

针对粗粒度可重构处理器的通用循环编译技术

金 晟, 赵仲元, 绳伟光, 何卫锋

(上海交通大学 电子信息与电气工程学院, 上海 200240)

摘 要: 粗粒度可重构处理器结合了高性能和高灵活性的优点, 但是现有编译技术难以处理不定长循环和非完美循环. 本文分别针对不定长循环和非完美循环提出了高效的编译技术, 使得编译器可以处理通用形式的循环且均可以充分利用可重构处理器的计算能力.

关键词: 可重构处理器; 不定长循环; 非完美循环; 编译技术

中图分类号: TP314

文献标识码: A

文章编号: 1000—7180(2017)07—0042—04

A General Loop-Compiling Technique for Course-Grained Reconfigurable Processor

JIN Sheng, ZHAO Zhong-yuan, SHENG Wei-guang, HE Wei-feng

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

Abstract: Coarse-grained reconfigurable processor combines the advantages of high performance and high flexibility, but the existing compiler technology is difficult to deal with variable-length loops and imperfect loops. In this paper, we propose an efficient compilation technique for variable-length loops and imperfect loops, so that the compiler can deal with the general form of loops and can make full use of the reconfigurable processor computing power.

Key words: reconfigurable processor; variable-length loop; imperfect loop; compilation technology

1 引言

粗粒度可重构处理器^[1]是一种新的处理器结构, 其结合了通用处理器的灵活性和专用集成电路的高性能的优点. 粗粒度可重构处理器通常由一个通用处理器和多个可重构处理单元组成, 其中核心的运算部分会由可重构运算单元通过并行执行的方式来提高运算性能.

这种异构的并行计算结构对编译器提出了很高的要求. 文献[2]提出了一种用于可重构处理器的编译器框架, 使得程序员无需了解处理器的物理层信息. 程序编译过程中的映射效果会对程序的性能产生很大的影响, REGIMap^[3]和 EPIMap^[4]分别提出了两种使用寄存器和满射来改善映射结果的方法, 提高了产生的配置信息的新能. 但是已有的编译技

术大多针对于结构较为简单的定长完美循环. 对于非完美循环, 文献[5-6]提出了一种基于循环展平的技术, 但是该方法会增加内层循环的算子而影响性能且需要硬件的额外支持. 文献[7]提出了一个可以同时开发内外层循环并行性的技术但是无法用于不定长循环. 对于不定长循环, 循环控制一般会交由通用处理器来进行控制而使得性能大幅下降.

本文针对通用形式的循环提出了一个有效的编译技术, 使得其可以处理不定长循环和非完美循环且编译出的程序可以达到和完美循环相当的性能.

2 可重构处理器架构

图 1 展示了本论文所采用的可重构处理器的硬件架构, 该架构主要包含了 1 个主处理器 ARM11、1 块 DDR 内存和多个可重构处理单元(Reconfigu-

nable Process Unit, RPU). 每个 RPU 又包含了 1 个协处理器 ARM7、4 个可重构运算阵列 (Process Element Array, PEA) 和 4 块共享内存 (Shared Memory, SM). 作为主要运算单元的 PEA 是由 16 个运算单元 (Process Element, PE) 组成的阵列. 程序执行时 ARM11 负责任务的分配工作, ARM7 负责进行 PEA 的配置工作和部分串行代码的执行, 最后由 PEA 通过并行执行的方式来加速计算过程. PEA 在工作之前必须由 ARM7 将数据搬运至共享内存中并将对应的配置信息装载入 PEA 之中后才可以开始工作.

