

一种针对可重构处理器流水线简化编程的设计范式

周君宇, 赵仲元, 绳伟光, 何卫锋

(上海交通大学 微电子学院, 上海 200240)

摘 要: 提出了一种面向可重构处理器流水线的设计范式, 并在 clang 编译架构的基础上添加新的语法, 使得用新语法编写流水线程序的时候, 能把代码量压缩为原来的 10%~20%, 并且让代码结构更加清晰和便于理解.

关键词: 可重构处理器; 流水线; 设计范式

中图分类号: TP39

文献标识码: A

文章编号: 1000-7180(2016)09-0006-04

A Design Pattern Targeting to Simplifying Pipeline Design Used on Course-Grained Reconfigurable Processor

ZHOU Jun-yu, ZHAO Zhong-yuan, SHENG Wei-guang, HE Wei-feng

(School of Microelectronics, Shanghai Jiao Tong University, Shanghai 200240, China)

Abstract: This paper propose a new design pattern for simplifying the pipeline design used on Course-Grained Reconfigurable Processor and develop a new compiler based on the clang system. Experiment results show that the amount of code decrease to 10%~20% after applying this designed pattern. Futher more, the new design pattern makes the code clearer and more comprehensive.

Key words: reconfigurable processor; pipeline; design pattern

1 引言

可重构处理器^[1]和传统通用处理器相比, 在处理计算密集和数据密集应用的时候有着明显的高效性, 在对不同逻辑处理时又不失灵活性. 可重构处理器的主要优势在于核处理器能异步调度协处理器, 在协处理器之内和协处理器之间进行不同层次的并行优化, 加快处理的速度.

流水线处理^[2]是一种很常见的并行计算范式, 通过划分流水线使任务分解为多个小的步骤, 使协处理器不间断的运行起来, 提高对数据处理的效率. 将流水线应用在可重构处理器的逻辑计算中, 会大大增加对硬件资源的利用率, 更好的发挥出可重构处理器的计算潜能.

本文针对异构粗粒度可重构处理器 GReP (General Reconfigurable Processor)^[3]设计一种针对流水线的简化编程的范式, 在编译器层面针对

Ping-Pong 流水^[4]的方式实现一个特殊的编程规范, 使得代码量大幅缩减的同时代码逻辑变得更加容易编写和理解. 这种设计范式不仅仅适用于该粗粒度可重构处理器, 还可以扩展到其他并行计算的范式设计上.

2 可重构处理器架构

本文描述的流水线处理设计范式针对于特定的可重构处理器, 但是其思想可以拓展到类似结构的多核并行处理架构上. 本部分将简单介绍该可重构处理器的硬件架构和 Ping-Pong 流水的实现方式, 以便理解后文所描述的代码简化方式.

2.1 硬件架构

该可重构处理器的可重构处理单元 (Reconfigurable Process Unit, RPU) 由一个 ARM7 的协控制器、四个可重构处理阵列 (Process Element Array, PEA)、四个共享内存 (Shared Memory, SM)

组成. 每个 SM 有 16 个 bank(图中用 4 组 bank 表示), 每个 PEA 有 16 个 PE(Process Element). SM 和 PEA 之间通过一个环状结构实现互联, 这样可以有效的降低连线复杂度.

如图 1 所示, 每个 SM 可以被两个 PEA 访问, 每个 PEA 可以访问到两个 SM, 这样一组数据可以在这个环状结构中流动.

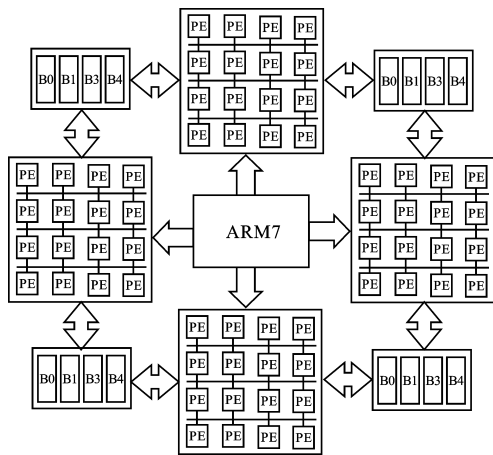


图 1 可重构处理器架构示意图

2.2 Ping-Pong 流水

流水线中生产者产生的新的数据必须等到消费者将上一拍的数据读取完毕才能写入. 由于这里数据流程图(Data Flow Graph, DFG)的粒度较大, 基本单元是一组数据/一套配置包, 因此依赖编程/编译时静态分析避免数据的读写冲突很困难. 为了避免两个 PEA 访问一个 SM 时发生读写冲突, 在此引入 Ping-Pong Buffer 机制.

