基于程序特征分析的流处理器 VLIW 压缩技术与解压实现*

管茂林[1],何义[2],杨乾明[1],张春元[1]
(1. 国防科技大学 计算机学院,湖南 长沙 410073; 2. 北京油料研究所,北京 102300)

摘 要:代码体积和代码稀疏是 VLIW 处理器一直存在的问题。通过对一系列典型应用在流处理器上的程序特征进行分析,提出了一种新的 VLIW 分域压缩技术,剔除各个子域中的空操作,并设计了分布式指令存储器对压缩后的代码进行解压缩执行。实验证明,该技术能够减少 MASA 流处理器中近 39% 的片外指令访存,降低约 65% 的片上指令存储器空间需求;同时使得指令存储器面积和系统面积分别减少了约 37% 和 8.9%。

关键词: 特征分析;流处理器;代码压缩;分布式指令存储器
中图分类号:TP302 **文献标志码**:A **文章编号**:1001-2486(2012)01-0138-06

Program characteristics analysis-based VLIW compression and decompression on stream processor

GUAN Maolin[1], *HE Yi*[2], *YANG Qianming*[1], *ZHANG Chunyuan*[1]
(1. College of Computer, National University of Defense Technology, Changsha 410073, China;
2. POL Research Institute of Beijing, Beijing 102300, China)

Abstract: Huge code cubage and poor code density have always been a serious problem in VLIW processors. Through analyzing the code characteristics of a series of typical applications on stream processor, a novel domain-divided VLIW code compression scheme to eliminate the nop operations in each domain, and a distributed instruction memory to decompress and execute of the codes are proposed. The experiments show that this scheme can reduce nearly 39% of the off-chip instruction memory accessing and about 65% of the on-chip instruction memory space demand. Meanwhile, it can also depress the area of on-chip instruction memory and MASA stream processor by 37% and 8.9% respectively.

Key words: characteristics analysis; stream processor; code compression; distributed instruction memory

作为开发指令级并行性(Instruction Level Parallelism,ILP)的经典结构,VLIW(Very Long Instruction Word)处理器依然在高性能微处理器领域占有重要位置。然而,VILW 处理器也面临着代码体积的严重障碍。一方面,为了获得足够多的 ILP,编译器往往采用循环展开、软件流水等措施,这会增加代码大小;另一方面,受限于程序固有的 ILP 以及编译器的能力,VLIW 常常不能被打满,存在很多的空操作。流体系结构[1]作为近年来涌现出来的新型体系结构,在图像处理、信号处理、科学计算等高性能计算领域取得了很好的应用效果,其核心级计算簇(cluster)采用典型的 VLIW 结构,利用分布式寄存器文件为功能单元提供数据,这一方面降低了功耗,提高了性能,另一方面也使得代码体积更为庞大,大量的空操作引起的访存带宽浪费、存储资源消耗和处理器能量耗散等问题将更加严重。

国内外很多学者都已经认识到代码体积带来的问题,并提出了多种解决方案。文献[2]将常见的指令序列抽象成例程,当需要执行这些代码序列时,调用例程库中相应的例程。该方法能获得一定的压缩比,但它增加了执行传输的额外开销,降低了编程的灵活性以及指令 Cache 的性能。基于硬件词典的指令压缩[3-4]是一种常见的压缩方法,面向特定领域的应用通常可取得很好的压缩效果,但是其缺点在于词典库不具备通用性,对于某些特殊程序,构建合适的词典库非常困难。另一类指令压缩方法专门针对 VLIW 进行压缩。一种方法是引入数据压缩的方法,利用压缩编码算法对 VLIW 进行重新编码[5],该方法的解码逻

* **收稿日期**: 2011-05-22
基金项目: 国家自然科学基金资助项目(61033008, 60903041);教育部博士点基金资助项目(20104307110002)
作者简介: 管茂林(1983-),男,江苏射阳人,博士生,E-mail: gfkdgmlgml@163.com;
张春元(通信作者),男,教授,博士,博士生导师,E-mail:cyzhang@nudt.edu.cn

