

基于模板的粗粒度可重构处理器编译环境设计

谢羽威, 葛冰晶, 绳伟光, 王 琴, 毛志刚

(上海交通大学 微电子学院, 上海 200240)

摘 要: 粗粒度可重构处理器以优良的结构和出色的性能日益受到广泛的应用, 其丰富的运算资源为应用程序高效并发执行提供了可能. 本文面向 REMus II 粗粒度可重构处理器硬件架构, 首次提出一种基于模板的程序编译方法. 从应用程序中的计算密集型代码出发, 定义了程序模板的格式和规范, 建立模板开发的设计流程, 设计了面向视频图像处理的程序模板库. 在此基础上, 面向程序员设计了基于模板的程序语言标注技术, 并最终完成了相应编译环境的构建. 实验结果表明, 该编译环境有效提高了编译后代码在可重构处理器上执行效率和编译速度.

关键词: 可重构处理器; 编译环境; 编译器; 模板; 标注

中图分类号: TN402

文献标识码: A

文章编号: 1000-7180(2013)07-0037-05

A Template-Based Compiling System Design for Coarse-Grained Reconfigurable Processors

XIE Yu-wei, GE Bing-jing, SHENG Wei-guang, WANG Qin, MAO Zhi-gang

(School of Microelectronics, Shanghai Jiaotong University, Shanghai 200240, China)

Abstract: Coarse-grained reconfigurable processors are increasingly popular by their optimized architecture and outstanding performance, which make it possible for applications to run with high parallel efficiency. In this paper we introduce for the first time a template-based compiling system on REMus II coarse-grained reconfigurable processor. Starting from the compute-bound code parts, we create a template-lib system for typical multimedia decoder algorithms, which involves the template structure and developing flow design. Then we complete the compiling system with an annotation technique on programming codes. The results show that the compiling system effectively increases the compiled codes' operating efficiency as well as the compiling speed.

Key words: reconfigurable processor; compiling system; compiler; template; annotation

1 引言

可重构计算(Reconfigurable Computing)是一种将软件的灵活性和硬件的高效性有效结合的计算方式. 可重构处理器以现场可编程门阵列(FPGA)为代表, 具有与通用计算处理器相似的灵活性, 在性能/功耗比上也正在接近专用计算, 在性能、功耗、灵活性等关键指标之间取得了很好的折中, 因而日益受到广泛的重视、研究和应用^[1]. 其中, 粗粒度可重构架构(Coarse-Grained Reconfigurable Architecture, CGRA)处理器是采用最小处理宽度大于 1 bit 的可重构单元(常见的为 8 bit 或以上)构成的可重

构阵列来对计算进行加速的处理系统. CGRA 比 FPGA 的细粒度结构更为规整, 片内互连所占面积更小, 全定制的运算单元(Processing Element, PE)更高效. 此外, CGRA 采用操作符级或字级的重构操作, 其需要的配置存储器也较少, 配置时间与 FPGA 的布局布线所需时间相比也大为减小^[2].

可重构处理器优良的结构设计和出色的性能对程序员和编译器也提出了更高的要求. 在传统的指令流和数据流之外, 可重构处理器还引入了构令流(Configuration Stream)的概念^[3], 使得应用到目标处理器平台的映射过程变得更加复杂. 程序员和编译器需要对可重构处理器的硬件架构有深入细节的

了解,生成高效的代码,因而大大限制了可重构处理器的应用和发展^[4].其中,高度优化的编译器可以将代码高效映射到可重构处理器上,降低编程难度,并极大地提高应用执行效率.因此,可重构处理器的编译技术是可重构处理器的重要研究内容,也是其中的难点之一.目前实验室 REmus II 粗粒度多媒体可重构处理器的编译环境基于一套信号流图(Data Flow Graph, DFG)生成的编译机制,实现了应用程序的全自动编译^[5],但是其编译流程通用性的特点决定了编译后代码性能的局限.

为了进一步提高 REmus II 的编译性能,本文设计一种基于模板的可重构处理器的编译环境.通过建立和设计多媒体模板库,提出一套改进的程序语言标注技术,将其在现有编译器中集成,实现对常用的多媒体算法,调用模板直接抽取性能更好的手工硬件配置信息,提高编译后代码的执行效率.同时,对现有的编译流程保留最大兼容性,由程序员代码自由灵活决定调用方式.

