文章编号:1001 - 2486(2011)04 - 0065 - 08

MGR-SAT:一种基于流模板的多粒度可重构数字信号处理器*

杨乾明,文 梅,伍 楠,苏华友,全 巍,张春元
(国防科技大学 计算机学院,湖南 长沙 410073)

摘 要:面对需求各异的数字信号处理应用,当前主流的通用处理器、DSP、ASIC 和 FPGA 不能同时满足各应用在成本、功耗、性能、灵活性方面的要求。针对这些问题,结合流处理技术、可重构技术和平台技术,提出一种基于流模板的多粒度可重构数字信号处理器 MGR-SAT。MGR-SAT 利用流处理技术来解耦合数据运算与访存,提供多种粒度的大规模数据并行,同时利用可重构技术对关键算法进行加速处理,而平台技术则为系统提供快速的重构能力。实验结果显示,MGR-SAT 在数字信号处理领域具有极大潜力。

关键词:流处理;模板;可重构;多粒度
中图分类号:TP302 **文献标识码**:A

MGR-SAT: A Multi-granularity Reconfigurable DSP Based on Stream Architecture Template

YANG Qian-ming, WEN Mei, WU Nan, SU Hua-you, QUAN Wei, ZHANG Chun-yuan
(College of Computer, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: Facing the varying demand for digital signal processing applications, the existing solution including general-purpose processor, DSP, ASIC and FPGA can not meet the requirements of cost, power consumption, performance and flexibility. To solve these problems, this study combined the technologies of stream processing, and reconfigurable platform, and proposed A Multi-Granularity Reconfigurable DSP based on Stream Architecture Template (MGR-SAT). MGR-SAT uses stream processing techniques to decouple the data on operations and memory access, offering a large-scale data parallel. Meanwhile it uses the reconfigurable technology to accelerate the key algorithms and uses the platform technology to provide fast reconfiguration as well. The experimental results show that MGR-SAT has a great potential to deliver high performance.

Key words: stream processing; template; reconfigurable; multi-granularity

随着人们需求的不断提高,多媒体、通信、网络等应用对数字信号处理提出越来越高的要求,并且呈分化趋势。有的应用对功耗要求苛刻,如以手机为代表的便携式系统;有的应用需要极高的处理器性能,例如卷积运算、FFT 运算需要就达到数十乃至数百亿次操作每秒;有的应用需要可编程以适应众多变化的标准和发展,例如视频编解码标准就有 MPEG2、MPEG4、H.264 等多种;有的应用在各种通信/存储系统中有不同的规格,例如 Reed-Solomn 纠错码在 DVD、CD、DVB 中的规格分别是 RS(208,192,8)、RS(32,28,2)、RS(204,188,8)等。

面对需求各异的种种数字信号处理应用,目前的解决办法有三个:一种是希望能用一个软件可编程的芯片满足所有的应用需求,然而,DSP 的"通用性"使得其并非对所有应用都能够实现很好的功耗效率。第二种是用实时可重构处理器,在运行时根据需要动态改变系统的电路结构,从而使系统兼具灵活、易于升级等性能。但芯片软硬件设计复杂,运行时进行重构速度慢。第三种解决办法是采用专用 IC 电路。通常有比较好的功耗效率,但是缺少灵活性、成本极高、开发周期长、风险高。可以采用完全硬件可编程的 FPGA 技术,但对处理器级设计仍需专业人员进行,而且频率、规模、功耗和成本受到很大限制,一般需要再转为 ASIC。

针对这些问题,本文提出一种基于流模板的多粒度可重构数字信号处理器(MGR-SAT: A Multi-Granularity Reconfigurable DSP based on Stream Architecture Template),以面向应用量身定做处理

* **收稿日期**:2010 - 09 - 27
基金项目:国家自然科学基金项目(60703073, 60903041);国家"863"高技术研究发展计划项目(2007AA01Z286)
作者简介:杨乾明(1984—),男,博士生。

器为目标,在流模型的统一体系结构框架下,根据应用的流化特征,针对应用需求在多个不同粒度配置硬件,满足性能、功耗效率、通用性的指标。流模型是一种计算模型。该模型将应用分解成一系列计算核心(kernel),输入数据以流(stream)的形式通过核心程序并被处理,每个核心有明确的输入流和输出流。基于流模板的多粒度可重构技术是一种平衡速度、功耗、可编程性之间矛盾的创新方法,适应 DSP 和 VLSI 的发展,有利于简化设计、降低成本、缩短周期。