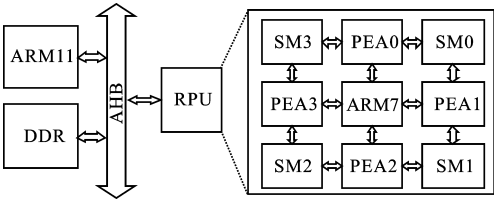


图 1 可重构处理器架构示意图

3 通用循环编译技术

循环可以根据循环次数是否是编译期已知或者循环体的数量和位置来进行分类. 根据循环次数是否在编译期已知可以分为定长循环和不定长, 根据循环体是否全部处于最内层循环内可以分为完美循环和非完美循环. 其中对于定长完美循环已有较为成熟的编译技术, 本文主要针对不定长循环和非完美循环来提出一套编译技术.

3.1 不定长循环

不定长循环可以根据循环次数是否在循环开始执行前已知分为两类.

for($i = 1; a < 10; ++i$) $a * = k;$ (1)

式(1) 展示了其中的第一类循环, 其终止条件依赖于循环体内的计算结果, 直到执行完当前循环体的内容才可以判断循环是否结束.

for($i = 1; i < k; ++i$) $++a;$ (2)

式(2) 展示了其中的第二类循环, 其中变量 k 的值由这段循环之前的代码所得, 虽然在编译时无法获得 k 的值, 但是在循环开始执行前可以确定 k 的值.

3.1.1 循环次数在循环开始前未知

由于循环的终止条件依赖于每次循环的运算结果, 所以循环控制相关的运算需要被提取出来后在循环每次循环执行时都运算一次来判断循环是否结束.

在这种情况下循环中单次表达式的内容会被置于整个循环之前执行一次, 而条件表达式和末尾循环体将会被分别置于循环体的头部和尾部来组成一个新的循环体. 为了控制循环的结束, 循环体以及条件表达式自身的循环次数会被设置成为无限循环直至条件表达式不满足.

因为采用并行执行多个循环的执行方式的原因, 同一时刻会有多个循环同时执行, 所以可能导致循环无法正确结束. 比如一个循环应当在第 k 次循环运行完成后终止, 但是由于第 $k + 1$ 次循环已经并行地开始执行且第 k 次循环的循环条件的计算完成过晚, 使得第 $k + 1$ 次循环也被执行. 为了防止这种情况, 循环条件应当被在循环体之前完成计算, 然后根据结果判断是否终止执行. 但是这种方式会使得循环条件和循环体之间变为串行执行, 降低了循环执行时的并行性, 尤其是在循环条件或者循环体的计算均较为简单时, 由于算子不足以填充的 PEA, 降低了 PE 的利用率从而影响了循环执行的性能.

为了解决这个问题在这里采用了阻止数据写回的方式来提高并行性, 即循环条件只需要在循环体中的写入操作之前完成运算即可. 然后将通过循环条件的输出结果用于控制 Store 指令的使能信号, 从而使得在循环条件不满足的情况下 Store 指令不会被执行, 最终保证了循环执行结果的正确性.

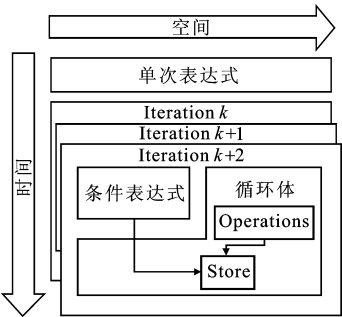


图 2 循环次数在循环开始前未知时的执行模式

3.1.2 循环次数在循环开始前可知

此类循环的循环次数由于在循环开始执行时已经确定, 所以循环条件不需要在循环每次执行时都进行运算并判断一次, 可以将这部分的工作移至编译期由编译器来负责计算从而减少循环的算子数量来提高其执行时的性能. 以代码(2) 为例, 在这种情况下编译器会分析循环控制部分的代码并分析出如式(3) 所示计算迭代次数的表达式.

$$\text{count}_{\text{iter}} = k - 1$$
 (3)

然后将该表达式置于循环开始执行之前进行运

算并将结果写入寄存器中. 这样循环在开始运行时便可以通过访问相应寄存器的值来获取循环次数, 此时通过将指令中的迭代次数字段配置为之前存放了计算所得的循环次数的寄存器即可.