为了实现 Ping-Pong 的机制, 需要将 SM 人工的划分为大小相同的两部分, 使得在 SM 的一部分数据被读取用作下一个 PEA 的输入的同时, 另一部分在写入上一个 PEA 的运算结果. 以流水线满载运行时 PEA1 及相关联的两个 SM 为例.

如图 2 所示, Ping-Pong 流水线奇数拍时, PEA1 从 SM1 虚线部分读取数据, 并将输出写入到 SM2 的虚线部分里; 与此同时, PEA0 把运算结果写入到 SM1 的实线部分, PEA2 从 SM2 的实线部分读取数据.

如图 3 所示, Ping-Pong 流水线偶数拍时, PEA1 会从 SM1 的虚线部分读取上一拍 PEA0 写入的数据, 并输出到 SM2 的虚线部分里; 与此同时, PEA0 把运算结果写入到 SM1 的实线部分, PEA2 从 SM2 的实线部分读取上一拍 PEA1 输出的数据.

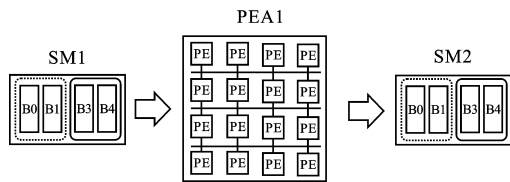


图 2 Ping-Pong 流水奇数拍

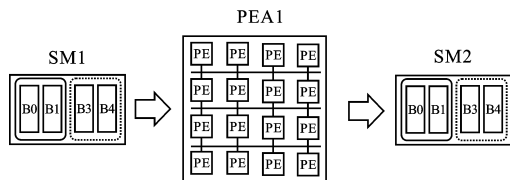


图 3 Ping-Pong 流水偶数拍

3 简单流水线编程规范的设计

为了实现 ARM7 对 PEA 和 SM 的调度和控制, GReP 提供了两个编译器调用的隐式 API 函数和三个程序员调用的显式 API 函数. 我们将利用这五个系统 API 函数, 和一个新的编程语法, 来完成可重构处理器的并行处理功能.

3.1 隐式 API 函数

(1) _setMemory(int)

该 API 的作用是设置 SM 内存的访问模式. 参数为 0 表示启用正常模式使用全部的 SM 内存空间, 1 表示 PEA i 对 bank 0—7 的访问依旧落在 SM i 中, 但其对 bank 8—15 的访问被映射到下一个 SM(即 $SM(i+1) \% 4$)的 bank 0—7; 2 表示 PEA i 对 bank 0—7 的访问映射到 SM i 的 bank 8—15, PEA i 对 bank 8—15 的访问被映射到 $SM(i+1) \% 4$ 的 bank 8—15.

(2) _syncCoproprocessor(bool, bool, bool, bool)

该 API 的作用是同步 4 个 PEA 的操作. 第 k 个实参布尔值指代是否需要等待编号为 k 的 PEA 执行完毕. 该函数将阻塞 ARM7, 等待指定的 PEA 和所有的 shared memory 的操作执行完毕.

3.2 显式 API 函数

对于显式 API 函数, 能访问到的 SM 的范围由 _setMemory(int) 决定, 参数中与 SM 有关的地址均为相对偏移.

(1) __copyinSharedMemory(int, char *, int, int)

该 API 的作用是将数据从主存拷贝到 SM 中. 第一个参数表示选择第几个 SM, 第二个参数表示数据在主存的位置, 第三个参数表示拷往 SM 的起

始位置,第四个参数表示拷贝数据的数量.