目前,国际上关于流计算的研究成果有很多,涌现出 Imagine[1]、RAW[2]、VIRAM[3]、TRIPS[4] 和 GPU 等多种新型体系结构,它们面向密集计算设计、解耦合数据运算和访存、由程序员显示的管理数据移动和任务分发,能够处理大量数据流,在数字信号处理、媒体处理等领域表现出良好的性能[5]。近年来,Stanford、MIT 等大学和 TI、ADI、Motorola 等公司正在研究具有流技术的数字信号处理器解决方案[6-7]。本文的前期研究[8]也表明流体系结构在媒体处理领域正发挥着其他处理器无可比拟的优势。因此,本文选择流计算模型作为 MGR – SAT 的体系结构模板。

利用可重构技术对数字信号处理应用进行加速也是国际上一个研究热点,主要有两种可重构模式:一种是"RISC 微处理器 + 可重构逻辑单元阵列"的细粒度可重构片上系统技术,另一种是"RISC 微处理器 + 可重构处理单元阵列"的粗粒度可重构片上系统技术。前者具有开发周期短、灵活性高等优点,但资源利用率低、芯片面积大、配置速度慢,只适合于关键算法或步骤的加速处理,比如 FFT,SAD、熵编码等。后者,具有计算性能高、低功耗、灵活性良好等优点,适合于对应用整体进行加速,代表性的研究成果有:台湾大学的 CRISP[9]、加州大学 Irvine 分校的 Morphosys[10]、PACT 的 XPP[11]。在此基础上,基于平台的片上系统设计方法也得到快速发展,成为学术界和工业界关注的焦点。典型的设计平台有 TI 公司的 OMAP[12]和 Triscend 公司的 A7[13]等,这些设计平台在支持高效率、高质量的片上系统芯片设计方面发挥了重要作用,使设计周期由原来的 2 ~ 3 年缩短到 1 年左右,而且在提高性能的同时,可有效地控制功耗。因此,本文提出将模板(模板实际上也是一种平台)技术和可重构技术相结合,在模板约束下对目标处理器进行重构,降低设计的难度和成本,加快设计进度。

1 体系结构

1.1 流模板

如图 1 所示,MGR-SAT 在单个芯片上集成了两个异构核,一个是标量处理核(scalar core),主要负责任务的调度和 IO 控制,可以处理应用中的条件判断、分支等控制型事务;另一个是流处理核(SPC:Stream Processing Core),主要负责对应用的计算部分进行加速处理,SPC 的内部可以在多个维度进行重构,整体以流处理的方式运行,因此我们称之为流模板。SPC 的主要部件有:一个粗粒度可配置阵列(CRA: Coarse-grain Reconfigurable Array),一个细粒度可配置阵列(FRA: Fine-grain Reconfigurable Array),一个微码控制器(MC: Microcode Controller),一个流寄存器文件(SRF: Stream Register File),两个地址产生器(AG: Address Generators)以及一个网络接口(NI: Network Interface)。

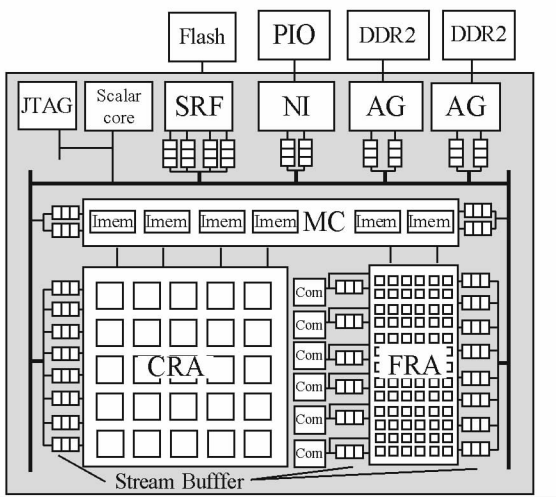

图 1 MGR-SAT 整体结构
Fig.1 The architecture of MGR-SAT

CRA 是一个由可配置 Tile(CT: Configurable Tile)组成的阵列,用来执行高级语言编写的算法模块(称为 SoftKernel),具有很好的可编程性,可以简化应用的映射并支持不同的应用。在该阵列上的数据网络和指令网络都是动态可配置的,用于支持多种并行执行模型。FRA 是一个包含细粒度可编程查找表(类似于 FPGA LUT)的资源池,它允许用户静态重构特殊功能单元(SPU: Special Purpose Unit),用于执行用户定义的指令,或者定制特殊的计算簇来执行硬件描述语言编写的算法模块(称为 HardKernel)。FRA 可以对应用中的某些不适合通用指令处理的操作(比如位合并、位重排操作等)进行定制实现,从而加速对某些关键算