通过这种方式不必每次循环中都去判断循环是否终止并等待判断结果计算完毕, 在减少了算子的数量的同时提高了计算的并行程度, 尤其是在循环控制较为复杂而循环体较为简单的情况下可以大幅提高循环的执行性能.

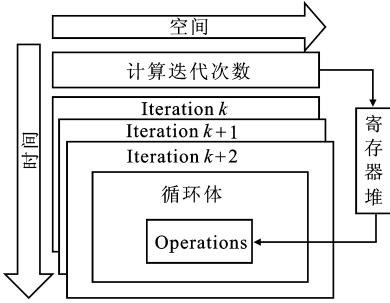


图 3 循环次数在循环开始前可知时的执行模式

3.2 非完美循环

现有编译技术都是基于对单个基本块进行的映射, 无法直接对多个基本块的情况进行处理. 在处理非完美循环时一般采用对每个基本块分别进行编译然后基本块间串行执行的方式, 与此同时还需要通用处理器来进行一部分的运算以及控制. 在这种情况下仅有最内层的基本块可以并行执行, 同时由于需要通用处理器的介入而增加了 PEA 配置和启动的开销等, 降低了 PEA 的利用率, 严重影响程序的性能.

对于非完美循环, 如图 4 所示, 本文通过将非完美循环转变成为多个完美循环的方式来进行编译和执行, 具体可以分为如下两个步骤:

(1) 循环分裂

将原有的循环分离成为多个完美循环, 其中每个循环包含原有循环中的一个基本块. 这一步会改变原有代码的执行顺序, 每个基本块的内容会被集中在一起一次性执行完毕, 而不是原有的内外层循环的基本块之间交替执行的流程.

因为处理器在运算的过程中会同时进行多个循环的计算, 所以当不同基本块间需要进行数据传递时需要为变量的每次循环都分配独立的存储空间, 从而使不同循环之间的计算不会相互干扰, 在这里总共需要 $O(\text{循环次数})$ 的额外空间.

(2) 循环重排列

由于在循环分离的过程中需要 $O(\text{循环次数})$ 的额外空间, 在循环次数较大的情况下会占用大量的空间, 所以这一步骤会对空间占用进行优化.

通过 PEA 的执行方式可知, 在同一时刻被同时并行执行的循环个数一定不会超过 PE 的个数. 进一步分析映射算法可知, 同时执行的循环个数满足如下条件:

$$L = \text{Util} * \text{Delay} * \text{Count}_{\text{PE}} / \text{Count}_{\text{op}}$$

其中 L 为同时执行的循环个数, Util 为 PEA 利用率, Delay 为输出延时, Count_{PE} 为 PE 个数, Count_{op} 为算子个数.

公式中的变量均可在编译过程中被确定, 所以在编译时便可以准确得知可以同时执行的循环个数 L .

在确定了 L 后对循环的执行顺序进行重新排列. 如图 4 所示, 将整个循环过程拆分成多组, 其中 L 次循环为一组, 每组依次执行. 在这种情况下仅需要 L 组额外的存储空间, 每组之间可以共用这一部分存储空间, 从而使得空间占用从 $O(\text{循环次数})$ 降低到了 $O(\text{PE 个数})$.

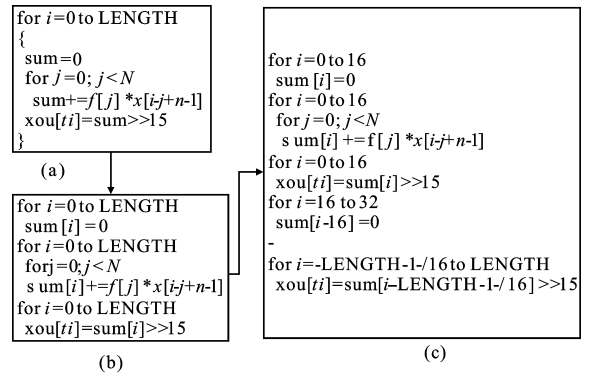


图 4 (a)原始非完美循环 (b)循环分裂后的循环 (c)循环重排列后的循环

4 实验结果

为了验证本文提出的通用循环编译技术的性能, 选取了部分包含非完美循环或者不定长循环的算法进行测试并将结果与传统的循环展平等的编译技术进行对比.