辑比较复杂,可能带来较大的硬件开销。另一种方法根据 VLIW 指令本身的结构特点,对 VLIW 中的空指令槽进行压缩[6-7],VLIW 空操作压缩比较直观,具有合适的软硬件开销,但随着处理器中功能单元的增加,VLIW 中指令槽中的空操作比例也日益增加,这不仅增加了代码的体积,而且占用了指令存储带宽。面向应用领域定制指令集体系结构[8]和多指令集体系结构[9]等技术都能通过特定的指令结构减少程序代码量;另外,循环 Cache[10]以及热代码管理[11]等技术,都能通过存储管理优化与硬件结构改进相结合的方式,减少代码对存储空间的需求。

针对于流处理器中的 VLIW 稀疏问题,本文通过对一系列典型的流应用程序进行特征分析,提出了一种新的 VLIW 分域压缩技术,同时设计了用于解压缩的分布式指令存储器。实验证明,该技术能有效地减少流处理器的片外指令访存和片上存储空间需求,同时能有效地减少流处理器中片上指令存储器的资源消耗和面积需求。

1　流程序特征分析

本文选择 MASA(Multiple-morphs Adaptive Stream Architecture)流处理器[1]作为研究验证平台,其 VLIW 格式如图 1 所示。每条指令包含了 3 个加法单元、2 个乘法单元、1 个通信单元、1 个便笺存储器单元、1 个条件寄存器单元、1 个微控制器、8 个流缓冲单元的微操作指令。

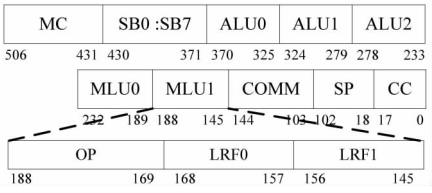

图 1　MASA VLIW 格式
Fig. 1　VLIW format of MASA

分析代码特征前首先要对流处理器的 VLIW 进行合理的子域划分。最简单的方法是每个功能单元对应一个域,这种划分方法比较直观,但是由于运算簇内采用分布式寄存器文件结构,指令执行完成后,并不负责将结果写入对应的寄存器文件,而是将其放到簇内总线上,由需要该指令结果本地寄存器文件从总线上读取数据。因此,ALU 中 ALU 操作对应的字段与写 LRF 操作对应的字段并没有联系,其空操作的比例和时机都存在差异,若将它们分于同一个域,必然会使得空操作比例的统计结果较实际情况要低。而 ALU 中 ALU 操作对应的字段与读 LRF 操作对应的字段紧密联系,互相影响。因此最精确的方法是将所有弱关联的字段全都分开。基于以上规则并考虑硬件复杂性,本文的实验中将 MASA 流处理器的 VLIW 分成了如图 2 中下方所示的 25 个子域。其中 lrf 子域专指对该功能单元的本地寄存器文件的写操作字段。

本文选择了若干典型流应用作为测试程序,如表 1 所示。图 2 统计了各个应用各子域非空操作比例的平均值。可以看出,应用中各子域的非空操作比例的均值都不超过 45%,其中计算类功能单元的非空操作比例较高,其他功能单元的非空操作比例与应用相关,但通常都低于计算类功能单元的非空操作比例。从图 2 中可以看出,流体系结构中 VLIW 代码稀疏问题非常严重。

表 1　测试程序集描述

Tab. 1　Description of benchmarks

应用	描述
RS	Reed Solomon 解码算法[12]
Susan	Mibench 中的图像处理边角检测算法[13]
LUCAS	利用 Lucas-Lehmer 测试方法检测梅生素数[14]
MPEG2	MPEG2 标准视频图像压缩算法[15]
Ygx2	属于 IAPCM 测试程序集,利用拉格朗日与欧拉算法,计算二维爆炸流体力学问题[16]
H.264	由 JVT(Joint Video Team)提出,一种新的视频编码标准中的编码算法[17]

