconfigurable VLIW dynamically technology is proposed, and the corresponding Kernel level instruction set and hardware circuit structure are designed. The experiments demonstrate that this technology can reduce VLIW width, thus improve the instruction density of VLIW effectively. Meanwhile, it can reduce about 33% of the Kernel volume, and depress the microcode store capacity from 96 kB to 64 kB. Thus it can also reduce the whole area and power consumption of chip respectively.

Key words: Cipher stream processor; Kernel level instruction; Very Large Instruction Word (VLIW); Dynamic reconfiguration; Instruction density

1 引言

随着人们对网络安全与信息安全问题的日益重视，为满足密码处理的高性能和灵活性的要求，可重构密码流处理器逐步成为了一个重要的研究方向。可重构密码流处理架构 S-RCCPA[1]融合了流体系结构[2–4]的高性能优势和可重构架构[5–8]的灵活性特征，具有密码运算性能高，算法适应性好，可扩展能力强的特点。借鉴流处理器的架构，S-RCCPA 采用了两级指令：流级指令和 Kernel 级指令。其中 Kernel 级指令是一种微指令，包括由编译器根据密码算法提前编译好的超长指令字(Very Large Instruction Word, VLIW)和配置指令，其位宽达到数百位。VLIW 分为若干指令分域，分别对应于各功能单元，而功能单元分域存在着相关性，为解决各功能单元指令分域相关性的冲突，编译器通常需要加入空操作来进行调度。由于受编译器能力与程序固有指令级并行度的限制，VLIW 往往存在大量的空操作，使得 Kernel 体积巨大，造成了处理器性能和功耗的浪费。

为减少 VLIW 指令槽浪费和 Kernel 体积，国内外进行了许多研究。文献[9]提出一种 VLIW 分域压缩技术，剔除各个子域中的空操作，并通过分布式指令存储器进行解压缩执行，但其压缩和解压效率较低。文献[10]提出一种基于指令模板的压缩方法，将模板域添加到指令中并重组编码，剔除其中的空

收稿日期：2016-03-07；改回日期：2016-07-22；网络出版：2016-10-09
*通信作者：王寿成 jeremy_419@163.com
基金项目：国家 863 计划项目(2009AA012201)，国家自然科学基金(61302107)
Foundation Items: The National 863 Project of China (2009AA012201), The National Natural Science Foundation of China (61302107)

操作，但其解码逻辑较为复杂，硬件开销较大。文献[11]在每条指令段后加入单比特标识位，标识此条指令段是否结束，通过压缩 VLIW 中的空操作指令段，该策略能够减少约 23%的指令宽度并提高工作频率，也能有效减少程序代码量。此外文献[12–14]也提出了一些压缩技术，但仅靠上述技术仍难以有效解决可重构密码流处理器 S-RCCPA 所面临的 VLIW 指令稀疏和 Kernel 代码体积过大的问题。

本文在可重构密码流处理器架构 S-RCCPA 上进行了大量的算法适配分析，发现其空操作分布与算法对可重构密码处理单元(Reconfigurable Cipher processing Unit, RCU)的使用情况有关。针对其分布规律，本文提出了 VLIW 可重构技术，通过改进其指令集格式并使 VLIW 可重构，有效解决了空操作比例较大的问题。实验证明，该技术有效降低了 VLIW 的宽度，减小指令存储/发射带宽，同时有效地降低了 Kernel 体积，减小了微码存储器的容量，从而有效减小 S-RCCPA 的面积需求和资源消耗。

2　可重构密码流处理指令特征分析

S-RCCPA 是一种基于流处理器架构的可重构分簇式分组密码处理阵列结构，能够实现任务级并行、数据级并行和指令级并行，具有很高的密码处理性能和灵活性。S-RCCPA 的 Kernel 级指令格式如图 1 所示，包括 VLIW 和配置指令。VLIW 的功能是 RCU 进行密码操作，其分为若干指令分域，分别对应于各功能单元；配置指令存储在配置寄存器(Configuration Register Files, CRF)中，负责对 RCU 进行配置，使其可灵活参与不同密码算法。