法(如 FFT,熵编码)的加速处理。MC 包含大小可配置的微码存储器,它可以在 kernel 执行时将 VLIW 指令流出到 CRA 和 FRA,同时载入接下来要执行的 Kernel。为了增加指令流出带宽,MC 可以被划分为 8 个 sub – MC,每个 sub – MC 中都有专有的指令总线来控制两个指定阵列。SRF 是一个大小可配置的软件控制的片上存储,它被用来存储流,可以捕获数据局域性,尤其是 Kernel 之间的生产者 – 消费局域性。AG 用于执行流的加载/保存,可以支持多路预取地址产生模式,使得能够将存储器中分散的数据访问组织成流,它还能支持流调度策略[14]来优化存储带宽。NI 通过连续传输整个流的方式来处理多个芯片之间的通信,支持芯片的大规模扩展。流缓冲(stream buffer)用于缓冲 SPC 内部各个模块之间的数据,可以根据需要实现预取,为各模块提供高带宽数据。图 2 显示了 SPC 的一个配置实例。

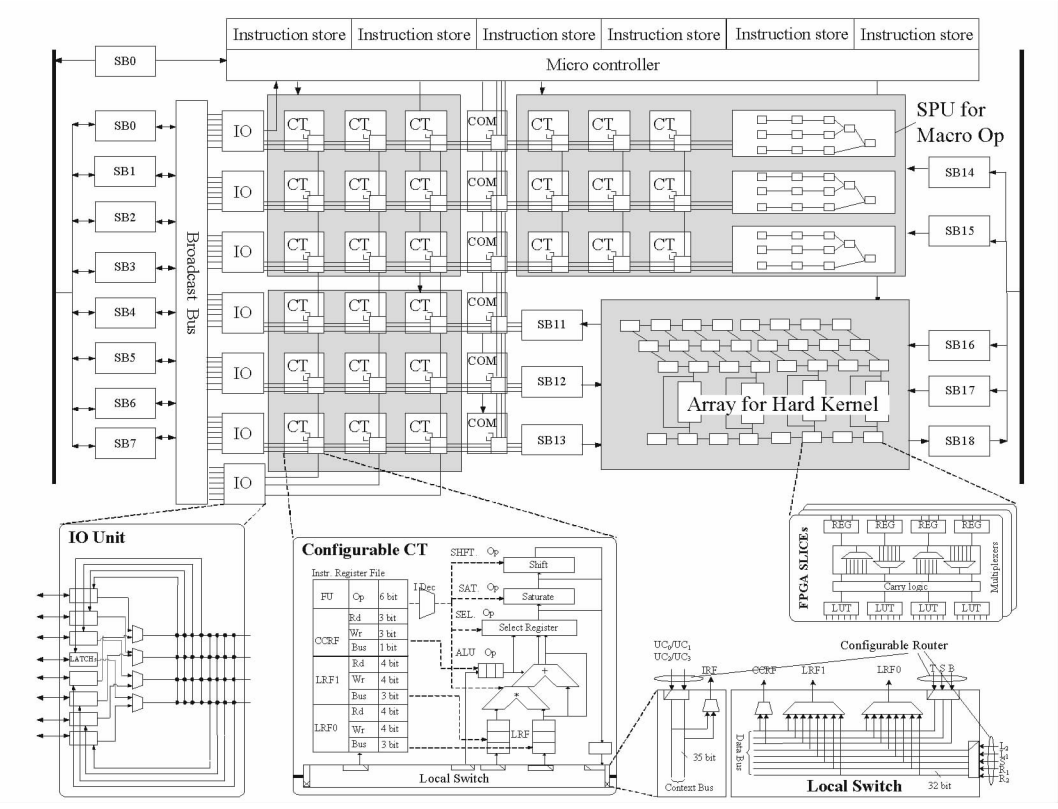

图 2　SPC 配置实例

Fig.2　Configuration example of SPC

CT 由功能单元组成,例如 ALU,乘法器,移位器以及少量类似于寄存器文件的存储单元。SRF 通过 8 个流缓冲向 CT 提供操作数。每个 Kernel 的配置文件也作为流的形式存储在 SRF 中。在 Kernel 执行之前 MC 从流中加载配置信息来配置每个 CT 的功能。配置完成后,MC 一拍一拍地为每个 CT 指令寄存器提供指令。图 2 右下角示例了 CT 的一个配置实例,35 位 VLIW 指定了 FU 的操作,LRF0、LRF1 和 CCRF 的输入,ALU 输出的值及移位方向以及结果存放的寄存器。

CT 之间的连接通路分为 32 位全双工数据总线和多个半双工指令总线,每个 CT 内部的本地选择开关用作这些总线的路由和接口。如图 2 所示,所有的 CT 都与最相邻的 CT(上、下、左、右)相连,此外,在非相邻的 CT 之间也有一定的连接,使得可以在行、列范围内的数据传输更为高效。这些连接都是可以通过本地选择开关重配置的,因为每个 CT 的多路选择器的输入都可以连接到其他 CT 上,通过配置信息可以确定多路选择器的输入。所有总线路由都在编译时静态确定。