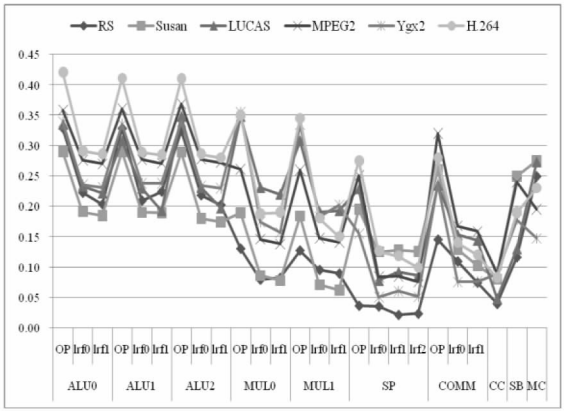

图 2　MASA 中 VLIW 非空操作统计
Fig. 2　Statistic of VLIW valid ops on MASA

2　VLIW 分域压缩技术

2.1　分域压缩思想

分域压缩技术通过将 VLIW 各个子域中的空操作剔除,减少无效指令对存储带宽和存储空间的浪费。其主要思想如图 3 所示,(a)是程序原

始的 VLIW 示意图,其中白色部分表示无效的空操作;(b)是分域压缩之后的 VLIW,它将原始指令中的空操作删除,并将所有相同子域的操作按先后顺序无缝地连续组合,形象地说,就是将图 3(a)中的白色部分删除,灰色部分“自由落体”堆积生成压缩后的 VLIW。最后将压缩后的 VLIW 进行规整化,即所有的子域都按照压缩后最长子域进行规整,长度不足的子域均在最后添补适量的空操作(即全 0 值),如图 3(b)中白色部分所示。

2.2 VLIW 分域压缩算法

分域压缩的伪代码算法如图 4 所示。输入为编译后生成的原始 VLIW,算法对每条指令的各个子域分别进行检测,判断其是否为空操作,并记录该信息。通常,在处理器的指令集设计中,将空操作的指令码设置为全零,因此,在空操作判断时,ALU、MUL 等功能单元仅需要判断其操作码,若操作码为 0,则表示该域对应一条空指令;LRF、CC 等单元则需要判断其所有位是否为零,若所有位均为零,则表示其对应一条空操作;IO 中包含 8 个流缓冲的操作,因此,需要判断 8 个流缓冲对应的操作码,若所有操作码均为 0,则表示 IO 域对应一条空操作;Microcontroller 域包含 MC 指令和控制指令两个部分,若 MC 操作码和控制指令均为零,则 MC 域对应一条空指令。

