

基于循环优化的可重构处理器任务编译器设计实现

栗念龙, 唐世卓, 何卫锋

(上海交通大学 微电子学院, 上海 200240)

摘 要: 任务编译器是粗粒度可重构处理器系统中的重要支撑软件. 针对 REMUS-II 可重构处理器系统架构特点, 提出了基于循环优化的任务编译方法. 该方法通过核心循环模版提取、配置信息复用及数据流优化, 达到缩短编译时间, 降低重构开销, 提高可重构系统执行效率的目的, 同时大幅缩小配置信息体积. 实验结果表明, 该配置方法与传统可重构编译技术相比, 编译时间缩短 40%, 总配置信息量减少超过 60%.

关键词: 可重构计算; 任务编译器; 配置复用; 数据流优化

中图分类号: TN41

文献标识码: A

文章编号: 1000-7180(2013)08-0124-04

Design and Implementation of Task Compiler Based on Loop Optimization for Reconfigurable Processor

LI Nian-long, TANG Shi-zhuo, HE Wei-feng

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

Abstract: Task compiler is a vital element of the coarse-grained reconfigurable computing system. In this paper, a loop-level optimized compiling method for REMUS-II reconfigurable architecture is provided. This method reduces compilation time, the overhead of reconfiguration and the amount of the configuration contexts based on kernel abstracting and data flow optimization. According to experimental results, proposed compiling method reduces compilation time by 40% and configuration contexts amount by 60%, compared with conventional compiling techniques.

Key words: reconfigurable computing; task compiler; configuration reuse; data flow optimization

1 引言

可重构处理系统因其结合了微处理器的灵活性和 ASIC 的高效性而逐步走进研究者视野. 典型的可重构处理器结构由主控制器和可重构硬件耦合^[1-3], 其中主控制器负责分支跳转、转移判断等控制操作, 以及配置和管理可重构硬件等工作; 可重构硬件主要处理运算密集型操作. 如 MorphoSys^[4]、ADRES^[5-6]、XPP-III^[7]. 由于可重构处理器的硬件结构与传统通用处理器(GPP)结构存在显著差别, 没有对应的成熟任务编译器, 限制了可重构处理器适用领域和范围, 因此提供针对可重构系统良好的任务编译支持十分必要. 可重构任务编译器功能即通过任务划分、代码变换、任务调度及映射等过程,

将程序编译成主控制器的控制码和可重构硬件的配置信息. 目前国内对可重构处理器研究中偏向于硬件架构研究, 而根据国外研究机构的任务编译器, 如 XPP-VC^[8]、Garp-C^[9], 可知针对不同可重构硬件需要对应的任务编译器.

本文针对一款可重构媒体处理器(REMUS-II), 设计基于循环优化的可重构任务编译器. 该编译器主要包括以下关键技术: 循环部分展开, 循环配置复用, 数据流优化. 通过对程序源码的软硬件划分, 循环优化, 任务调度和映射, 生成可重构阵列(RCA)可执行的配置信息及硬件接口.

2 REMUS-II 硬件架构

REMUS-II 处理器主要由主控制器(ARM)、可

重构单元(RPU)、系统总线、存储器、直接存储访问控制器(DMAC)和其他外设构成,其简化结构示意图如图1所示.其中RPU负责计算密集型运算,一个RPU包含4个可重构阵列(RCA),RCA间通过共享内存方式进行数据交互,因此仅有一个外部数据访问接口.

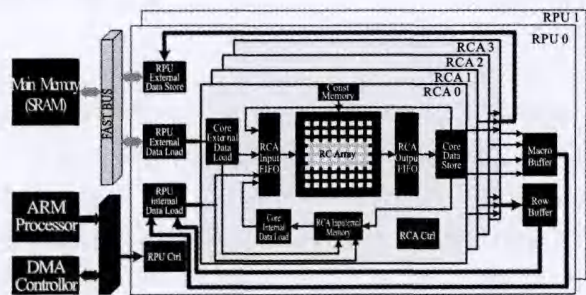


图 1 REMUS-II 结构简化示意图

2.1 RCA

RCA 是 REMUS-II 处理器中重要组成部分, 内部主要由运算模块、存储模块、数据管理模块、控制模块构成. 运算模块由一个规模为 8×8 的 2-D 可重构单元阵列(RC Array)构成, 其基本单元为可重构运算单元(RC), RC 可依据不同配置信息配置为不同功能的算子. 存储模块主要包括输入、输出 FIFO、常数存储器, 同时为了解决数据频繁访问外部存储器的缺点, RCA 中引入中间存储器(RIM), 暂存运算数据, 进一步提升系统性能. 数据管理模块负责数据的拆分、拼接、组合、传输, 其包含多种不同的存储目标及传输模式, 因此增加了任务编译器的设计的难度. 控制模块主要根据 GPP 对阵列的启动、运算、终止进行控制.