VLIW 的格式如图 1(a)所示，长度为 384 bit，分为 14 个域。前 4 个域分别为保留域(Res)、特征域(OP)、微控制器域(MC)和 IO 端口域，特征域标志着指令类型，微控制器域和 IO 端口域分别实现指令发射和数据交互的功能。后 10 个域分别对应于 10 个 RCU，每个分域的宽度为 16 bit 或 32 bit。经过对 30 余种分组密码算法在 S-RCCPA 上的映射分析发现，VLIW 的前 4 个指令分域在每种算法都需要使用，而后 10 个分域则只需部分使用。分析发现，S-RCCPA 架构在进行上述分组密码运算时只使用部分 RCU，也就是说 S-RCCPA 架构在进行一种密码算法时，除需要的 RCU 外其余 RCU 所对应的指令分域在整个算法运算期间一直是空指令，这就造成了指令槽和微码存储器容量的极大浪费。

本文选择了 6 个具有代表意义的分组密码算法 DES, AES, IDEA, RC6, Twofish 和 PP2[15]进行分析和对比。表 1 列出了上述算法的 RCU 使用情况，从表中可以看出每个算法使用 RCU 的数量不同，但都不超过 5 个。以 DES 和 AES 为例说明，S-RCCPA 架构运行 DES 算法只用到 S 盒、比特置换和逻辑运算单元 3 个 RCU，也就是说最大非空操作比例为 57.29%。S-RCCPA 运行 AES 算法时使用的 RCU 有 S 盒、比特置换、有限域乘法单元和逻辑运算单元，最大非空操作比例为 61.46%。在实际算法应用中 VLIW 的非空操作比例远远低于上述最大非空操作比例。图 2 统计了上述分组密码算法在 S-RCCPA 以 ECB 和 CBC 两种模式加密时的非空操作比例，

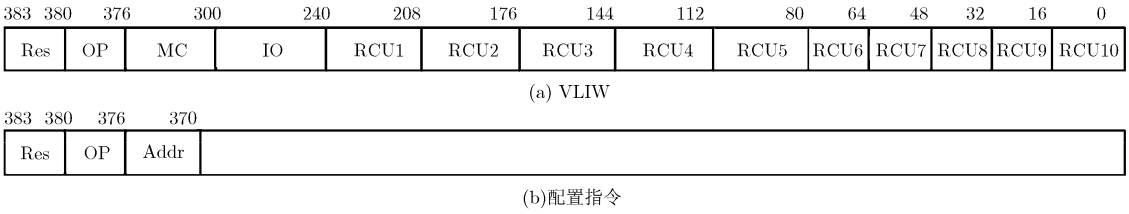


图 1　S-RCCPA 架构的 Kernel 级指令

表 1 密码算法使用到的可重构密码流处理单元 RCU

算法	S 盒	比特置换	移位	有限域乘法	逻辑运算	模加/减	模乘	模乘逆
DES	√	√			√			
AES	√	√		√	√			
IDEA			√		√	√	√	√
RC6			√		√	√	√	
Twofish	√		√	√	√		√	
PP2	√	√			√	√		

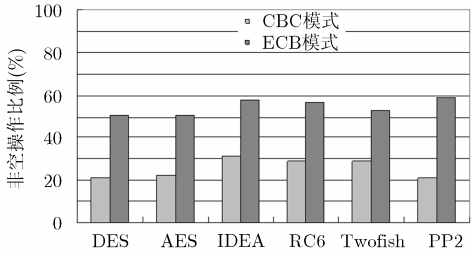


图 2 S-RCCPA 中 VLIW 非空操作统计

可以看出这个比例均值不超过 40%。从图 2 可以看出 S-RCCPA 架构在进行密码运算时 VLIW 代码稀疏和 Kernel 体积过大的问题十分严重。

3 VLIW 可重构技术

3.1 VLIW 可重构思想

VLIW 可重构技术将 VLIW 中未使用到的 RCU 分域对应的成列空操作分域剔除，缩短 VLIW 的指令宽度并减少 Kernel 的体积。其主要思想如图 3 所示，图 3(a)是原始的 VLIW 指令集示意图，其中白色方格代表无效的空操作分域，因未使用到相应 RCU 导致成列空操作的出现使得 VLIW 指令宽度和整个 Kernel 代码体积增加；图 3(b)是支持可重构的超长指令字(Reconfigurable Very Large Instruction Word, RVLIW)的指令集，通过该技术将成列的空指令分域剔除，使指令分域与 RCU 之间不再固定对应，有效降低了 RVLIW 指令宽度与 Kernel 代码体积。为还原指令分域与 RCU 之间的对应关系，RVLIW 指令集还增加了一条配置信息指令，如图 3(b)网格状方格示，此条信息指令用于将 RVLIW 指令分域中的相关指令分派到相应的 RCU 中。