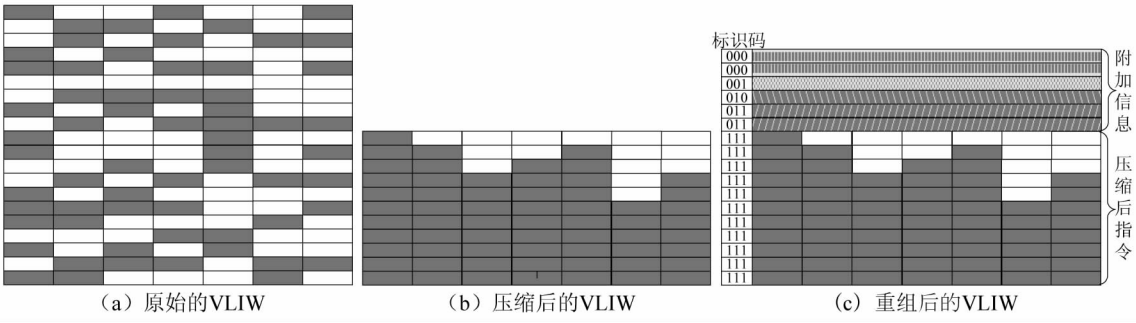

图 3 VLIW 压缩与重组

Fig. 3 VLIW compression and recombination

```
定义：L为编译生成VLIW序列长度，R为划分的VLIW子域的数目；
输入：VLIW[L][R]：核心程序经编译后生成的原始VLIW;                    输出：vliw[L][R]：分域压缩后的指令；
其他结构: nulloff[L][R]: 各个子域是否为空操作，初始值为0;            instr_use[R]:各个子域有效操作的数目，初始值为0;
          max：各个子域有效操作数目的最大值，初始值为0;              branch：跳转指令的数目；
          offsets[branch][R]：压缩后跳转指令各子域的偏移量，初始值为0;
CodeCompression(VLIW[L][R], vliw[L][R]){
  //判断各个子域是否为空并记录相关信息
  for(int i = 0; i < L; i++)
    for(int j = 0; j < R; j++)
      if(!isnull(VLIW[L][R]))
        nulloff[i][j] = 1;
  //根据nulloff，记录有效代码并计算每个子域的实指令的数目；
  for(int i = L - 1; i >= 0; i--)   //自底向上开始记录
    for(int j = 0; j < R; j++)
      if(nulloff[i][j] == 1){
        instr_use[j] = instr_use[j] + 1;
        vliw[L - instr_use[j]][j] = VLIW[i][j];
      }
  //计算各个子域有效操作数目的最大值
  for(int i = 0; i < R; i++)
    if(instr_use[i] > max)
      max = instr_use[i];
  //计算跳转指令数目及压缩后的跳转指令各个子域实际的偏移量
  for(int i = 0; i < L; i++)
    if(VLIW[i][MC]控制位 != 0){
      branch = branch + 1;
      //以向后跳转为例，offset为原始VLIW中的跳转指令偏移量
      for(int j = i; j < j + offset; j++)
        for(int k = 0; k < R; k++)
          if(nulloff[j][k] == 1)
            offsets[branch][k] = offsets[branch][k] + 1;
    }
}

  最后输出的从vliw[L-max-1]到vliw[L-1]的max
  条指令即为压缩后的指令

//判断子域代码是否为空操作
bool isnull(VLIW[i][j]){
  switch(FU_region){              //对各个子域的类型进行区分
  case ALU or MUL or COMM or SP:
    if(VLIW[i][j]操作码==0) //通用的功能单元判断操作码
      return true;
      break;
  case lrf or cc:                 //lrf和cc单元判断是否所有位为0
    if(VLIW[i][j]==0)
      return true;
      break;
  case IO:                        //流缓冲单元判断所有操作码为0
    if((IO0操作码&IO1操作码&……&IO7操作码)==0)
      return true;
      break;
  case MC:                        //MC单元无MC操作与控制指令
    if((MC操作码&MC控制位)==0))
      return true;
      break;
  }
  return false;
}
```

图 4 分域压缩算法

Fig. 4 Domain-divided compression algorithm

根据子域空操作信息,对原始的 VLIW 自底向上依次记录各子域代码的有效操作,计算各个子域的有效代码的数目及其最大值,这个最大值也就是压缩后的 VLIW 序列的长度。为了能够有效地解压缩执行,需要对控制程序执行的分支指令进行特殊处理。检测出每条分支指令,获取其分支目标指令的偏移地址,在这两条指令之间,根据前面记录的子域空操作信息,计算每条分支指令在压缩后的 VLIW 中每个子域与分支目标指令实际的偏移地址,并记录该信息。

2.3　VLIW 重组

分域压缩改变了原始 VLIW 各子域以及各条指令的操作之间的相关关系,为了硬件能够顺利地解压缩执行,本文为压缩后的指令添加了若干辅助信息,并将其与压缩后的指令一起打包生成新的指令。本文将这一过程称为 VLIW 重组。为了区分压缩后的指令与辅助信息,本文在重组后的指令之前添加了 3 位标识码,用以说明其后的指令的实际意义,如图 3(c)所示。下面结合图 4 的算法对新添加的辅助信息进行详细说明。

(1)子域有效指示,标识码为 3'b000。后续的微代码存储了原始 VLIW 的各个子域是否有效地指示信息,每一位代表一条 VLIW 的一个子域的代码是否为有效代码,对应于图 4 中的 nulloff[L][R]。

(2)指令及子域有效代码长度,标识码为 3'b001。后续的微代码存储了原始 VLIW 及其各个子域的有效代码长度信息,对应于图 4 中的 L 及 instr_use[R]。

(3)子域起始地址信息,标识码为 3'b010。后续的微代码存储了压缩后的指令在程序执行时,各个子域的起始地址信息。整个程序的起始地址由编译产生,这个地址也是有效代码长度最长的子域的起始地址,其他子域的起始地址可根据该地址以及图 4 中的 instr_use[R]计算获得。