2.2 REMUS-II 编程抽象模型

可重构处理器结构的特殊性导致其运行方式的复杂性,其既需要 GPP 上运行的可执行程序又需要 RCA 配置信息,因此需要编译器提供一个简单抽象的编程模型来供程序员使用. REMUS-II 编程抽象模型如图 2 所示,该模型中 RCA 内部各控制模块的具体实现细节对程序员透明,使程序员仅需将计算密集型运算加标注以放置在 RC 阵列上执行.

任务标注格式分为前标和后标,表示任务开始和结束的位置分别如下:

```
# pragma message("DFG_BEGIN");
```

```
# pragma message("DFG_END");
```

前标可以添加参数,以明确执行阵列位置 (“DFG_BEGIN:file__function_NODE— $b_0-b_1-b_2b_3-b_4b_5b_6$ ”);各参数含义如表1所示.

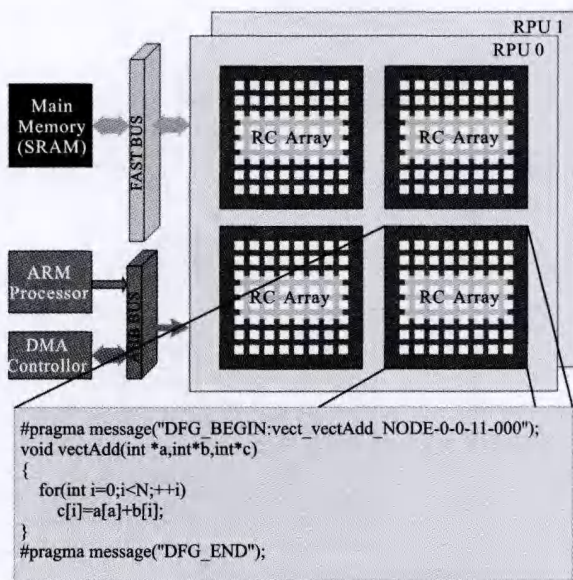


图 2 REMUS-II 编程抽象模型

表 1 标记参数定义

标记参数	定义
file	任务所在文件名
function	任务所在函数名
b ₀	bit 类型, 0 表示 RPU0; 1 表示 RPU1
b ₁	int 类型, 表示执行的第 b ₁ 组任务
b ₂ b ₃	bit 类型, 对应 RPU 中阵列编号, 取值 0~3
b ₄ b ₅ b ₆	bit 类型, 同组任务阵列同步模式, 取值 0~7

图 2 中给出了一个简单的标注实例,表示该任务在 RPU0 中第 0 组运算在 RCA3 上执行,同步模式为 0,即各 RCA 无数据相关、无等待,独立执行。

3 基于循环优化任务编译器

REMUS-II 任务编译器对源程序进行代码划分、分析、转化、优化、映射,生成 REMUS-II 可重构处理器的可执行文件。编译流程图如图 3 所示,任务编译器输入高级语言程序(如 C/C++ 程序),并通过标注在预处理阶段划分主控核和可重构阵列中执行的任务;将可重构阵列执行代码生成数据流图(DFG);通过 DFG 图的解析、时域划分、序列化和存储约束生成 RPU 可执行配置信息和接口文件;整合接口文件和主控制器程序通过标准 C 编译器得到主控制器的执行文件。

REMUS-II 主要针对多媒体运算,而其中大部分算法的执行时间主要消耗在循环体内,因此对循环的优化能有效地提升系统的执行能力.基于此,在任务编译器的编译技术中创新性地引入循环展开、循环配置复用和数据流优化等关键技术.

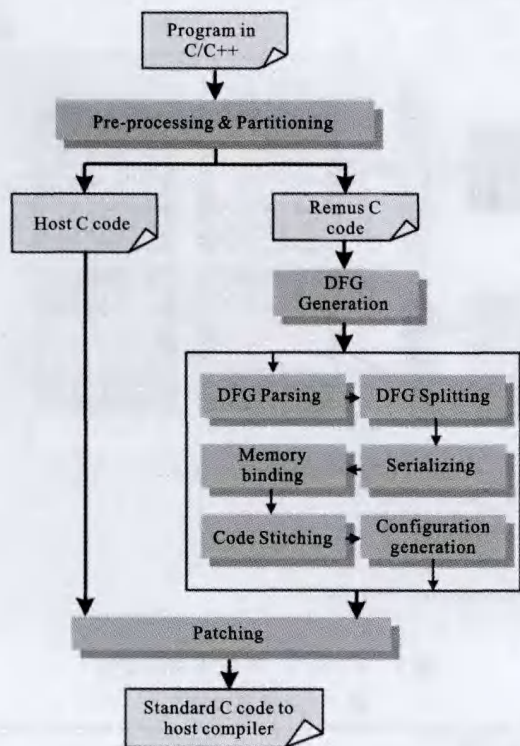


图3 任务编译器流程图

3.1 循环展开

循环展开是指独立的循环迭代通过多次复制循环体和终止代码来并行执行. 循环展开后能有效提高循环体内程序并行度, 减少转移开销.