VLIW 可重构技术包括对 VLIW 的可重构生成和对重构后的 RVLIW 进行重构分派两个过程，其主要原理是根据密码算法对 RCU 的需求情况，将原始 VLIW 指令集进行可重构设计，生成只包含使用到的 RCU 对应指令分域的 RVLIW 以及相应的配置信息，并将此指令集发送到微码存储器进行指令发射操作；RVLIW 从微码存储器中逐条发射，在分派到 RCU 之前，根据配置信息对 RVLIW 进行二次重构并恢复指令分域与 RCU 原有的对应关系，使得每

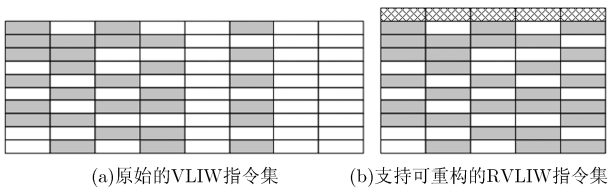


图 3 VLIW可重构示意图

个指令分域分派到相应 RCU，完成相应的密码操作。如图 4 所示，密码算法实际使用到的功能单元只有 RCU1, RCU3 和 RCU4, RCU2 和 RCU5 所对应的 VLIW 分域一直为空操作。传统的 VLIW 分派原理如图 4(a)所示，需要对每个 RCU 对应的指令分域存储、发射，并将相应指令分域分派给每个 RCU，大量的空操作使得 VLIW 过长。如图 4(b)所示，经过对原始 VLIW 的可重构，生成 RVLIW 及对应的指令分派信息，RVLIW 中的每个指令分域不再与 RCU 一一对应。将 RVLIW 从微码存储器发射之后，根据指令分派信息对 RVLIW 进行重构，将指令字分域分派到对应的 RCU 中，并完成相应的密码操作，对于不参与运算的 RCU，则不再存储、发射和分派对应的指令字分域，这样能够有效降低 VLIW 指令宽度和 Kernel 体积。

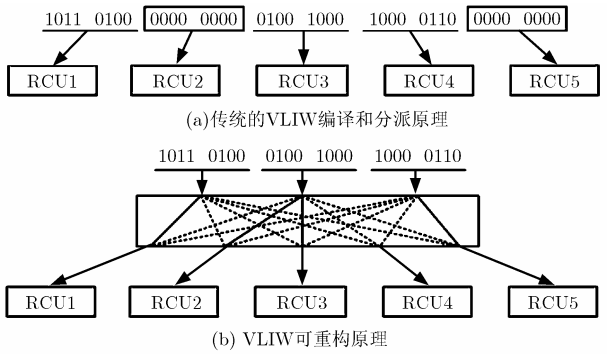


图 4 VLIW可重构原理示意图

3.2 支持可重构的 Kernel 级指令

根据 VLIW 可重构思想，本文为 S-RCCPA 架构设计了支持可重构的 Kernel 级指令，采用的是 RISC(精简指令集)的设计思想，每条指令功能较为简单。Kernel 级指令包括执行操作的 RVLIW 和配置指令。其指令格式如图 5 所示。

RVLIW 的格式如图 5(a)所示，长度为 256 bit，分为 11 个指令分域。前 4 个分域分别为保留域(Res)、特征域、微控制器域(MC)和 IO 端口域，这 4 个指令分域及功能与 VLIW 中的相应分域保持相同。后 7 个分域不再与 RCU 有对应关系，而是 7 个指令槽，每个指令槽内的指令不再固定，由 VLIW 可重构算法根据密码算法对 RCU 的使用情况进行分配调度，将使用到的 RCU 对应的指令分域按顺序分配到 7 个指令槽中。配置指令如图 5(b)，长度为 256 bit，分为 4 个指令分域，分别为保留域(Res)、特征域、地址域和配置信息域。这 4 个指令分域与图 1(b)的区别在于配置信息域的宽度缩短，此时的配置信息包括还原 RVLIW 的配置信息和对 RCU 的