2 基于模板的编译环境基本组成

REmus II 粗粒度可重构媒体处理器^[6]是一款基于 ARM7TDMI 和可重构运算阵列(Reconfigurable Processing Unit, RPU)的多媒体解码芯片,主要面向嵌入式多媒体系统的开发和应用,其系统架构如图 1 所示. RPU 模块是一个动态可重构的硬件运算单元阵列,有近似于 ASIC 的性能. REmus II 包括两个 RPU,每个 RPU 由 4 个可重构运算单元(Reconcilable Cell Array, RCA)组成, RCA 内部是多个 PE 的排列,这种分组结构允许用户充分利用 REmus II 的加速性能来执行并行运算.

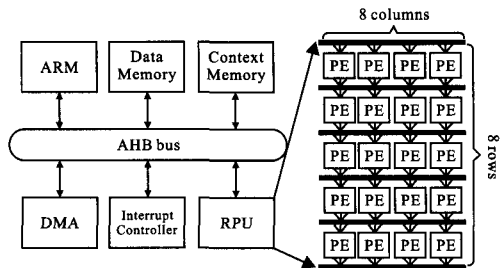


图 1 REmus II 粗粒度可重构处理器的系统架构

现有的 REmus II 编译环境^[5]基于“程序”和“编译器”两部分的框架.首先由程序员分析程序中的运算密集型代码,标注出需要映射到处理器的部分,然后从命令行调用编译器对代码进行编译.编译器首先对源代码进行扫描,将相应代码转化为 DFG 表示,然后对 DFG 进行子图分割、映射、内存分配等

操作,生成硬件配置信息.同时,编译器将这些代码用包含有配置信息的宏替代,生成编译后的源代码,最后调用传统编译器对新生成的工程进行编译,生成可执行文件.

编译器的通用性特点决定了其未能针对一些常用经典算法进行必要的优化,而通常这些算法在整个工程中的占空比较大.若能根据硬件资源特点对其进行高效的划分和映射,将极大提高该算法在可重构系统上的执行效率.本文提出了模板这一机制.针对常用的多媒体算法,编译器通过调用模板,直接抽取性能更好的手工硬件配置信息作为编译结果,以提高编译后代码的执行效率.

建立和设计多媒体模板库作为编译器的附属工具.模板库是若干模板的统一组织结构,由一系列自动化工具支持,并对编译器提供调用接口.其中各模板的主要内容是视频编解码、图像处理 and 音频编解码等算法的手工硬件配置信息.模板库需由编译器开发人员或模板库管理员负责管理,各模板需要由模板开发人员手工独立开发.整个编译环境的结构关系如图 2 所示.

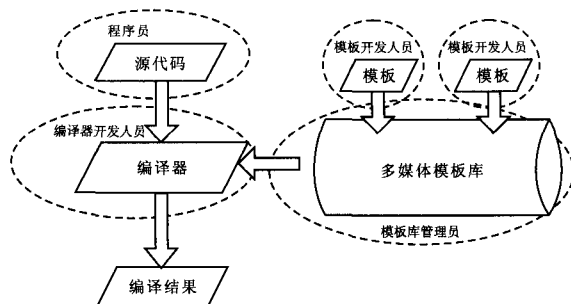


图 2 基于模板的编译环境结构关系

根据上述的编译环境,基于模板的总体编译原理可以用图 3 表述.首先用手工方法得到常用经典算法的硬件配置信息,将其预先包含进多媒体模板库中.编译器从相关源代码生成 DFG 之前,首先搜索此算法是否在模板库中已经存在,若存在则检查输入输出参数是否符合规则要求.若符合,则将相应的配置信息提取出来,直接进入后续编译;若不符合,则按照文献^[5]的一般流程编译.

3 多媒体模板库的设计

在整个模板库系统中,模板文件是整个系统的核心,它包含编译器调用模板时所需要的所有相关文件和硬件配置信息.模板文件由模板设计人员开发生成,首先通过配置文件生成工具从应用算法手工映射得到模板文件,在模版验证环境中验证其正

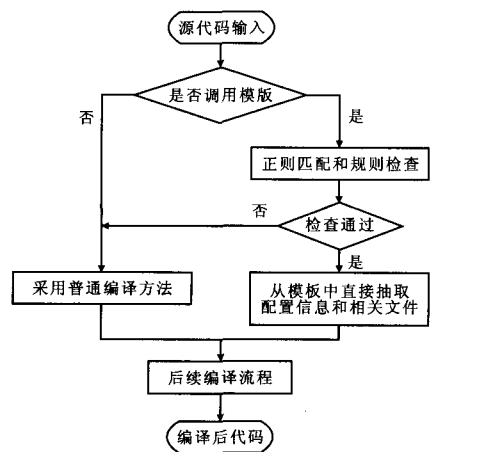


图 3 基于模板的可重构处理器基本编译原理

确性,生成模板调用接口,然后使用模板库管理工具将模板文件与模板调用接口集成到整个系统中.其整体生成流程由图 4 所示.