1.2　配置模式

为了适应不同的应用需求,SPC 设计成可以配置成支持多种 kernel 执行的模式,称为态(morphs)。morph 能够按照 kernel 的设置进行静态配置,这样就可以适应不同类型的 kernel,常用的 morphs 如图 3 所示。

SIMD SoftKernel morph:该模式用于支持数据并行粒度较高的应用,比如深度萃取、图形图像压缩等多媒体应用。如图 3(a)所示,每个 CT 的本地开关被配置成连接到同一行其他的 CT 上,因

此,每个 cluster 包含了一整行的 CT。在这种模式下,kernel 在 SPC 中是分时执行的。所有的指令存储累积到能流出一条 VLIW 指令。在 kernel 的执行期间,相同的 VLIW 指令以 SIMD 的方式被广播到所有的 4 个 cluster 中。流的读/写指令通过将流元素读出或写入 SRF 来执行。

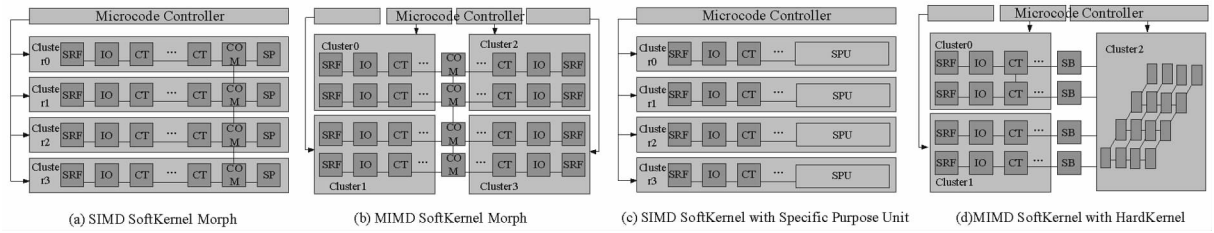

图 3 多态 SPC 可配置阵列

Fig.3 Multi-morphs of SPC

MIMD SoftKernel morph:该模式用于支持数据并行粒度较小或多任务应用,比如汽车控制、视频监控等。如图 3(b)所示,每个 CT 的本地开关被配置成和相邻的 CT 相连——上下左右 4 项相连。因此,每个 cluster 都是由 CT 矩形阵列组成的。在这种模式中,指令存储被划分为几个独立的存储,每个都能为其相应的 cluster 一拍流出一条 VLIW 指令。这意味着多个 kernel 可以空分地并行执行。一个 kernel 产生的输出记录会直接通过 COM(communication unit)单元传输到要用该结果的 kernel。也就是说,中间流可以在 cluster 之间直接传输,这样多个 kernel 可以在流水线中同步执行。如果一个 kernel 的计算量很大,那么多个 cluster 可以分配给这个 Kernel。如果生产和消费数据记录的速度不能很好地匹配,那么就需要 SRF 来存储输入/输出流以及中间流。

带 SPU 的 SoftKernel morph:该模式用于支持需要特殊指令(由 SPU 执行)进行加速的应用,例如熵编码中的位合并和移位操作等。如图 3(c)所示,FRA 可以被配置成 CRA 的特殊功能单元(SPU)。SPU 的复杂度是不同的。例如:伽罗华乘复杂度较低,而超越函数复杂度则较高。SPU 通过本地总线直接连接到相应 cluster 中其他 CT。在这种情况下,SPU 的频率和 SIMD/MIMD 方式就需要与 cluster 相对应。SPU 所执行的宏指令也被打包在 VLIW 中。

HardKernel morph:HardKernel 中不包含可编程的 cluster。一些 CRA 和 FRA 被用来对应用进行专有设计。这种模式主要用来处理那些简单的、不规则的以及具有极高计算要求的 kernel,例如信号处理所用的超大 FFT、视频压缩中的熵编码等。HardKernel 被作为一个单独的 cluster 来处理流。在这种情况下,它的执行模式可能和其他的 cluster 不同。另外,由于 HardKernel cluster 的频率和其他 cluster 的很难匹配,因此可以使用流缓冲来适配其他 cluster 的处理过程。如图 3(d)所示。

多态的配置信息是以 kernel 为粒度进行加载的。这意味着并不是在 kernel 执行过程中改变配置信息,而是在一个或者几个 kernel 执行前就改变配置信息,这样做的优势是不容易造成编程混乱。指令流和配置流都是通过相同的数据通道在不同时间进行传输的,因此应用的执行被划分为两部分:配置时间和执行时间。

1.3 运行机制

在 MGR-SAT 中,标量处理核作为主处理器控制流处理核以协处理的方式运行,程序执行前,运行在标量处理核上的预设程序将首先对 CRA 和 FRA 进行配置,然后启动应用的执行。