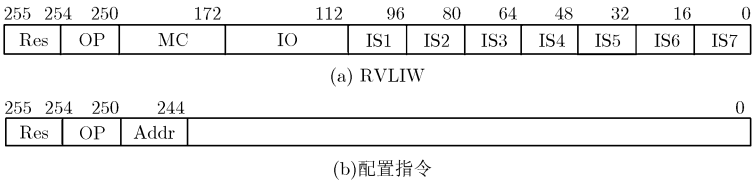


图 5 支持可重构的 Kernel 级指令

配置信息。支持可重构的 Kernel 级指令通过重构指令槽与指令分域的对应关系缩短了 RVLIW 的指令宽度，有效降低了 Kernel 的体积。

3.3 VLIW 可重构算法

密码算法程序是基于密码专用指令的汇编程序。通过编译器，将算法程序映射到 S-RCCPA 上执行，经过 VLIW 调度、寄存器分配和一系列优化后生成 VLIW 指令集。在编译时根据专用指令使用情况自动生成对应 RCU 的使用信息 RCU_use[i]，如若使用到 SBOX8T8 指令，则将 S 盒对应的使用信息 RCU_use[i]置为 1。在完成对算法程序的编译后，根据 RCU_use[i]对编译好的 VLIW 指令集进行逐条可重构生成 RVLIW 及相应的配置信息。由于分组密码算法大多是迭代型的，其指令集中指令条数通常较少，如 AES 的指令集有 15 条指令，DES 的指令集有 10 条指令。故 VLIW 可重构算法的时间消耗约为数十单位时间，相对应于繁琐的编译过程，此时间消耗并不是很大，对编译效率的影响很低。

VLIW 可重构的伪代码算法如表 2 所示。输入为编译后生成的原始 Kernel 及密码算法对 RCU 的使用信息 RCU_use[i]，算法对每条 VLIW 逐个检测，根据使用信息 RCU_use[i]来决定是否保留 VLIW 中相关的指令分域，若密码算法使用到 RCUi，则将其对应的指令分域按序写入 RVLIW 中的指令槽中。由于前 5 个 RCU 分域的指令位宽为 32 bit，若使用到此 5 个 RCU，则相应的指令分域信息占用 RVLIW 中的两个指令槽；而后 5 个 RCU 分域的指令位宽为 16 bit，若使用到此 5 个 RCU，则相应的指令分域信息只占用一个指令槽。此外还需要生成指令分域与指令槽对应关系的配置信息，由于密码算法在运算过程中对 RCU 的使用情况固定，故只需要生成一条配置信息。配置信息的生成算法与 VLIW 的重构算法类似，前 5 个 RCU 和后 5 个 RCU 的配置信息分别生成。经过 VLIW 可重构算法，将原始的 VLIW 和 RCU_use[i]转化成 RVLIW 和配置信息 Config，其中 RVLIW 主要是将 VLIW 中 RCU 对应的指令分域进行可重构设计，只保留 RCU_use[i]为 1 时 RCUi 对应的指令分域，对于 RCU_use[i]为 0 时 RCUi 对应的指令分域则不再保留。

3.4 VLIW 可重构硬件实现

VLIW 可重构算法生成的 RVLIW 改变了指令分域与 RCU 之间的位置对应关系，而 Kernel 代码在加载、存储和发射时都以 RVLIW 的形式进行，为使指令信息能够发送到相应的 RCU，本文设计了 VLIW 可重构的硬件结构，如图 6(a)所示。当微控制器执行 Kernel 时，在执行单元的控制下，将 Kernel 级指令逐条发送到指令分析单元中。指令分析单元通过识别指令中的特征域，将配置指令按地址发射到配置寄存器 CRF 相应的地址空间；将 RVLIW 中的 MC 域发射到微控制器执行单元，将 IO 域和 7 个指令槽发射到指令可重构单元中。

