

多层异构粗粒度可重构处理器的编译器后端设计

刘毅超, 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: 基于 LLVM 编译器平台, 针对一种多层异构粗粒度可重构处理器, 设计了完整的任务编译器后端模块. 针对这种处理器独特的访存机制, 提出了一种新的中间数据结构 ConfigIR, 使得编译器可以完成计算密集型任务在可重构阵列上的调度, 最终生成并行配置信息. 对相关计算密集型任务的性能进行测试, 结果表明相对于串行执行, 该编译器管理下的可重构架构可以获得 2~3 倍的加速.

关键词: 可重构处理器; 编译器后端; 配置信息

中图分类号: TP39

文献标识码: A

文章编号: 1000-7180(2016)08-0015-04

A Compiler Back-end for Multi-layer Heterogeneous Coarse-Grained Reconfigurable Architectures

LIU Yi-chao, ZHAO Zhong-yuan, SHENG Wei-guang, HE Wei-feng

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

Abstract: In this paper, we present a novel task compiler back-end based on LLVM compiler framework but targeting a kind of multi-layer heterogeneous CGRA. Aiming at the processor which has distinct way accessing memory, we propose a new data structure called ConfigIR, which can analyze computation intensive programs, schedule parallel tasks and generate configuration codes eventually. Experiment shows that several computation intensive algorithms can get twice or three times speed up than serial processing.

Key words: reconfigurable processor; compiler back-end; configuration information

1 引言

可重构处理架构是一种全新的计算架构, 它结合了通用处理器(General purpose processor, GPP)的普适性灵活性和专用集成电路^[1](Application specific integrated circuits, ASIC)的高性能特点. 通常粗粒度可重构处理器包含一个通用处理核心和若干个可重构处理单元(Reconfigurable processing units, RPU)^[2], 这种结构也成为双层异构. 在一个强大的任务编译器管理下, 粗粒度可重构处理器在计算密集型任务处理方面可以发挥出强大的潜能. 每当运行一个并行任务, 编译器可以自动检测可并行的任务模块——称之为“热点”, 并且将它们调度到不同的 RPU 上并行执行, 从而达到加速整个程序的效果. 文献[3]介绍了一种可以直接将 C 的高级

语言代码编译到可重构处理单元上进行并行计算的编译器架构, 不需要程序员预知具体的寄存器级架构; 文献[4]提出了一种可在程序运行时翻译二进制指令的编译器架构, 即所谓的“运行时编译”; 文献[5]面向 KAHKRISMA 可重构架构开发了一套支持混合指令集的编译器后端, 但是这些编译器只能面向传统的双层异构可重构处理器, 如果处理器架构延伸为多层异构结构, 这些编译器架构就不再适用.

研究面向多层异构粗粒度可重构处理器, 这种处理器除了包含一个通用处理核心和若干个可重构运算阵列外, 还在可重构运算阵列外耦合了一个负责搬运配置包的协处理器. 针对这种处理器, 编译器前端负责分析原始的高级语言代码并将可并行部分作相应标记, 生成中间代码表示(IR). 编译器后端负责将 IR 表示翻译成二进制机器代码并将它们以

子任务为单位调度到不同的运算阵列上. 对于已有的可重构处理器的编译器, 在实现指令调度时存在极大的局限性, 只能机械地进行分析和调度. 对此, 引入了一种全新的数据结构 ConfigIR 作为配置信息存储的媒质, 通过维护 ConfigIR 完成指令调度、配置包打包等工作, 最终生成二进制代码.

2 可重构处理器架构

图 1 是本研究面向的多层异构可重构处理器硬件架构示意图. 该架构主要由主控核 ARM11、主存 DDR 和多个可重构处理单元(RPU)构成, 它们之间通过总线进行通信. 每个 RPU 内部除了包含 4 个可重构运算阵列(PEA)外, 还包含 1 个协处理器 ARM7 和 4 个共享内存(Shared Memory, SM), 其中 ARM7 负责配置包和数据包的搬运, SM 是存储数据包的载体. 整个可重构系统由主控制器、协控制器和可重构处理单元三个异构结构耦合而成. 程序运行时, ARM11 负责执行串行代码, ARM7 负责将数据包和编译器处理后产生的并行代码配置包搬运到 RPU 中, 其中数据包存储在 SM, PEA 读取配置包内的运算指令进行并行运算. 这种多层异构结构的优势在于, 在处理并行任务时可以通过预处理的方式隐藏配置包和数据包搬运产生的延迟, 从而加快代码的整体运行速度.