(2) __copyoutSharedMemory(int, char *, int, int)

该 API 的作用是将数据从 SM 拷贝到主存中. 第一个参数表示选择第几个 SM,第二个参数表示数据拷贝往主存的位置,第三个参数表示数据在 SM 的起始位置,第四个参数表示拷贝数据的数量.

(3) __callCoproprocessor(int, char *)

该 API 的作用是将配置包载入 PEA 并调用 PEA 执行配置包的内容. 第一个参数表示 PEA 的编号,第二个参数表示配置包的名字.

3.3 新增的编程语法

增加一个新的编程语法:

```
for_stream(inter_num){ ... }
```

这个语法用于描述一段具有上述 Ping-Pong 流水行为的任务. 解释器遇到该语法后将花括号内的代码进行展开.

在此以调用两个 PEA 计算为例,说明如何使用这个新的语法,如图 4 所示. 在这个计算中,首先需要从主存把数据拷贝到 SM 中,然后调用 PEA0 从 SM 中读取数据,计算完毕后输出给 PEA1 作为输入,PEA1 计算完后输出到 SM 中,最后调用 API 函数将结果从 SM 拷回主存. 这个过程中数据从主存 DDR 开始,依次经过 SM0、PEA0、PEA1、SM2,然后回到主存.

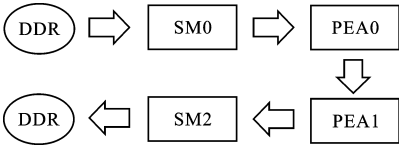


图 4 数据流通示意图

依照上述过程,可以得到显示 API 的调用顺序,伪代码如下所示. 其中 N 为总数据量与每次处理的数据量的比值.

```
for_stream(N){
    __copyinSharedMemory; //DDR->SM0
    __callCoproprocessor; //call PEA0
    __callCoproprocessor; //call PEA1
    __copyoutSharedMemory; //SM2->DDR
}
```

代码 for_stream 使用样例

4 简单流水线语法的编译实现

按照普通的流水线编写形式,代码分为流水线填充阶段 (prelude),流水线满载运行阶段,流水线

收尾阶段 (finale) 三部分. 流水线满载运行阶段分为奇数拍部分和偶数拍两部分. 填充阶段和收尾阶段需要根据 API 函数的调用顺序进行合理的分级,来决定需要多少拍填满流水线和收尾结束. 分级的数量依据表 1 来决定,表内数据代表横排 API 之后跟竖列 API 之后是否需要分级,1 代表分级,0 代表不分级,x 代表不允许此情况发生.

表 1 流水线 API 调用分级决策表

	copyin	copyout	call
copyin	0	x	1
copyout	x	0	0
call	0	1	1

这样,编译器根据 for_stream 表达式进行以下展开的操作. 展开后伪代码如下.

(1)根据 pipeline body 中 API 的调用情况分析分级情况和级数 stage_num,并得到循环执行次数 N .

(2)根据循环执行次数 N 以及级数 stage_num 生成流水初始部分 prelude,流水循环体和流水结尾部分 finale. 目前我们先假设循环体中 callCoprocesor 函数不大于 4(即 PEA 的个数). Prelude(finale) 有 stage_num 个 Phase,各个 Phase 中操作的数量依次递增(递减). 这里的“操作”指的是分级后其中一级的 API 调用. 流水循环体将 for_stream 的循环体复制两次分别作为 even phase 和 odd phase,然后该循环体会执行 $(N - \text{stage_num}) / 2$ 次.

程序员需要确保各个“操作”所需时间不要相差太大,尤其是各个 callCoproprocessor 的执行时间.