任务编译器中, 循环展开生成的 DFG 数据依赖关系明确, 易于 DFG 根据关键路径划分, 减少 RCA 的使用数目和数据传输开销. 如图 2 中的简单向量加的例子, 未采用循环展开的情况下, 每次循环迭代需要占用一个 RCA. 而循环展开后, 该代码中循环迭代间没有数据依赖, 因此可完全并行执行, 在 8×8 RCA 阵列中可将 64 次循环一次执行完毕.

循环展开分为循环全展开和部分循环展开. 全展开是将整个程序段从内层循环到外层循环依次展开, 不存在跨迭代的数据依赖关系. 但全展开的 DFG 图较为复杂, 不宜编译器处理产生配置信息. 部分展开是将内层循环按一定规模复制迭代次数, 同时改变循环终止条件, 生成适合硬件处理的 DFG 规模, 从而有效实现处理规模和并行度的平衡.

3.2 循环配置复用

由于循环全展开的 DFG 节点较多, 关键路径较长, 编译器处理较慢. 而部分循环展开后对于未展开的程序段需要重复配置, 降低了系统执行效率. 由此提出了循环配置复用以降低 DFG 图的复杂度, 减少配置开销. 循环配置复用就是使用部分循环的全展开, 或者仅对内层循环展开, 得到一个 DFG 通用模

板, 可以根据外层循环的次数来多次复用该模板的配置信息. 该方法使处理器一次配置可以多次执行, 加快了执行速度, 降低了配置信息量, 减少重构时间.

循环配置复用的核心在于得到一个适合规模的 DFG 模板和多次循环的输入数据, 通过优化和子图同构来将各次循环抽象成一套 DFG 模板, 并且将各次循环输入输出端口归一化, 即输入输出端口和循环每次的输入输出变量对应. 图 4 中给出了一个简单的四点离散反余弦变换 (IDCT4) 的例子, 假设可重构阵列 4×4 , 循环运算 4 次, (a) 为 IDCT4 的 DFG 图, 其中圆形图标为输入, 方形图标为运算节点. (b) 和 (c) 分别为 IDCT4 循环配置复用和未循环配置复用时硬件上的映射方式. 由图 4 可知, 配置复用后所使用的 RCA 个数由 3 减到 1, 并且随着循环次数的增加, 有更好的性能提升.

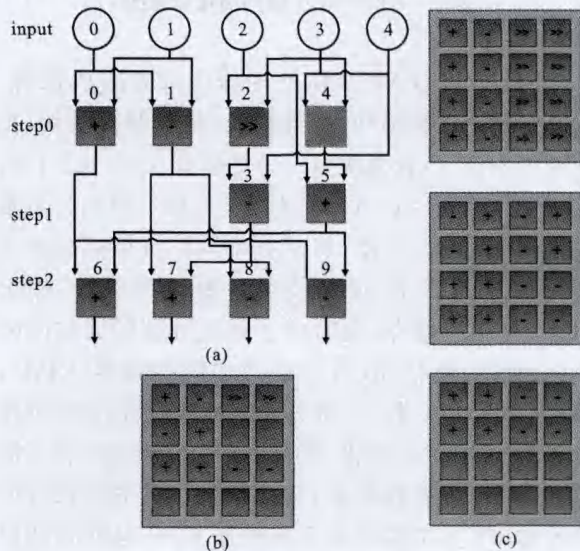


图4 idct4 DFG 及硬件映射图

3.3 数据流优化

循环配置复用的使用, 会导致每次配置的数据量随着复用次数而线性增加, 并且由于输入数据来源较多, 导致数据传输开销较大. 而数据流优化可以使数据块存储, 并通过数据管理模块对数据进行块传输, 大大提高了数据传输效率.

数据流优化整理、梳理各 RCA 的输入源, 实现外部输入数据仅从第一个 RCA 输入, 后继 RCA 的数据输入仅从前驱 RCA 获得, 数据输出只能由最后一个 RCA 执行. 因此每个 RCA 的数据依赖关系仅与一个父 RCA 相关, 解决了因外部数据和内部数据的不同来源对数据传输影响的问题.