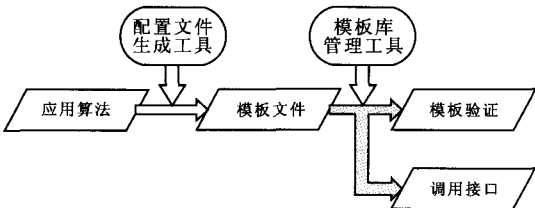


图 4 模板文件生成流程

每个算法的模板文件按照规定的格式、命名和目录结构放置.对于 REmus II 可重构处理器,一个典型的模板需要由表 1 所述的各文件构成.

表 1 一个 REMUS II 模板的文件构成

模板文件	用途
硬件配置信息 (ConfigWord)	用于配置可重构阵列,实现硬件加速功能.由编译器抽取使用.
接口宏文件 (Interface)	定义可重构阵列的输入和输出数据在物理地址上的对应关系.由编译器抽取使用.
规则文件 (Rule)	模板库参数接口在标注中的顺序说明.用于正则匹配和规则检查.
标准输入/输出文件 (I/O Pattern)	标准测试输入/输出流.用于模板后续优化验证.
算法源文件和说明文档	表明模板功能和接口的源文件.用于模板库维护和说明.

对于 REmus II 可重构处理器,其硬件配置信息 (ConfigWord) 由 ConfigWord、GroupWord 和 ContextWord 三层配置字组成,每一层都有相应规定的文本格式.接口宏文件 (Interface) 是一个规范化的 C 头文件,包含了可重构阵列输入输出数据的物理地址,用于编译器抽取并将其替换到用户指定需要加速的源代码中.规则文件 (Rule) 是多媒体模板库专用的接口相关文件,规定了模板的接口参数个数、命名和调用顺序,用于正则匹配和检查模板调用是否正确,其格式为:

```
input
// INPUT COUNT
// INPUTS
output
// OUTPUT COUNT
// OUTPUTS
```

以一个矩阵相加 (matrix_add) 模板为例,其需要两个输入参数 matrix_A、matrix_B 和一个输出参数 matrix_SUM,则其规则文件 add_matrix_rule 为:

```
input
2
matrix_A
matrix_B
output
1
matrix_SUM
```

为保持编译环境的模块化管理,模板的硬件配置信息、接口宏文件和规则文件等在通常情况下对于程序员是不可见的.模板开发人员需要将模板的说明文件和调用接口在说明文档中正确完整地注明,以便编译器正确识别和调用.

4 基于模板的程序语言标注技术

可重构系统通常由一个或多个可重构运算阵列 (RPU) 组成,每个 RPU 可能包含一个或多个可重构运算单元 (RCA),RCA 内部是多个运算单元 (PE) 的排列. REmus II 可重构处理器包括两个 RPU,每个 RPU 由 4 个 RCA 组成.这种分组结构允许程序员充分利用 REmus II 的加速性能来执行并行运算.为实现程序员与编译器的交互,采用了程序语言的标注技术,即程序员通过在源代码中添加编译指导来指定源代码在 RPU 上的执行方式,编译器将检测用户标注并对指定的代码段进行编译,将其转换成 DFG 并进行后续处理操作.源代码标注的标准格式为:

```
#pragma message("DFG_BEGIN:\nFILENAME__FUNCNAME__DFGNAME__CFGIN-FO");
// code to accelerate
#pragma message("DFG_END");
```

各标注参数的意义如表 2 所示.

基于模板的程序语言标注技术是对上述标注技术的改进,其目的是在兼容上述标注功能的前提下加入多媒体模板库支持,同时兼顾模板调用的便利性,避免让用户重复书写代码.基于模板的编译器支持两种形式的模板调用标注:一种是类似于传统编译流程

表 2 用户标注中各参数意义

标注信息	意义
DFG_BEGIN	标记需要加速代码的起始
FILENAME	代码所在的源文件名
FUNCNAME	代码所在的函数名
DFGNAME	用户指定的 DFG 编号
CFGINFO	RPU 功能配置, 包括 RPU 编号、组号、RCA 编号和同步方式等信息
DFG_END	标记需要加速代码的结束

的标注扩展,在函数实体处进行标注.这种标注的优点是与原编译器的兼容性好,易于调试,推荐使用这种标注方法;另一种是直接函数调用处进行一句式标注,这种标注的优点是直观,代码可读性高,缺点是与原版的兼容性差,而且 RPU 功能配置不易于管理.