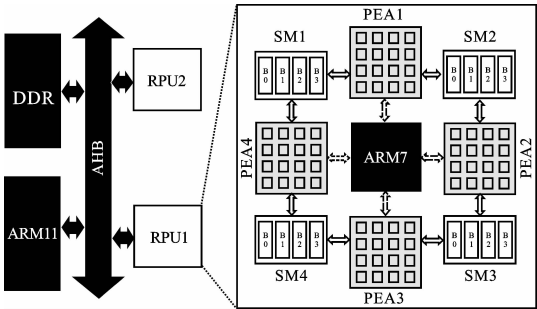


图 1 可重构处理器架构示意图

3 编译器后端框架

编译器架构基于 LLVM 编译器框架^[6]实现. 编译器前端将原始的高级语言源代码转换成类汇编形式的中间表示(Intermediate Representation, IR), 同时负责标记可并行的子任务, 通过标记协调各个 RPU 之间的同步关系. 编译器后端接收 IR 作为输入, 经过一系列的工序对 IR 进行优化、提取和翻译处理, 生成二进制配置包作为输出, 每一道工序称为一个 Pass. 在原始的 LLVM 框架中, 加入了四个全

新的 Pass, 以适应多层异构的处理器架构, 最终的编译器后端流程图如图 2 所示.

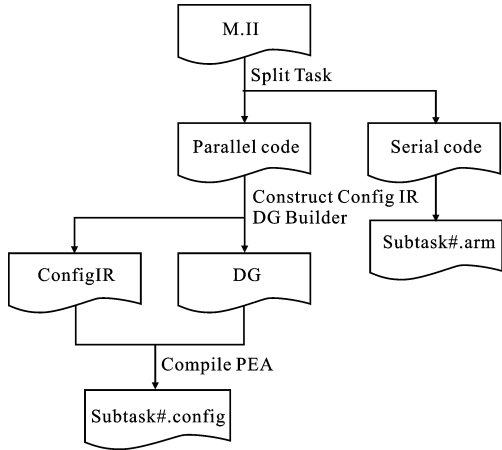


图 2 后端流程框架

图 2 中, 原始 IR 首先经过“SplitTask” Pass 的处理进行串、并代码的分割. 具体地, 利用 IR 中名为的 metadata 数据结构, 将串行代码和并行代码做不同的标记, 将可并行的待处理的循环体代码从原始 IR 中剥离出来, 生成一个个待处理的代码块, 称为 kernel. 每个 kernel 记录了一个需要可重构计算单元进行运算的详细代码内容.

对每个 kernel, “ConstructConfigIR” Pass 负责提取参与循环运算的相关信息例如循环变量、循环深度、具体运算指令等. 之后, 编译器为每条运算指令(包括基本的算术运算指令、逻辑运算指令、跳转指令和访存指令)创建一个存储结构, 称为 ConfigIR. 而利用这个结构可以将中间语言 IR 变化成为可供编译器调度的详细指令, 并对其主要内容进行分析和简化. 从形式上来看, ConfigIR 是最终二进制配置信息的可读性表示形式, 从宏观功能上来看, ConfigIR 提供了可重构处理器硬件架构和任务编译器独立并行开发的平台.

利用“ConstructConfigIR” Pass 提取到的指令信息和 IR 指令间的相互调用关系, “DGBuilder” Pass 绘制指令间的依赖关系图(Dependence Graph, DG), 为之后的指令并行化调度做准备.