(3)对 Prelude, finale 和流水循环体的代码进行修改. 首先在每个 Phase 开头增加 _setMemory(k)函数,参数根据 Phase 数的奇偶性选择 1 或 2. 然后根据该 Phase 内使能的 PEA 在各个 Phase 的结尾增加 _syncCoproprocessor (bool, bool, bool, bool)函数,第 k 个实参布尔值指代是否需要等待编号为 k 的 PEA 执行完毕. 该函数将阻塞 ARM7,等待指定的 PEA 和所有的 shared memory 的操作执行完毕. 最后对 _copyin/_copyoutSharedMemory 的第二、三个实参进行调整,因为 ARM7 对 Shared Memory 的访问不会因为 _setMemory 的变化自动转换. 在偶数次的 Phase 的 _copyinSharedMemory 的第三个实参(即 SM 的目标地址)加 512. 对于 _copyout,在倒数第 i 个 Phase,若 i 为偶数,则 _copyoutSharedMemory 的第三个实参(即 SM 的目标地

址)减 512. 然后第二个实参,即 DDR 中的地址,需要根据 Phase 的计数和 for_stream 的 index 进行调整.

(4)最后在生成代码的结尾使用_setMemory(0)对编址模式进行复原.

对 for_stream 表达式展开后,代码结构如下所示.

```
prelude
for(phase=stage_num; phase<N; phase++){
    even phase
    if(phase++≥N) break;
    odd phase
}
finale
```

代码 for_stream 展开后结构

5 实验结果

基于本文提出的针对可重构处理器的流水线设计范式,在 clang^[5] 编译环境的基础上,对 c99^[6] 的语法进行了扩展,增加了对新语法的支持,当遇到这种语法的时候会自动进行代码展开,完成对流水线的控制. 针对分别使用 2、3、4 个 PEA 时流水线的典型运作方式进行代码编写比较,实验结果如表 2 所示.

表 2 应用范式前后代码量比较

流水线级数	2	3	4
应用前	41	55	71
应用后	5	6	7
代码缩减率/%	87.8	89.1	90.1

6 结束语

本文简单介绍了粗粒度可重构处理器的系统架构与在此架构之上 Ping-Pong 流水的实现思路,并基于此架构创新性地提出了一种简化编程的设计范式. 在用此处理器做 2—4 级 Ping-Pong 流水的情况下,使代码量压缩为原来的 10%~20%,并且使代码结构更加清晰和便于理解. 这种设计范式不仅仅适用于该粗粒度可重构处理器,对于其他可以用来并行计算的处理器也有很强的借鉴意义. 目前该设

计范式受到硬件中 PEA 数量的限制,在本实验中 GReP 只有 4 个 PEA,应用 for_stream 语法时最多只能包含 4 个调用 PEA 的接口调用函数. 下一步的研究可以向突破硬件 PEA 数这一限制进行,使得在计算核心不足的情况下仍然能顺利完成流水过程.

参考文献:

[1] Mirsky E, DeHon A. MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources[C]//FPGAs for Custom Computing Machines, 1996. IEEE Symposium on. Napa Vally, IEEE, 1996: 157-166.

[2] Gordon M L, Thies W, Amarasinghe S. Exploiting coarse-grained task, data, and pipeline parallelism in stream programs[J]. ACM SIGOPS Operating Systems Review, 2006, 40(5): 151-162.

[3] 楼杰超, 绳伟光, 何卫锋, 等. 异构粗粒度可重构处理器的自动任务编译器框架设计[J]. 微电子学与计算机, 2015, 32(8): 23-25.

[4] Dieffenderfer J N, Kalla R N. Ping-pong data buffer for transferring data from one data bus to another data bus: US. Patent 5,224,213[P]. 1993-06-29.

[5] Lattner C. LLVM and Clang: Next generation compiler technology[C]//The BSD Conference. Canada, Ottawa, 2008: 1-2.

[6] International organization for standardization, ISO/IEC 9899:1999, Programming Languages—C[S]. Geneva, Switzerland, ISO, 1999.

作者简介:

周君宇 男, (1992-), 硕士研究生. 研究方向为粗粒度可重构处理器的后端编译技术.

E-mail: junyuzhou@outlook. com.

赵仲元 男, (1990-), 博士研究生. 研究方向为可重构计算的编译架构.

绳伟光 男, (1977-), 博士, 且理研究员. 研究方向为可重构计算系统及其编译技术、高可靠集成电路设计、VLSI 电子设计自动化.

何卫锋 男, (1976-), 副教授. 研究方向为视频信号处理的 VLSI 结构、低功耗可重构多核处理器、计算机体系结构、SoC 设计与系统集成.