标量处理核的主要功能有:(1)控制 MGR-SAT 的各种 IO 模块进行输入输出,比如 JTAG,AG,NI 等模块;(2)将数据以流的形式加载到 SRF,同时控制 SRF 在 CRA 和 FRA 之间的移动;(3)加载配置数据对 CRA 进行粗粒度配置,对 FRA 进行细粒度配置;(4)启动线程在 CRA 和 FRA 上执行,并监测其执行状态。

流处理核主要对应用进行加速处理。其中,CRA 执行 VLIW,可以开发指令集并行和数据级并行,根据应用的需求,可以配置工作在不同的并行模式下。FRA 以数据流的方式对特定算法进行加速处理,通过流缓冲可以与 CRA 共同构成一个完成数据流处理过程。

2 应用开发

MGR-SAT 的应用开发实际上是一个软硬件协同设计流程,它遵循“流化 - 配置 - 实现”的可重构架构思想,用户首先根据应用的特点和流模板,确定一个大体的流化实现方案;然后基于此方案,利用软件模拟器在一定的体系结构空间内进行模拟搜索,找到符合需求的目标处理器体系结构,以脚本配置文件的形式给出;在此基础上,对

实现方案(程序)进行修正,映射到具体的定制处理器上,包括硬件和软件设计。具体过程如图 4 所示。图中加色的矩形框表示工具,白色矩形框表示文件。

图 4 的上半部分示意了目标体系结构的搜索过程,该过程主要获得目标处理器的定制描述和应用程序代码。用户首先将应用的核心算法描述成 kernel(分为 SoftKernel 和 HardKernel),将核心算法间的数据移动描述成流传输(流控制);然后使用软件模拟器 Msim(本文作者所在课题组自行开发)对应用进行模拟,通过不断地调整可配置的参数,获得满足用户需求的 MGR-SAT 定制描述,然后根据此描述,对核心算法的实现进行修正。

图 4 的下半部分示意了应用程序的编译过程,该过程将应用程序映射到目标处理器上。在顶层,流级编译器 Mstream(基于 Istream[15] 开发)进行顶层的流调度,将 kernel 映射到不同的 CRA 和 FRA 上,并安排它们的流数据传输网络,优化不必要的数据传输,消除多余的中间结果。在底层,MSCD(基于 ISCD[15] 开发)将 SoftKernel 编译成 VLIW 在 CAR 引擎上(CAR engine)上执行,在此过程中 MSCD 将对通信进行调度,自动均衡 CRA 中功能单元和寄存器的使用率。对于 HardKernel,MSCD 基于用户定制库(customer lib)产生 FAR 的配置文件(Verilog 描述文件),然后借助于可编程逻辑的综合与布局布线工具,将配置文件转换为二进制网表,用于配置 FAR 引擎(FAR Engine)。在标量核上运行的目标代码(C scalar code)由程序员指定,通过 GCC 编译和链接直接分配到标量核上执行,在编译的过程中,需要获取 kernel 的大小,使得可以对 kernel 的加载和配置过程进行管理。