最后, 利用 DG 和 ConfigIR 中的运算指令、访存信息, 编译器调用相关调度算法^[7]为每条 ConfigIR 配置信息选定执行的 PE 序号和执行时间, 完成空间和时间上的调度. 值得注意的是, 在我们的处理器架构中, ARM7 协处理器会预先将待运算的数据包搬运到 RPU 内部的 SM 中, 运算访存不需要再越过总线访问 DDR, 因此 PE 运算时的访存操作可

以在与其他算术操作同数量级的时钟周期内完成,因此访存指令可以像普通的运算指令一样进行调度。

4 ConfigIR 维护细节

ConfigIR 是存储配置指令的载体,也是指令调度的基本单位。每条 ConfigIR 中的数据分为三个数据域存储,分别为指令域,迭代域和配置域,如图 3 所示。

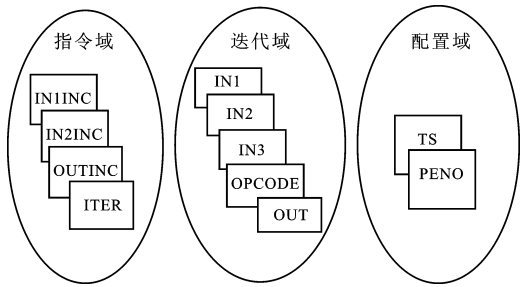


图 3 ConfigIR 基本结构

(1)指令域

指令域存储具体运算指令的信息,包括算术运算、逻辑运算的两个输入操作数、输出操作数和操作符。特别地,对于选择运算、移位运算,需要额外一个输入操作数,因此指令域中共有五个字段。

由于参与配置的指令绝大多数都是源代码循环体中重复执行的指令,配置信息具备很强的迭代性。再考虑到输入输出字段的数据来源多为 SM 中存储的数组数据,因此 IN1、IN2 及 OUT 字段中填入的数据多是参与运算的数组变量在 SM 中存储的首地址。这个地址信息在原始 IR 中并不存在,因此我们采用添加空函数、传递参数的方式将初始地址传递到 IR 中。

(2)迭代域

参与配置的指令绝大多数都是源代码循环体中重复执行的指令,为了压缩配置包,使用迭代域记录指令迭代的相关信息。迭代域用三个数据域分别记录两个输入操作数和一个输出操作数可能存在的迭代增量,即每次执行该指令时访存增加的地址量。另外,设置 ITER 字段,记录当前指令的迭代次数。

迭代次数的计算直接从 IR 中提取相关循环的循环次数信息即可。迭代增量的计算比较复杂,需要综合考虑参与运算数组的大小、维度、数组在 SM 中存储的方式等因素。考虑一个三维数组的典型情形:

假设数组变量表示为 $A[a_{1i}+a_{2j}+a_{3k}+a_4][b_{1i}+b_{2j}+b_{3j}+b_4][c_{1i}+c_{2j}+c_{3k}+c_4]$,三个维度的大小分别为 RangeI、RangeJ 和 RangeK,数组元素在

SM 中顺序排列,每次迭代时只有最后一维的循环变量递增 1,那么地址迭代增量

$$\Delta address = a_3 \times RangeJ \times RangeK + b_3 \times RangeK + c_3.$$

(3)配置域

配置域顾名思义,是记录当前 ConfigIR 参与具体调度时信息的数据域。配置域中,TS 字段记录当前 ConfigIR 首次参与运算的时间节点,PEN0 字段记录 PEA 中执行该指令的 PE 编号。这两个字段的信息均需等到指令调度算法执行完后方可填入。

综上所述,ConfigIR 中各数据域、各个字段的信息并不是一次性填入,而是随着编译器后端各个 Pass 的执行,依次完善的。ConfigIR 的存在时整个后端流程中的重中之重,将可重构阵列硬件开发与相应编译器软件开发隔离开来,使得二者可以并行设计、互不影响,提高了软硬件协同设计的开发效率。

5 实验结果

为了验证整个编译器后端流程的可行性,测试了几个可高度并行化的计算密集型程序,得到了表征其并行化的参量,并且对于能够调度的指令数,将本文的结构和模调度算法的理论最大值进行了比对。在模调度算法中,初始迭代间隔(Initial Interval)表征了算法的可并行度,越小的 II 值表征该算法的并行度越高。在表 1 中给出了几个算法的 II、MII(理论 II 的上界)和能够调度的指令节点数量。

表 1 部分算法的调度结果

算法	MII	II	原始算法 调度节点	ConfigIR 算 法调度节点.
DCT	2	2	18	20
FFT	6	7	84	92
GAOSI	2	2	20	22
GEMM	2	2	18	20
IDCT	2	2	18	20
MD5	2	2	24	24
MONTECARLO	2	2	27	27