(4)分支指令信息,标识码为 3'b011。后续的微代码存储了程序遇到分支指令时在压缩后的指令中各个子域实际的分支偏移信息,对应于图 4 中的 offsets[branch][R]。

(5)有效指令,标识码为 3'b111。后续的微代码存储实际的有效代码,对应于图 4 中的从 vliw[L-max-1]到 vliw[L-1]的 max 条指令。

3　硬件解压缩实现

3.1　分布式指令存储器

为了对压缩后的 VLIW 进行解压缩,本文为流处理器设计了分布式指令存储器。如图 5 所示,分布式指令存储器不仅将流处理器中多个 cluster 共享的集中式指令存储器分布到各个 cluster 中,而且将其根据划分好的 VLIW 子域,进一步切割成多个更小的分布式指令寄存器文件(DIRF:Distributed Instruction Register File),分别与各子域对应的功能单元直接紧密连接,为其提供所需的指令。所有 cluster 对应的 DIRF 之间有环形的指令传输通路。当 kernel 代码导入 DIRF 之后,一条 VLIW 某个子域的操作只会存在于某个 cluster 内对应的 DIRF 中,因此,若其他 cluster 需要执行该指令,则需要通过互连进行指令传输。为保证程序的正确执行,本文设计了索引寄存器文件(IDRF:Indexed Register File)和起始地址与分支偏移量寄存器(BBR:Begin and Branch Register)。IDRF 和 BBR 与所有的 cluster 连接,在程序执行时,IDRF 为功能单元提供各 VLIW 对应子域的空操作信息,BBR 提供每个核心程序起始的 VLIW 中各子域 DIRF 的地址和分支指令执行时各子域对应的分支偏移量。

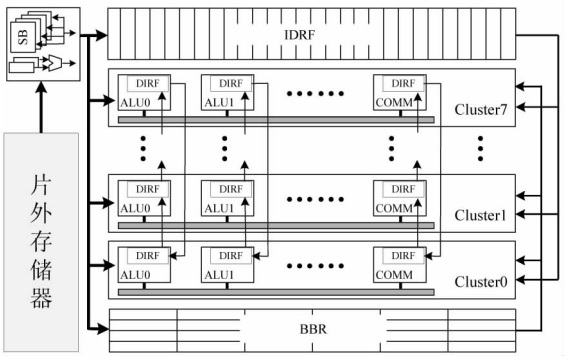

图 5　分布式指令存储器总体结构

Fig. 5　Structure of distributed instruction memory

3.2　硬件解压缩模型

kernel 执行时,首先微控制器从 BBR 中读取 kernel 在各 DIRF 中的起始地址,并将各 DIRF 的 PC 值指向对应的起始地址;然后,从 IDRF 中读取第一条 VLIW 对应的子域标识位,并将其送入对应的 DIRF 中。若某 DIRF 对应的子域标识位为 1,则将其 PC 对应的微指令发送至对应的功能单元中执行,PC 加 1;否则,将 0 号地址中的空指令送入功能单元,PC 值不变。在指令执行时,若译码为一条分支指令,则微控制器读取 BBR 中分支指令各子域对应的偏移量,若分支成功,各 DIRF 通过当前 PC 值与偏移量计算出分支的目的 PC 值,若分支不成功,则执行下一条指令。按照这种执行方式,依次对 kernel 中所有的 VLIW 进行处理,直至 kernel 结束。图 6 展示了硬件解压缩模型,根据上述的执行模式可以看出,该解压

缩模型并不是将压缩后的代码还原成原有的 VLIW,而是在硬件执行上保持了原 VLIW 所完成的功能,所有功能单元从各自的 DIRF 中获取的指令(不管是空指令还是非空指令),在逻辑上可以组装还原成未压缩的原始 VLIW。