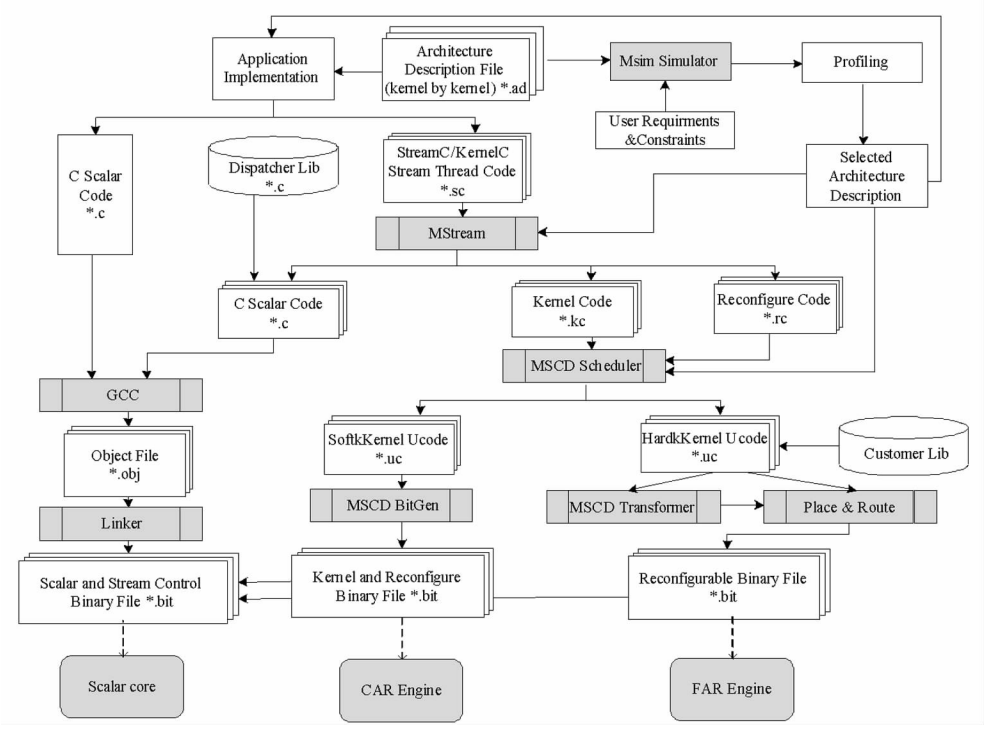

图 4　软硬协同设计流程

Fig.4　Hardware/Software co-design flow

MGR-SAT 使用修改的 StreamC/KernelC[15] 和 Verilog 语言进行软硬件协同设计。StreamC 用于描述 kernel(包括 SoftKernel 和 HardKernel)的外部接口和 kernel 之间的数据传输过程,KernelC 用于描述 SoftKernel 的内部实现,而 Verilog 用于描述 HardKernel 的内部实现。图 5 为一个简单程序的示例代码。该程序将两个数组进行相乘,然后对结果做一次 FFT 输出。数组的乘法以 KernelC 语言描述成 SoftKernel,被分配到 CRA 上执行,利用 VLIW 并行来获得较高的 ILP;FFT 使用 Verilog 语言进行描述,综合成二进制网表下载到 FRA 上执行,利用专用硬件进行加速;结果的输出由标量核控制外设来完成。

```
0:  Streamprog Test_Multicore(String arg){                    流程序定义
1:  int input0[128];
2:  int input1[128];
3:  int output0[128];
    ...
4:  im_stream<im_int> a(128);                                 定义stream
5:  im_stream<im_int> b(128);
6:  im_stream<im_int> c (128);
7:  im_stream<im_int> d (128);
    ...
8:  streamLoadBin(input0,128,a);                              调用StreamC发
9:  streamLoadBin(input1,128,b); //加载流                      射 stream op
    ...
10: kernel0(a,b,c);  //SoftKernel, 运行在CRA上
    ...
11: kernel1(c,d,128);    //HardKernel, 运行在FRA上
    ...
12: streamSaveBin(output0, d); //保存流
    ...
13: for(int k=0;k<128;k++){                                   嵌入式 C++ 代码
14:   cout << output0[k]; //输出, 运行在Scalar Core上
15: }
```

```
SOFT_KERNEL kernel0(
instream<int> a,b,
outstream<int> c,
configurefile "kernel0.rc")
{
 loop_stream() {
   int ai[], bi[];
   a>> ai[i];  b>> bi[i];
   c << ai[i]*bi[i];
 }
```

Kernel C 描述

```
HARD_KERNEL kernel1(
instream<int> e,
outstream<int> g,
parameter length )
{
 int e_in[length], e_out[length];
 e_in<<e;
 fft8(e_in(0, length/2),
      e_in(length/2,length)
      e_out(0, length));
 g<<e_out
}
```

连接到 Customer Lib (Verilog 文件)

图 5 程序示例

Fig.5 An example of program

3 实例分析

为了测试 MGR-SAT 的有效性,本文以典型的高性能数字信号处理应用 H.264 为例,在由 FPGA 和 ASIC 组成的硬件平台上对其进行了测试。

3.1 实验设置

测试程序 H.264 编码的主要过程如图 6 所示,涉及的主要算法有离散余弦变换(DCT)、量化(Quant)、反量化(DeQuant)、反变换(iDCT)、zig-zag 扫描、残差计算(SAD)、去块滤波(Deblock _ filter)、运动估计(ME)、帧内预测、帧间预测、熵编码(CAVLC)等。