使用函数实体处标注时,RPU 配置信息(即原 DFG_BEGIN 标记)必须保留.模板库的标注信息必须用一个 LIB 标记紧跟在 DFG_BEGIN 标记后,写在同一个 pragma 语句内,其格式如下:

```
#pragma message("DFG_BEGIN;\n
FILENAME__FUNCNAME__DFGNAME__CFGINFO__\n
LIB:LIBNAME INPUTS OUTPUTS");
// code ignored by compiler
#pragma message("DFG_END");
```

其中 LIBNAME 是调用的模板名,INPUTS 和 OUTPUTS 是所有传入和传出变量的列表,按照模板规定的接口参数意义和顺序排列.标注其它部分的意义同表 2 所述.将带有此标注源文件提交编译时,编译器将忽略该标注部分的任何代码实现,直接输出相应配置信息.

使用函数调用处标注时,需要在出现标注的文件上引用模板库固定头文件,然后将需要调用模板库的函数用 LIB_BEGIN 标注出来即可,标注格式

如下:

```
#include "remus_template_lib.h"
//some other code
#pragma message("LIB_BEGIN;\n
FILENAME__LIBNAME__DFGNAME__CFGINFO__\n
FO");
// do function call
```

其中 FILENAME 为代码所在的源文件名,LIBNAME 是调用的模板名,标注的其余部分同表 2 所述.该标注方法的优点是不需要具体函数实现.编译器在编译过程中将主动生成包含相关函数实现的源文件,输出预期的编译结果.

5 编译器的模板库集成

REmus II 粗粒度可重构处理器的编译器总体编译流程如图 5(a)所示.编译器的输入为带有标注信息的源代码,经过预处理、DFG 生成、宏替换、配置信息生成和后续处理共五个处理模块,生成处理后源代码、接口宏文件和硬件配置信息输出.其间生成的其它中间文件(如 dfg.list 和 DFG 文件等)由编译器回收处理.

基于模板的编译器是对现有编译器的功能扩展,其总体编译流程是对一般编译流程的改进,如图 5(b)所示.编译器的输入仍为带有标注信息的源代码,调用模板时,模板相关的标注被预处理和模板调用模块识别.若识别成功,则调用模板进行操作,编译器将对中间文件 dfg.list 进行适当标记,告知配置信息生成模块直接从模板抽取配置信息进行操作;若识别失败,则采用一般流程进行自动编译.

模板识别过程主要基于算法 1 的伪代码,并集成在图 5(b)的预处理模块内.该算法首先判断当前

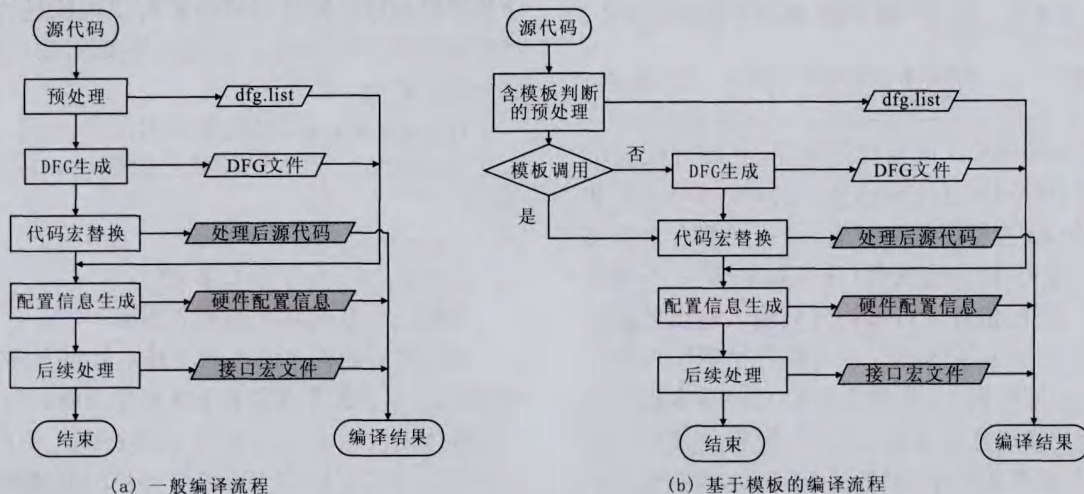


图 5 REmus II 可重构系统的编译流程比较

代码行是否为一个合法的模板调用标注,若合法,则将调用的模板名和模板参数传入 `checkAndGetPathFromLib()` 函数进行正则匹配和规则检查,并尝试获得模板文件路径。若匹配成功,则根据两种标注格式的情况分别处理。对于函数实体标注,首先对中间文件 `dfg. list` 进行模板化标记,然后更新接口宏文件内的变量名,最后将标注内部的函数语句全部清空;对于函数调用处标注,则首先检查函数实体是否已经存在,若不存在,则生成一个空的函数实体,然后继续执行上述过程。