实现 VLIW 可重构的关键部件是指令可重构单元。指令可重构单元的结构如图 6(b)所示，其主要功能是完成 RVLIW 的重构分派，保证各指令槽的指令信息发射到对应的 RCU 中。指令可重构单元只接收部分 RVLIW，如图 6(b)所示，接收到的 RVLIW 共 174 bit，包括 IO 单元分域和 7 个指令槽。其中 IO 单元分域不需要进行指令重构分配，而是直接发送给 IO 单元；指令可重构单元根据配置信息对其硬件结构进行配置，固定其数据选择路径，使指令槽的指令信息能够分配到相应的 RCU 中，RVLIW 经过重构后的指令集格式与图 1(a)所示 VLIW 右部分相同。指令可重构单元将 7 个指令槽的指令信息分配到对应的 RCU 中，并进行相应的密码操作；而未接收到指令信息的 RCU 则不进行任何操作，有效降低了系统功耗。

4 性能评测

4.1 资源消耗分析

本文对原始 S-RCCPA 架构和支持 VLIW 可重构的 S-RCCPA 架构分别进行了资源消耗分析。相比于原始架构，本文提出的支持 VLIW 可重构架构对微码存储器容量的要求降低，但增加了一个新的功能部件 —— 指令可重构单元。本文采用 55 nm CMOS 工艺，利用综合工具基于标准单元库逻辑综合获取了其硬件资源代价信息，其中 RAM 由

表 2 VLIW可重构算法

```
定义：i为RCU的序号(范围是[1,10])；j为指令槽的序号(范围是[1,7])
      k为VLIW在Kernel的序号；length为Kernel中VLIW的总条数
      RCU_base为RCU1分域首地址；IS_base为IS1指令槽首地址。
输入：  VLIW: Kernel代码编译的原始超长指令字；
      RCU_use[i]：算法对RCUi的使用情况，使用为1，否则为0。
输出：  RVLIW：经过指令可重构的超长指令字；
      Config: VLIW可重构后相应的配置信息。
VLIWReconfig(RCU_use, VLIW, RVLIW){ //指令可重构
for(int i=1; i<=10; i++){
    j=1;                          //先从指令槽1的指令重构开始
    if(i<=5){                   //RCU1-5的指令分域宽度为32 bit
        if(RCU_use[i]) {
          RVLIW(IS_base-16(j-1), IS_base+1-16(j+1))
          =VLIW(RCU_base-32(i-1), RCU_base+1-32i);
          //若用到此功能单元，将原始VLIW相应的指令分域分
          配到RVLIW相应的指令槽
            j=j+2;
        }
    }
    else{                       //RCU6~10的指令分域宽度为16 bit
        if(RCU_use[i])  {
          RVLIW(IS_base-16(j-1), IS_base+1-16j)
          =VLIW(RCU_base-16(i+4), RCU_base+1-16(i+5))
          j=j+1;
        }
    }
  }
KernelReconfig(VLIW[length], RVLIW[length]){
   //对整个Kernel代码进行可重构
for(int k=0; k<length; k++)
      VLIWReconfig(RCU_use, VLIW[k], RVLIW[k]);
}
ConfigGEN(RCU_use, Config){       //配置信息生成
    j=1;                          //首先生成指令槽的配置信息
    if(i<=5){                   //RCU1~5的指令分域宽度为32 bit
        if(RCU_use[i])  {
          Config[2i-1]=j; //此时需生成两个指令槽配置信息
          Config[2i]=j+1;
          j=j+2;
        }
        else{
          Config[2i-1]=0;//若RCU未用到，则对应的配置信息为0
          Config[2i]=0;
        }
    }
    else{                       //RCU6~10的指令分域宽度为16 bit
        if(RCU_use[i])  {
          Config[i+5]=j;//此时只需生成一个指令槽的配置信息
            j=j+1;
          }
        else{
            Config[i+5]=0;
        }
    }
}
```