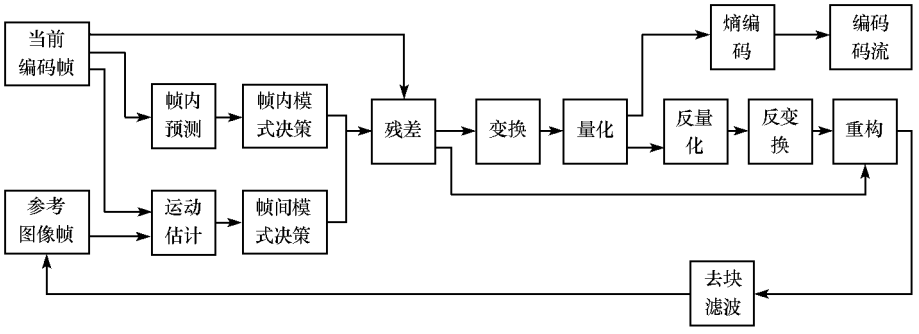

图 6 H.264 编码过程

Fig.6 Framework of H.264 coding

前期对核心关键算法的分析[8,16]表明,熵编码 CAVLC 的相关性强,串行特征明显,难以在并行处理器上得到很好的加速,需要使用专用硬件进行加速处理。同时,CAVLC 包含了大量的可以细粒度并行的位操作,虽然不能在通用处理器上得到加速(通用处理器的操作数都是 32 位或者更宽,位并行操作需要由多个 32 位操作完成),但是可以轻易地被专用硬件进行加速,且开销较小。因此本文将 CAVLC 作为 HardKernel 映射到 FRA 上去执行,而将其他的核心算法作为 SoftKernel 映射到 CRA 上去执行。

目标测试平台如图 7 所示,由 ASIC 和 FPGA 构成,同时包含一块高清输入板,为 H.264 编码提供规格为 1080p 的编码视频源。在该平台中,SPI 芯片[17](ASIC 芯片)用来模拟 CRA 的一种配置,而 FPGA 用来模拟 FRA 的一种配置,两者的片上互联通过高速串行传输通道来模拟。测试的过程如下:视频输入源由高清板 YPrPb 接口输入,经过 Stream IO[17]专有接口传给 SPI 芯片;在 SPI 芯片上经过预测、变换量化后传输到 FPGA 上;再经过硬件 CAVLC 加速处理,然后通过该板上的万兆以太网输出。

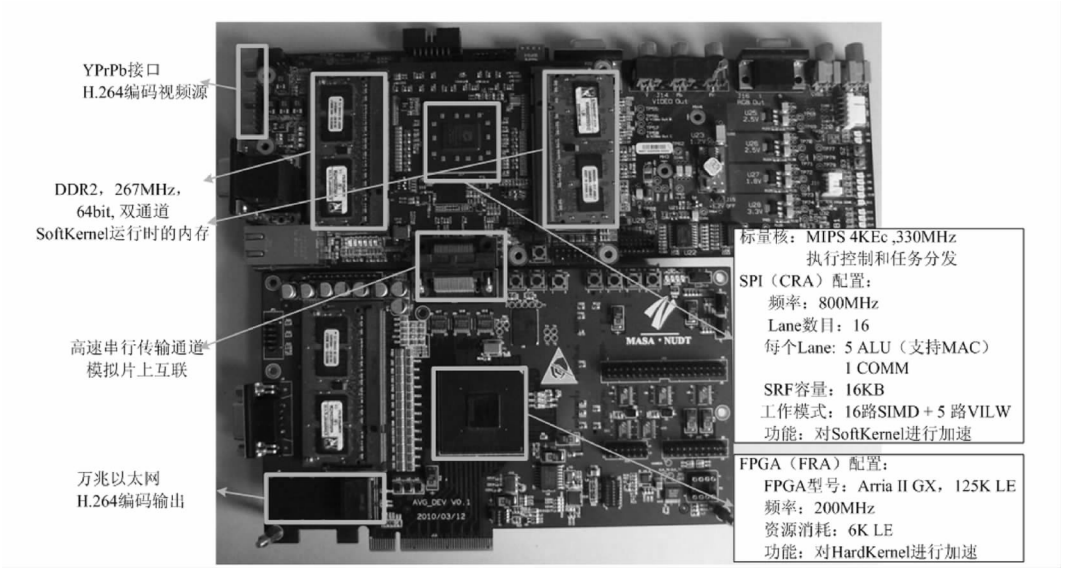

图 7　ASIC + FPGA 实验平台

Fig.7　Experimental platform base on ASIC and FPGA

3.2　结果分析

本文在图 7 所示平台上对多个视频输入序列进行了测试,结果如表 1 所示,从表中可以看出目标处理器在获得良好压缩效果的同时,可以达到 30 帧每秒的实时压缩性能,能够满足大多数高性能嵌入式应用的需求。

表 1　测试结果

Tab.1　Experimental Results

序列名称	序列内容	序列特点	分辨率	序列大小	帧数	压缩比	性能
Blue _ sky		对比度高，树冠部分细节较多，蓝天部分面积较大、色彩差异较小	1080P	659138KB	217	71:1	30.1fps
Station2		大量缓慢移动的汽车，有部分雾气遮掩，视角缓慢近移	1080P	950738KB	313	199:1	30.7fps
Rush _ hour		画面细节多，包含规则的铁轨的静态背景	1080P	1518750KB	500	112:1	30.5fps

为了更好地衡量 MGR-SAT 的性能,本文将 MGR-SAT 的性能与 X86 E7500(通用处理器)、TMS320(DSP)进行了比较,测试程序采用流化后的程序(流化后的 H.264 程序比标准的 H.264 程序性能有所提高[8]),结果如表 2 所示。从表 2 中可以看出,与当前主流的通用处理器和 DSP 相比,MGR-SAT 显示出强大的性能优势。