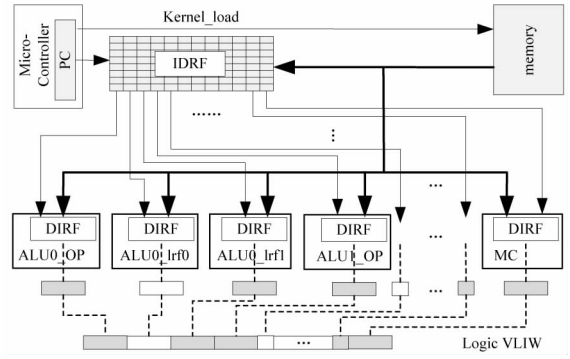

图 6 硬件解压缩模型

Fig. 6 Model of hardware decompression

4 实验与结果分析

4.1 代码压缩比评估分析

本文对表 1 中的应用程序进行了测试,其片外和片上指令码压缩比如表 2 所示。其中,片外指令码体积指的是 VLIW 重组之后的代码体积,片上指令码体积指的是存放在分布式指令存储器内的包括所有的非空操作、VLIW 子域标识码、kernel 起始地址和分支偏移信息等在内的代码体积。

从表 2 中可以看出,本文提出的分域压缩技术能够有效地减少代码体积,节省宝贵的片上指令存储器空间。片外指令码和片上指令码分别平均可获得 61.44% 与 34.58% 的压缩比。片外指令码的压缩效果较片上指令码差,主要是由于 VLIW 压缩时将"参差不齐"的各个子域按最长子域补齐而添加了若干空操作,而这些空操作并未写入片上指令存储器。分域压缩技术能获得较好的片上压缩效率,这主要是因为虽然核心循环体中计算单元的利用率较高,但其他部分功能单元利用率差异很大;另外,分布式寄存器文件结构使得功能单元对应的本地寄存器的写操作与功能单元操作解耦合,而前者的比例通常远低于后者。

表 2 代码压缩性能

Tab. 2 Performance of code compression

	RS	Susan	LUCAS	MPEG2	Ygx2	H264	平均
原始程序体积	12,864B	90,112 B	61,632 B	289,408B	338,688B	384,384B	
片外指令码体积	7,680B	56,640B	31,744B	182,976B	225,088B	257,024B	
片外压缩比	59.70%	62.86%	51.51%	63.22%	64.46%	66.87%	61.44%
片上指令码体积	4,069B	30,710B	22,237B	103,694B	113,799B	139,339B	
片上压缩比	31.63%	34.08%	36.08%	35.83%	33.60%	36.25%	34.58%

4.2 资源消耗分析

本文对采用分布式和集中式指令存储器的 MASA 流处理器分别进行了资源消耗分析。集中式指令存储器采用 RAM 实现,在分布式指令存储器中,IDRF 采用 RAM 实现,而 DIRF 和 BBR 均用寄存器文件实现。MASA 中集中式指令存储器能够存储 2K 条 VLIW。根据上一小节对应用压缩比的分析,本文将每个 DIRF 定义为 128 项,即总容量为集中式指令存储器的 50%,大于所有应用程序的片上压缩比。本文采用了 0.18μm CMOS 工艺,利用 Synopsis Design Complier 工具基于标准单元库逻辑综合获取代价信息,寄存器文件、RAM 均由 Memory Compiler 定制生成。系统的资源消耗如表 3 所示。可以看出,相对于集中式指令存储器,分布式指令存储器减少了 37% 的片上指令存储器面积,使得 MASA 流处理器的系统面积降低了 8.9%。

表 3 资源消耗评估(单位:逻辑门)

Tab. 3 Evaluation of resource overhead (unit: logical gate)

	整个系统	指令存储器
集中式指令存储器	7068824	1901514
分布式指令存储器	6438285	1193379
面积节省	8.9%	37%

5 总 结

VLIW 处理器的代码体积和代码稀疏问题一直是困扰研究人员的重要问题。在流体系结构中,大量的运算单元和分布式寄存器文件结构使得 VLIW 异常复杂,代码稀疏问题更加严重。本文通过对一系列流应用程序进行分析,量化各个子域代码的空操作比例,提出了一种新的 VLIW

分域压缩技术,并设计了分布式指令存储器对压缩后的代码进行解压缩执行。实验证明,分域压缩技术能够有效减少流处理器中的片外指令访存、降低片上指令存储器空间需求;分布式指令存储器能够有效减少片上指令存储器面积,降低系统资源消耗。