Memory Compiler 定制生成。原始 S-RCCPA 架构和支持 VLIW 可重构的 S-RCCPA 架构的硬件资源消耗对比如表 3 所示。

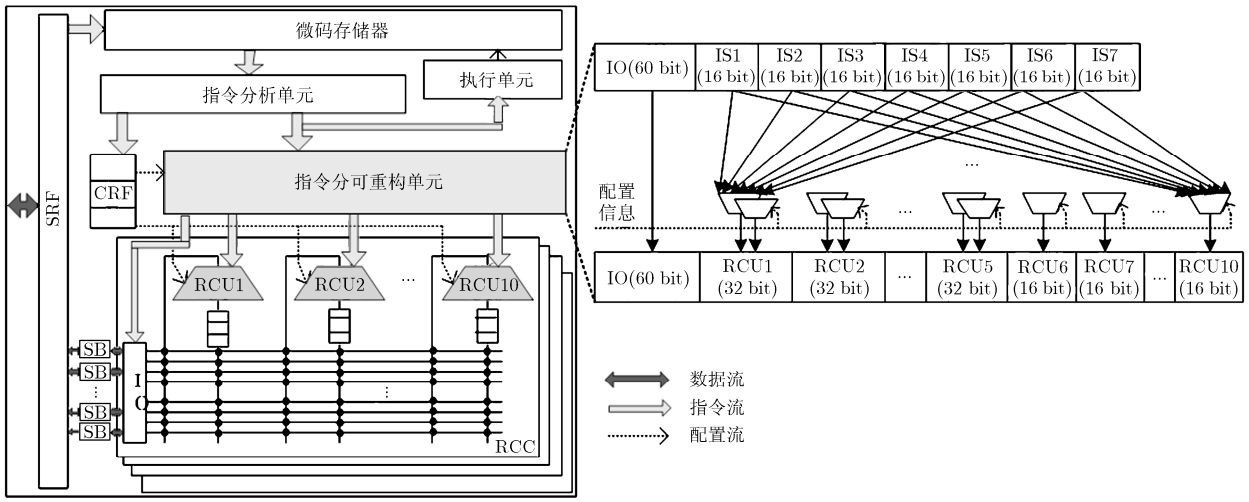
表 3 中，架构 1 代表原始 S-RCCPA 架构，架构 2 代表支持 VLIW 可重构的 S-RCCPA 架构。指令可重构过程在指令分析完成后进行，此阶段位于流水线译码栈。原始架构译码栈时间延迟为 1.07 ns，执行栈是关键路径，其延迟为 1.25 ns。新架构中，译码栈增加了指令可重构单元，其时间延迟必定增加，经综合得其延迟时间为 1.21 ns，增加了 0.14 ns。由于译码栈时间延迟并没有超过执行栈而成为关键路径，即指令可重构单元的增加并不影响处理器的工作频率。此外，微码存储器的容量由 96 kB 降为 64 kB，面积也相应降低了 69609.6 μm^2。而增加的指令可重构单元使用的逻辑门数仅为 2989.9，面积为 4305.6 μm^2。相比原始架构，本文提出的支持 VLIW 可重构的架构有效降低了处理器整体面积和系统功耗。

4.2 性能对比分析

VLIW 可重构技术是一种减少代码体积的新型技术，其实现效果与指令压缩技术类似。为了评估 VLIW 可重构的性能和效率，本文选择了相关指令压缩技术进行比较，如表 4 所示，主要从面积消耗和指令压缩比两个方面来进行比较。其中指令压缩比是压缩后 VLIW 代码体积与压缩前的比值，其值越小，说明压缩效率越高。

文献[9]针对流处理器 MASA 进行 VLIW 分域压缩，其测试程序集包括 RS 解码算法, MPEG2, H.264 算法等流程序，平均压缩比达到 61.44%，指令存储器面积减小 37.24%。文献[10]是针对传输触发型架构 TTA 进行可变长指令压缩，其测试程序集包括 AES 算法，JPEG 算法，motion 算法等，其平均压缩比达到 63%，但面积却增加了 3 倍多。文献[11]针对高速嵌入式 DSP 进行多级指令压缩，其测试程序集包括 MP3, MPEG, amr 等程序，其平均压缩比为 77%，文中未提及其资源消耗情况。文献[12]针对粗粒度可重构架构 CGRA 进行自适应 VLIW 压缩，其测试程序为 H.264 算法，其压缩比为 52%，未提及资源消耗情况。本文提出的 VLIW 可重构技术的压缩比对所有算法 Kernel 的压缩比均为 66.67%，其指令存储器面积减小 31.11%。相比之下，本文提出的技术对于密码算法 Kernel 体积的减小具有普遍高效性，并且其资源消耗也较小。