表 2　MGR-SAT 与其他主流处理器的性能比较

Tab.2　Performance Comparison

测试序列 / 目标处理器	Blue _ sky		Station2		Rush _ hour	
	性能	加速比	性能	加速比	性能	加速比
X86 E7500, 2 cores, 2.93GHz	10.5fps	2.87	10.3fps	2.98	9.85fps	3.10
TMS320 C6416, 1cores, 600MHz	1.57fps	19.17	1.54fps	19.94	1.42fps	21.48

值得注意的是,限于客观条件约束,本文的测试是在一个多芯片的环境中进行的,高速片间传输通道不能完全模拟片上互联的高带宽和低延迟,如果定制实现 MGR-SAT,那么可以获得更好的应用性能。

4 结 论

在高性能数字信号处理领域,流处理技术解耦合数据访存与运算,很好地暴露出算法数据局域性,表现出巨大的性能优势;而可重构技术则是传统的对关键算法进行加速的技术,近年来随着 FPGA 工艺的飞速发展,可重构技术正焕发出前所未有的活力;然而,在追求更高性能的同时,芯片的规模也在不断扩大,可靠性却不断降低,同时,其 time-to-market 越来越长,很难适应应用的需求,基于平台的技术则可以改变这一现状。本文对这些新兴的体系结构技术进行深入研究,将流处理技术、平台技术和可重构技术相结合,提出一种基于流模板的多粒度可重构数字信号处理器——MGR-SAT。实验结果显示,MGR-SAT 与同类型的通用处理器和 DSP 相比,有着明显的性能优势。

参 考 文 献:

[1] Khailany B, Dally W J, Kapasi U J, et al. Imagine: Media Processing with Streams [J]. IEEE Micro, 2001: 35 – 46.

[2] Taylor M B, Lee W, Miller J, et al. Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams [C]//ISCA2004, 2004: 2 – 13.

[3] Kozyrakis C. Scalable Vector Media-processors for Embedded Systems [D]. University of California at Berkeley, California, 2002.

[4] Sankaralingam K, Nagarajan A, McDonald R, et al. Distributed Microarchitectural Protocols in the TRIPS Prototype Processor [C]// 39th Annual International Symposium on Microarchitecture, 2006: 480 – 491.

[5] Jung H A, Dally W J, Kapasi U J, et al. Evaluating the Imagine Stream Architecture [C]//Proceedings of the 31st Annual International Symposium on Computer Architecture, 2004: 14 – 25.

[6] Chai S K, Chiricescu S, Essick R, et al. Streaming Processor for Next-Generation Mobile Image Applications [J]. IEEE Communication Magazine, 2005: 81 – 89.

[7] Hankins R A, Chinya G N, Collins J D, et al. Multiple Instruction Stream Processor [C]//Proceedings of the 33rd International Symposium on Computer Architecture, 2006: 114 – 127.

[8] Wu N, Wen M, Wu W, Ren J, et al. Streaming HD H.264 Encoder on Programmable Processors [C]//ACM Multi-Media 2009, Beijing China, 2009: 371 – 380.

[9] Chien S Y, Chen T H, Chen J C, et al. Course-Grained Reconfigurable Image Stream Processor Architecture for Embedded Image/Video Processing and Analysis[C]//ICME2009, 2009: 1578 – 1579.

[10] Singh H, Lee M, Lu G, et al. Morhposys: Case Study of a Reconfigurable Computing System Targeting Multimedia Applications [C]//Proc. Design Automation Conference (DAC'00), Los Angeles, California, 2000: 573 – 578.

[11] PACT XPP Technologies [EB]. The XPP-III White Paper. Release 2.0.1, 2006.

[12] Texas Instruments Inc[EB]. http://focus.ti.com/pdfs/wtbu/ti_omap2430.pdf, 2009.

[13] Triscend A7S Configurable System-on-Chip Platform [EB]. www.triscend.com, 2010.

[14] Rixner S, Dally W J, Kapasi U J, et al. Memory Access Scheduling [C]//Proceedings of the 27th Annual International Symposium on Computer Architecture, 2000: 128 – 138.

[15] Mattson P, Kapasi U J, Owens J D, et al. Imagine Programming System User's Guide [EB]. http://cva.stanford.edu, 2002.

[16] Wei W, Wu N, Ren J, et al. A Streaming Implementation of HD H.264/AVC Encoder on STORM Processor [C]//IEEE Multimedia Computing and Information Technique (MCIT2010), 2010: 97 – 100.

[17] Stream Processors Inc. Stream Processor Architecture [EB]. http://www.streamprocessors.com, 2008.