参考文献(References)

[1] 张春元,文梅,伍楠,等. 流处理器研究与设计[M]. 北京:电子工业出版社,2009.
ZHANG Chunyuan, WEN Mei, WU Nan, et al. Research and design of stream processor[M]. Beijing: Publishing House of Electronics Industry, 2009. (in Chinese)

[2] Lau J, Schoenmackers S, Sherwood T, et al. Reducing code size with echo instructions[C]//Proceedings of the 2003 International Conference on Compilers, Architectures and Synthesis for Embedded Systems, 2003: 84 - 94.

[3] Corliss L, Lewis C, Roth A. A DISE implementation of dynamic code decompression[C]//Proceedings of the 2003 ACM SIGPLAN Conference on Language, Compiler, and Tools for Embedded Systems, 2003: 232 - 243.

[4] 赖明澈,王志英,戴葵,等. 基于代码特征分析的 TTA 指令压缩技术与解压部件实现[J]. 电子学报,2008(11): 2234 - 2238
LAI Mingche, WANG Zhiying, DAI Kui, et al. Characteristics analysis-based code compression and one-cycle, decompression engine for transfer triggered architecture[J], Acta Electronica Sinica, 2008 (11): 2234 - 2238. (in Chinese)

[5] Wolfe A, Chanin A. Executing compressed programs on an embedded RISC architecture[C]//Proceedings of the 25th International Symposium on Microarchitecture, 1992: 81 - 91.

[6] Black-Schaffer D, Balfour J, Dally W J, et al. Hierarchical instruction register organization [J]. IEEE Computer Architecture Letters, 2008, 7(2): 41 - 44.

[7] Liu J, Bell B, Truong T. Analysis and characterization of Intel Itanium instruction bundles for improving VLIW processor performance[C]//First International Multi-Symposiums on Computer and Computational Sciences, 2006: 389 - 396.

[8] 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(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 (4):714 - 721. (in Chinese)

[9] Cheng A, Tyson G, Mudge T. FITS: Framework based instruction-set tuning synthesis for embedded application specific processors[C]//Proceedings of the 41st Annual Conference on Design Automation, 2004: 920 - 923.

[10] Gordon-Ross A, Cotterell S, Vahid F. Tiny instruction Caches for low power embedded systems [J]. ACM Transactions on Embedded Computing Systems, 2003, 2(4): 449 - 481.

[11] He Y, Ren J, Wen M, et al. Software managed instruction scratchpad memory optimization in stream architecture on hot code analysis of kernels [C]//Proceedings of the 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools, 2010: 823 - 830.

[12] Wen M, Zhang C, Wu N, et al. A parallel reed-solomon decoder on the imagine stream processor [C]//Second International Symposium on Parallel and Distributed Processing and Applications, 2004 (12): 28 - 33.

[13] Guthaus M R, Ringenberg J S, Ernst D, et al. MiBench: a free, commercially representative embedded benchmark suite [C]//IEEE 4th Annual Workshop on Workload Characterization, 2001: 3 - 14.

[14] GIMPS. The Math [EB/OL]. [2007-05-30]. http://www.mersenne.org/math.htm.

[15] Ahn J H, Dally W J, Khailany B, et al. Evaluating the imagine stream architecture[C]//Proceedings of the 31st Annual International Symposium on Computer Architecture, 2004: 14 - 25.

[16] 袁国兴,邵京云. 评几种高档微处理器在运算科学计算问题时的性能[EB/OL]. [2003-06-20]. 北京应用物理与计算数学研究所,http://www.ccw.com.cn.
YUAN Guoxing, SHAO jingyun. Evaluating performance of scientific applications on several high performance processors [EB/OL]. [2003-06-20]. Beijing Institute of Application Physics and Computing Mathematics, http://www.ccw.com.cn. (in Chinese)

[17] Yuri V I, Bleakley C J. Dynamic complexity scaling for real-time H.264/AVC video encoding[C]//Proceedings of the 2007 ACM International Conference on MultiMedia, 2007: 962 - 970.