指令压缩方法通常是针对某类特定架构和某些特殊程序而提出的，由于架构和程序自身特征差异较大，与不同压缩方法直接对比仅能说明 VLIW 可



(a)总体结构图　　(b)指令可重构单元结构图

图 6　VLIW 可重构的结构图

表 3 综合结果对比表

架构	时间延迟		微码存储器		指令可重构单元	
	译码栈(ns)	执行栈(ns)	容量(kB)	面积(μm^2)	逻辑门数(门)	面积(μm^2)
1	1.07	1.25	96	193813.8	/	/
2	1.21	1.25	64	129209.2	2989.9	4305.6

表 4 压缩性能比较

	文献[9]	文献[10]	文献[11]	文献[12]	本文方法
面积消耗(%)	−37.24	+331	/	/	−31.11
压缩比(%)	61.44	63	77	52	66.67

重构技术具有一定的指令压缩能力，并不能直观说明其提高指令密度的效率。因此通过比较两种指令集结构的非空操作比例来评估 VLIW 可重构技术提高指令密度的能力。本文在两种处理器架构上适配了大量的密码算法，并分别统计了两种架构进行 ECB 和 CBC 模式的密码运算时的非空操作比例。图 7 列举了 DES, AES, IDEA, RC6, Twofish 和 PP2 等 6 种算法在上述 4 种情况下的非空操作比例。

从图 7 可以看出，支持 VLIW 可重构的 S-RCCPA 架构中算法非空操作比例比原始 S-RCCPA 架构提高了 15%∼30%，这是主要是由于 VLIW 可重构技术直接剔除了不需要的指令分域，缩短了 VLIW 的指令宽度，使得算法 Kernel 整体的指令利用率得到了巨大的提升。此外，在进行 ECB 模式加密时，非空操作比例较 CBC 模式时有了明显提升。这是因为 S-RCCPA 架构进行 CBC 模式加密时不仅能够挖掘数据级并行度，也能够有效开发指令级并行度，尽可能高效地使用 VLIW 资源。

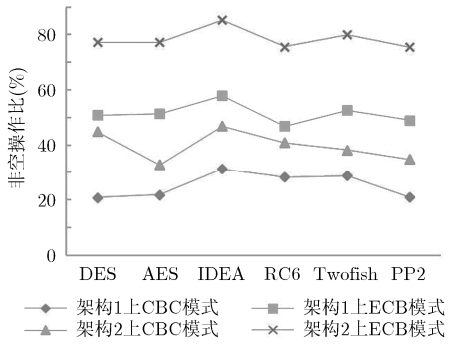


图 7 非空操作比例对比图

5　结束语

本文通过对 S-RCCPA 架构 Kernel 级指令和密码流处理特征进行分析，提出了 VLIW 可重构技术，设计了相应的 Kernel 级指令集，同时还设计和实现了硬件电路结构。实验证明，该技术通过较小的硬件资源消耗实现了较大的指令密度提升，同时在代码执行过程中无其他操作，代码压缩和代码执行过

程中都具有很高的效率。经过对大量算法的适配表明，VLIW 可重构技术通过减少 Kernel 级指令宽度有效提升了 VLIW 中的非空操作比例，同时也压缩了 Kernel 代码体积，使得微码存储器的容量减小了 33.33%，从而有效降低芯片的整体面积和系统功耗。但支持 VLIW 可重构的 Kernel 级指令集仍存在指令槽为空的情况，下一步可结合 VLIW 编译优化和指令压缩等方法进一步提高 VLIW 的指令密度。

参 考 文 献

[1] 陈韬, 罗兴国, 李校南, 等. 一种基于流处理框架的可重构分簇式分组密码处理结构模型[J]. 电子与信息学报, 2014, 36(12): 3027–3034. doi: 10.3724/SP.J.1146.2014.00023.
CHEN Tao, LUO Xingguo, LI Xiaonan, *et al.* An architecture of stream based reconfigurable clustered block cipher processing array[J]. *Journal of Electronics & Information Technology*, 2014, 36(12): 3027–3034. doi: 10.3724/SP.J.1146.2014.00023.

[2] LEE K Y, KYUNG G, PARK T R, *et al.* A design of a GP-GPU based stream processor for an image processing[C]. IEEE International Conference on Telecommunications & Signal Processing, Prague, Czech Republic, 2015: 535–539.

[3] CHENG Tengyuan, CHEN Tsunghuang, CHEN J C, *et al.* Coarse-grained reconfigurable image stream processor architecture for high-definition cameras and camcorders[C]. IEEE International SoC Design Conference, Incheon, South Korea, 2010: 95–98.