4 验证及性能比较

REMUS-II 主要面向嵌入式多媒体应用, 因此

验证选用了媒体处理中常用的算法 H. 264/MPEG2/AVS, RPU 功能单元中主要完成矩阵运算、离散反余弦变换(IDCT)、运动补偿(MC)、去块滤波(DEBLOCKING)等。

实验分析结果如图 5 所示,横坐标为 H. 264 中部分用 RPU 执行的函数编号(1、2 为 add_clip, 3—6 为 h264_h_loop_filter, 7—10 为 idct, 11—42 为 h264_luma_mc), 纵坐标为 RPU/ARM 核心加速比, 即相同函数 RPU 执行和 ARM 执行的速度比值。由图可知, 优化前编译器编译结果在 RPU 上执行的平均加速比为 5, 而循环优化后这一平均加速比能达到 15, 极大地提升了系统的性能。

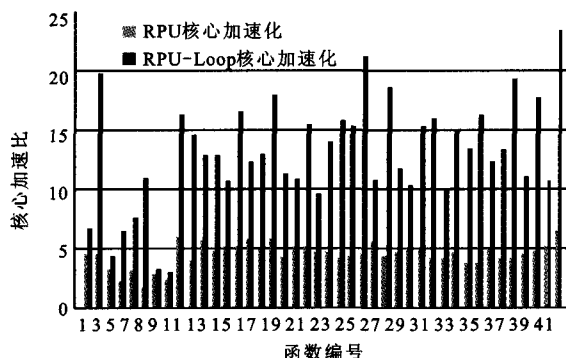


图 5 RPU 循环优化性能图

循环优化后的编译器由于减小相同功能下的 DFG 规模, 因此编译速度也有所提升; 并且配置复用大大减少了配置信息量。表 2 中列出了 H. 264 中关键任务的编译时间和配置信息压缩率。由表可知, 优化后编译器编译时间缩短约 40%, 配置信息量减少超过 60%。

表 2 编译器性能比较表

任务名	编译时间/s		配置信息压缩率/%	
	优化前	优化后	DCC ^[10]	本文
IDCT	7.152	5.300	36.87	63.7
MC	7.083	5.155	38.98	83.4
loopfilter	6.846	5.383	38.67	60.6

5 结束语

本文针对 REMUS-II 处理器, 提出了基于循环优化的任务编译器, 通过部分循环展开, 配置复用和数据流优化等技术解决了任务编译技术中配置信息量和数据传输开销太大的问题。实验结果显示, 该任务编译器对媒体处理中的关键算法编译结果执行效率较通用处理器平均有 15 倍的加速比, 同时相比传统可重构编译器编译时间缩短 40%, 总配置信息量减少 60% 以上。

参考文献:

- [1] Khawam S, Nousias I, Milward M, et al. The reconfigurable instruction cell array[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2008, 16(1): 75—85.
- [2] Todman T J, Constantinides G A, Wilton S J E, et al. Reconfigurable computing: architectures and design methods[J]. Computers and Digital IEEE Techniques Proceedings, 2005, 152(2): 193—207.
- [3] Compton K, Hauck S. Automatic design of reconfigurable domain-specific flexible cores[J]. Very Large Scale Integration (VLSI) IEEE Transactions on Systems, 2008, 16(5): 493—503.
- [4] Singh H, Ming-Hau Lee, Guangming Lu, et al. MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications [J]. IEEE Transactions on Computers, 2000, 49(5): 465—481.
- [5] Mei B, Vernalde S, Verkest D, et al. ADRES: an architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix[J]. Field Programmable Logic and Application, 2003: 61—70.
- [6] Mei B, Berekovic M, Mignolet, J-Y. ADRES & DRES: architecture and compiler for coarse-grain reconfigurable processors[M]. Berlin: springer, 2008: 255—297.
- [7] XPP-III Processor Overview White Paper [EB/OL]. [2012-10-28]. <http://www.pactxpp.com>.
- [8] Cardoso J, Weinhardt M. XPP-VC: A C compiler with temporal partitioning for the PACT-XPP architecture[C]// Field-Programmable Logic and Applications; Reconfigurable Computing Is Going Mainstream Berlin Heidelberg: Springer-Verlag, 2002: 207—226.
- [9] Callahan T J, Hauser J R, Wawrzynek J. The garp architecture and c compiler[J]. Computer, 2000, 33(4): 62—69.
- [10] Yoonjin Kim, Mahapatra R N. Dynamic context compression for low-power coarse-grained reconfigurable Architecture[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2010, 18(1): 15—28.

作者简介:

栗念龙 男, (1988—), 硕士研究生。研究方向为可重构计算。

唐世卓 男, (1988—), 硕士研究生。研究方向为可重构计算。

何卫锋 男, (1976—), 博士, 副研究员。研究方向为计算机体系结构与微体系结构、SOC 设计方法学与系统集成。