表 1 和表 2 分别显示了不同算法在两种编译方式下的算子数和最终执行耗时的比较结果. 从表 1 中可以看到对于通用形式的循环, 本编译技术可以减少有效降低有效算子的数目. 同时从表 2 可以看到得益于算子数目减少以及运算并行度的提升, 对于通用形式的循环可以获得最高 245% 的性能

提升.

表 1 不同算法的算子数对比

算法	本技术	传统技术	算子减少/%
montecarlo	27	35	22. 9
dct	15	17	11. 8
fir	5	8	37. 5
ncc	22	26	15. 4

表 2 不同算法的用时对比

算法	本技术	传统技术	加速比/%
montecarlo	3 887	9 532	245
dct	3 927	8 233	210
fir	16 578	33 936	205
ncc	1 628	2 366	145

5 结束语

本文为粗粒度可重构处理提出了一种可以处理通用形式循环的编译技术. 针对传统技术在处理非完美循环和不定长循环时算子过多和性能低下的问题,通过改进循环的执行方式和循环分离的方式,以极小的常量大小的空间为代价在减少了有效算子数目的同时提高了运算的并行度. 对于通用形式循环的支持使得粗粒度可重构处理器可以处理更为复杂的计算,大幅提高了粗粒度可重构处理器的应用范围. 在此技术下循环的执行性能主要受制于循环间的数据依赖,如何加速循环间数据依赖紧密的情况会成为后续研究的重点方向.

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. Proceedings. IEEE Symposium on. Napa Valley, California, IEEE, 1996: 157-166.

[2] Yin C, Yin S, Liu L, et al. Compiler framework for reconfigurable computing system[C] //Communica-

tions, Circuits and Systems, 2009. ICCAS 2009. International Conference on. IEEE, Milpitas, California, USA,2009: 991-995.

[3] Hamzeh M, Shrivastava A, Vrudhula S. REGIMap: register-aware application mapping on coarse-grained reconfigurable architectures (CGRAs)[C] // Proceedings of the 50th Annual Design Automation Conference. San Francisco, CA, USA ,ACM, 2013: 18.

[4] Hamzeh M, Shrivastava A, Vrudhula S. EPIMap: using epimorphism to map applications on CGRAs[C] // Proceedings of the 49th Annual Design Automation Conference. San Francisco, California, USA ,ACM, 2012: 1284-1291.

[5] Lee J, Seo S, Lee H, et al. Flattening-based mapping of imperfect loop nests for CGRAs? [C] //Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2014 International Conference on. Jersey City, New Jersey, USA,IEEE, 2014: 1-10.

[6] Sim H, Lee H, Seo S, et al. Mapping imperfect loops to coarse-grained reconfigurable architectures [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2016, 35 (7): 1092-1104.

[7] Lin X, Yin S, Liu L, et al. Exploiting parallelism of imperfect nested loops with sibling inner loops on coarse-grained reconfigurable architectures [C] // 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC). Chiba, Japan, IEEE, 2016: 456-461.

作者简介:

金 晟 男,(1992-),硕士研究生. 研究方向为粗粒度可重构处理器编译技术. E-mail:timepirate@sjtu. edu. cn.

赵钟元 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-),助理研究员. 研究方向为高可靠集成电路设计.

何卫峰 男,(1976-),副研究员. 研究方向为 SoC 设计与系统集成.