[4] KRIMER E, PAWLOWSKI R, EREZ M, *et al.* Synctium: A near-threshold stream processor for energy-constrained parallel applications[J]. *IEEE Computer Architecture Letters*, 2010, 9(1): 21–24. doi: 10.1109/L-CA.2010.5.

[5] 李校南, 王雪瑞, 戴紫彬, 等. 可重构分簇式分组密码处理架构[J]. 计算机应用与软件, 2014, 31(1): 315–318. doi: 10.3969/j.issn.1000-386x.2014.01.085.
LI Xiaonan, WANG Xuerui, DAI Zibin, *et al.* Reconfigurable clustered block cipher processing architecture[J]. *Computer Applications and Software*, 2014, 31(1): 315–318. doi: 10.3969/j.issn.1000-386x.2014.01.085.

[6] HUANG Wei, HAN Jun, WANG Shuai, *et al.* A low-complexity heterogeneous multi-core platform for security SoC[C]. IEEE Asian Solid-State Circuits Conference, Beijing, China, 2010: 126–129.

[7] WANG Bo and LIU Leibo. A flexible and energy-efficient reconfigurable architecture for symmetric cipher processing [C]. IEEE International Symposium on Circuits and Systems, Lisbon, Portugal, 2015: 1182–1185.

[8] SAYILAR G and CHIOU D. Cryptoraptor: high throughput reconfigurable cryptographic processor[C]. IEEE/ACM International Conference on Computer-Aided Design, San Jose, California, USA, 2014: 155–161.

[9] 管茂林, 何义, 杨乾明, 等. 流体系结构指令存储器优化设计研究[J]. 电子学报, 2012, 40(7): 1379–1385. doi: 10.3969/j.issn.0372-2112.2012.07.016.
GUAN Maolin, HE Yi, YANG Qianming, *et al.* Optimized design research of instruction memory for stream architecture[J]. *Acta Electronica Sinica*, 2012, 40(7): 1379–1385. doi: 10.3969/j.issn.0372-2112.2012.07.016.

[10] HELKALA J, VIITANEN T, KULTALA H, *et al.* Variable length instruction compression on transport triggered architectures[C]. IEEE International Conference on Embedded Computer Systems: Architectures, Modeling & Simulation, Samos Island, Greece, 2014: 149–155.

[11] JIN T, AHN M, YOO D, *et al.* NOP compression scheme for high speed DSPs based on VLIW architecture[C]. IEEE International Conference on Consumer Electronics, Las Vegas, Nevada, USA, 2014: 304–305.

[12] HE Yi, GUAN Maolin, ZHANG Chunyuan, *et al.* Fully distributed on-chip intruction memory design for stream architecture based on field-divided VLIW compression[C]. IEEE 14th International Conference on High Performance Computing and Communications, Liverpool, United Kingdom, 2012: 25–32.

[13] CHUNG Mookyoung, KIM Junkyoung, CHO Yeongon, *et al.* Adaptive compression for instruction code of coarse grained reconfigurable architectures[C]. IEEE International Conference on Field-programmable Technology, Kyoto, Japan, 2013: 394–397.

[14] 李勇, 王志英, 赵学秘, 等. 配置流驱动计算体系结构指导下的 ASIP 设计[J]. 计算机研究与发展, 2007, 44(4): 714–721.
LI Yong, WANG Zhiying, ZHAO Xuemi, *et al.* Design of application specific instruction-set processors directed by configuration stream driven computing architecture[J]. *Journal of Computer Research and Development*, 2007, 44(4): 714–721.

[15] BUCHOLC K, CHMIEL K, GROCHOLEWSKA C A, *et al.* PP-2 block cipher[C]. International Conference on Emerging Security Information, Systems and Technologies, Barcelona, Spain, 2013: 162–168.

严迎建： 男，1973 年生，博士，教授，研究方向为安全专用芯片设计技术.

王寿成： 男，1992 年生，硕士生，研究方向为可重构计算、体系结构设计技术.

徐进辉： 男，1978 年生，博士，讲师，研究方向为可重构计算、体系结构.

陈 韬： 男，1979 年生，博士，副教授，研究方向为通信与信息安全专用集成电路设计技术.