算法 1 模板标注识别算法伪代码

```
1 if isLibPragma (current_line) begin
2   lib_name = getLibName(current_line);
3   lib_parameters = getLibParameters(current_line);
4   lib_src_path = checkAndGetPathFromLib (lib_name,
5     lib_parameters);
6   if ! isValid (lib_src_path) return ERROR;
7   if isNormalLibPragma (current_line) begin
8     putIntoDFGList (lib_name, lib_parameters, lib_src_
9       path);
10    updateInterfaceFile (lib_src_path, lib_parameters);
11    current_line = getNextLine();
12    while ! isEndPragma (current_line) begin
13      deleteLine (current_line);
14      current_line = getNextLine();
15    end
16  end
17 else begin // LIB_BEGIN pragma
18   if ! functionExists (lib_name, lib_parameters) begin
19     addFunctionWithPragmaToFile (lib_name, lib_paramete-
20       ters);
21   end
22 end
23 // do normal compiler procedures
24 end
```

6 性能分析和比较

使用 H. 264 视频解码流程中的几个经典算法,将基于模板的编译器与现有编译器作性能和编译时间方面的比较,其中各算法均由手工开发生成模板,结果如表 3 所示。其中,将 REmus II 处理器的两两 RCA 之间设置为串行操作,由于一次 RCA 配置和运算时间为恒定,则每个算法的性能指标可以通过编译生成的 RCA 个数来反映。编译在同一硬件和软件环境下进行,编译操作各做三次,编译时间取三次的平均值。可以看到,基于模板编译出的 RCA 数量较少,编译时间也减少到相对恒定。

表 3 一般编译与基于模板的编译性能比较

算法名称	基于文献[5] 的一般编译		基于本文 模板的编译	
	RCA 数量 (个)	编译时间 (s)	RCA 数量 (个)	编译时间 (s)
idct4_h	3	6.0	1	5.0
idct8_h	11	10.0	2	5.0
luma_mc_01_c_8x8_1	14	14.0	5	6.0
luma_mc_11_c_8x8_1	17	16.0	7	7.0
loop_filter_chroma_c_1	8	10.0	3	3.0
loop_filter_luma_c_1	25	19.0	5	4.0

7 结束语

本文提出了一种基于模板的粗粒度可重构处理器编译环境。以实验室 REmus II 粗粒度可重构处理器为例,通过建立和设计多媒体模板库,改进程序语言标注技术以及在现有编译器中集成这三部分的设计和任务划分,实现对常用的多媒体算法,编译器直接抽取高效的硬件配置信息生成编译结果,同时对现有的编译流程保留最大兼容性。通过与一般编译流程的比较,证明了基于模板的编译流程可以有效提高编译后代码在粗粒度可重构处理器上的执行效率和编译速度。

参考文献:

[1] Compton K, Hauck S. Reconfigurable computing: a survey of systems and software [J]. ACM Computing Surveys, 2002, 34(2):171—210.

[2] Hartenstein R. A decade of reconfigurable computing: a visionary retrospective [C]// Proceedings of the Conference on Design, Automation and Test in Europe (DATE' 01). Termamy; Munich, 2001:642—649.

[3] Coussy P, Morawiec A. High—devel synthesis; from algorithm to digital circuit [J]. Computer, 2000, 33 (4):70—77.

[4] Cardoso J M P, Diniz P C, Weinhardt M. Compiling for reconfigurable computing: a survey [J]. ACM Computing Surveys, 2010, 42(4):1—65.

[5] Sheng W, He W, Jiang J, et al. Full automatic task compilation flow from C to REmus coarse grain reconfigurable media processor[J]. Journal of Convergence Information Technology, 2012, 6(12):193—202.

[6] Zhu Min, Liu Leibo, Yin Shouyi, et al. A reconfigurable multi—processor SoC for media application[C]// ISCAS 2010. Paris; IEEE Press, 2010:2011—2014.

作者简介:

谢羽威 男,(1987—),硕士研究生.研究方向为可重构处理器的编译技术。