从表 1 可以看到除了 FFT 算法,其他算法的 MII 值与 II 值均相等,这说明这些算法在本文的编译器架构下均能完成较理想的调度。而对于 70% 以上的算法,本文的结构能够使得模调度算法的调度节点数得到提升,增加了并行程度。

同样的,分别在可重构架构和单核处理器架构下进行了实验。选取了若干 EEMBC 标准测试算法

的例子来进行检验,如表 2 中的结果,可以明显的看出,利用该流程,能够将典型的计算密集型算法的运算速度提高 2~4 倍左右,而对于可重构平台,理论上能够实现的加速比在 10 倍左右,但这仅仅是理论数字,在实际处理的过程中,考虑到数据搬运的大量时间耗费,所得到的加速比已经几乎达到了手工配置算法得到的加速比.

表 2 可重构处理器与单核处理器的性能比较

算法名称	可重构处理器性能	单核处理器性能	加速比
IDCT	28 765	64 773	2. 25
FIR	188 394	429 866	2. 28
矩阵乘	3 489	10 864	3. 11

6 结束语

本文面向多层异构粗粒度可重构处理器,提出了一种全新的编译器后端架构. 该架构在 LLVM 编译器框架的基础上,添加和更改了四个关键步骤,使其能够适应处理器的多层异构特点. 创新性地提出了一种全新的数据结构 ConfigIR 以储存配置指令的相关信息,在这个基础上,指令依赖图的构建和指令调度均以 ConfigIR 为基本单位进行. ConfigIR 结构有效地隔离了可重构处理器硬件架构和编译器软件,使得二者的开发可以相互独立、互不干扰,提高了软硬件并行开发的效率. 针对若干计算密集型算法的测试表明,该编译器架构可以获得较理想的并行度和相对于串行计算较高的性能加速比. 值得注意的是,目前的编译器架构仅能支持循环和分支关系较简单的高度计算密集型程序运行,因此接下来的研究可朝进一步分析 IR 中的循环结构、分支关系方向继续努力,使得整套软硬件系统可以支持更多的不规则循环体程序和多分支结构程序.

参考文献:

[1] Zhu Min, Liu Leibo, Yin Shouyi, et al. A cycle-accurate simulator for a reconfigurable multi-media [J]. IEICE

Trans. on Inf. & Syst., 2010,93(12): 3202-3210.
[2] Cardoso João M P, Hübner Michael. Reconfigurable computing: from FPGAs to hardware/software code-sign[M]. Berlin, Springer, 2011.
[3] Yin Chongyong. Compiler framework for reconfigurable computing system[C]// ICCAS 2009. International Conference on. USA, California, IEEE, 2009.
[4] Stripf Timo. A compiler back-end for reconfigurable, mixed-ISA processors with clustered register files [C]// Parallel and distributed processing symposium workshops &. PhD forum (IPDPSW), 2012 IEEE 26th International. China, Shanghai, IEEE, 2012.
[5] Koenig R, Bauer L, Stripf T, et al. KAHRISMA: a novel hypermorphic reconfigurable instruction-set multi-grained-array architecture [C] // Design, Automation Test in Europe Conf. Exhibition (DATE). Germany, 2010: 819-824.
[6] Lattner C, Adve V. LLVM: a compilation framework for lifelong program analysis & transformation[C]// Proc. of Int’l Symposium on Code Generation and Optimization. USA, San Jose, 2004.
[7] Mei B, Vernalde S, Verkest D, et al. Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling [J]. Computers and Digital Techniques, 2003,150(5):255-261.

作者简介:

刘毅超 男, (1990-), 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: liuyc. 19@163. 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-), 硕士, 助理研究员. 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.

(上接第 14 页)

[5] 吴灏, 肖吉阳, 范红旗, 等. TMS320C6678 多核 DSP 的核间通信方法[J]. 电子技术应用, 2012, 38(9): 11-13.
[6] 朱良辰, 胡越黎, 冉峰. 高速 Mcu 核中并行优先级中断系统的设计与实现 [J]. 微电子学, 2004, 34 (4): 483-485.
[7] 胡永华, 高明伦, 王锐. 微处理器中中断电路的高层设

计[J]. 微电子学与计算机, 2001, 18(4): 1-4.

作者简介:

张跃玲 女, (1986-), 硕士, 设计师. 研究方向为大规模 SOC 及数字集成电路设计.
E-mial: lynnzhanged@yeah. net.