if-conversion 及其在 LLVM 上的实现

高翔

qasdfgtyuiop@gmail.com

February 26, 2014

Abstract

人们追求程序运行的性能的脚步从来都没有停歇过，为了使得程序更快地运行，人们不断的改进处理器的性能。然而，处理器的性能只是程序性能的一部分，要实现高性能，编译器生成的代码的执行效率也至关重要。更先进的构架给编译器提出了更高的要求，人们也一直在探索如何让编译器生成更高效率的代码。由于早先的编译器无法很好地处理控制依赖，Allen 等人于 1983 年提出了 if 转换（if-conversion）的概念 [1]。if-conversion 能够消除程序中的除了后向分支以外的所有分支，这样就使得程序中的控制依赖被转化为数据依赖，编译器可以在此基础上进行进一步优化。1987 年，Ferrante 等人提出了程序依赖图（PDG）的概念 [3]，使得编译器可以轻易的处理控制依赖。尽管现代的编译器可以轻易处理控制依赖，分支语句的存在有时候仍然会严重影响性能。分支语句会中断指令流水，造成严重的性能损失。程序中大约 20% 的语句是分支语句，如果一个分支语句会造成 2 个周期的延迟，那么会有大约 40% 的机器时间浪费在分支上 [2]。此外，分支语句会将程序划分为若干基本块，从而阻断了进一步的优化。If 转换可以通过现代处理器提供的条件执行指令，实现分支消除，这样做不仅可以减少分支造成的延迟，而且可以将若干小的基本块合并成一个大的基本块，进而为后续优化提供便利。由于上述原因，虽然现代编译器能够很好地处理控制依赖，if 转换仍然显得尤为重要。

We have never stopped pursuing the performance of the program, to achieve this goal, the performance of the processor has been highly developed. However, the performance of the processor is only part of the performance of the program. The performance of the code generated by the compiler is also an important factor. The modern architecture has brought greater challenges to the development of compiler. How to design a compiler that generates high performance instructions has been a problem for a long time. Early compiler couldn't process control dependency. To solve this problem, Allen et al. came up with the concept of if-conversion [1], which can remove all branches in the program except for the backward branch. The deletion of branches converts control dependency to data dependency,

making it possible for the compiler to do further optimizations. In 1987, Ferrante et al. came up with the concept of program dependency graph (PDG) [3], making it easier for compilers to handle control dependency. Albeit this improvement, if-conversion is also important since the branch can significantly harm the performance. Branch interrupts the pipeline of instructions, causing great loss in performance. According to statistics, about 20% instructions are branches in a program. Assume that one branch can cause about 2 cycles delay, then about 40% of machine time will be wasted in branch [2]. What's more, branch divides the program to several basic blocks, making it harder for further optimization. If-conversion removes branches by taking use of predicted execution mechanism of modern processor. This can not only eliminate the delay caused by branch, but also merge several small basic blocks to a large basic block, making a lot of optimizations possible. Due to the reasons stated above, if-conversion is also very important.

目录

1 分支语句与程序性能的关系

1.1 分支语句对性能的影响

在现代的高性能处理器中，分支指令是一个代价昂贵的指令 [5]。深度流水以及多发射极大的提高了处理器的性能，但是频繁出现的分支导致的控制流转移会中断指令流水线的连续性，从而降低性能。为了解决分支语句对性能的影响，现代处理器往往配置分支预测机制，即在条件分支执行前，处理器对分支方向进行预测，并按照预测的方向进行取指执行。不幸的是，分支预测经常出错，一旦预测失败就会造成非常严重的性能损失。研究显示，错误的分支预测会使得性能降低 2-10 倍 [6] [7] [4]。

分支预测失败对性能的影响主要有三个方面。首先，处理器会在预测的分支方向取出大量指令执行，一旦分支预测失败，所有执行的结果都必行被丢弃。这就意味着，预测失败时，处理器浪费了大量的指令槽来执行无用的指令。浪费的指令槽的数目与处理器的发射宽度有关，处理器发射宽度越高，浪费的指令槽的数目也就越多。其次，分支预测失败以后，处理器必须撤销预测执行对处理器的影响，处理器必选让流水线的排空，同时还要无效化处理器缓存，以避免其内容被更新到处理器状态。再次，当发现预测失败的时候，处理器必须恢复到正确的分支进行执行，这就需要计算正确的指令地址，以及初始化正确方向的取指过程。另外，由于由于分支语句大量存在于程序中，超标量处理器必须要能够在一个周期内执行多个分支。假设指令流中包含 25% 的分支，那么一个 8 发射的超标量处理器必须具有每个周期执行至少 2 个分支的能力。一个周期执行多个分支增加了流水线的复杂度。在一个高发射率的处理器中，重复算术功能单元要比预测并执行多个分支要容易得多。

除了分支预测失败造成的处罚以外，分支语句会将程序分割为若干小的基本块，指令调度无法在基本块之间进行，从而影响性能。为了解决这些问题，现代处理器提供谓词执行的机制。谓词执行指的是指令的条件执行，它给指令增加一个布尔操作数，这个布尔操作数称为谓词。如果谓词的值为真，那么这个指令被正常地执行，而如果这个谓词的值为假，则这个指令就被当成空指令来执行。

有了处理器的支持，编译器就可以通过 if-conversion 来将分支转化为谓词执行。通过分支的移除，可以消除分支预测失败的惩罚，也可以避免让宽发射处理器在一个周期内处理多条分支从而提高了性能。

1.2 分支消除与性能

既然分支语句对性能有这么大的影响，那么是否将所有的分支都用谓词执行来代替就是好的呢？答案显然是否定的。

2 if-conversion 的发展简史

if-conversion 最早由 Rice 大学的 Allen 等人在 1983 年提出 [1]。当时已有向量机的出现，并且新一代的 Fortran 也一定会提供向量操作来支持向量机。不幸的是，由于之前版本的 Fortran 中不支持向量操作，所以相当大的一部分旧有的 Fortran 程序无法充分利用处理器的新特性，这就意味着，自动向量化的研究势在必行。当时 Rice 大学的人正在开发一个翻译程序，称为并行 Fortran 转换器（Parallel Fortran Converter，简称 PFC），它能将 Fortran 66 或者 Fortran 77 代码翻译成新一代 Fortran 的代码，并且通过自动向量化来充分利用向量操作。当时的编译器进行程序变换还是基于数据依赖，分支语句产生的控制依赖不能非常好的融入到当时的体系中去。与此同时，分支语句的存在也会使得自动向量化无法进行。由于这两个原因，Allen 就提出了将程序中的所有分支语句移除而转换成谓词执行的一些算法。Allen 的算法的目的比较看重向量化，这算法是直接在源代码上进行操作的，它可以将所有的前向分支跟出口分支消除。但是 Allen 的算法没有考虑到如何最小化谓词的分配等的问题。尽管 Allen 的算法可以将分支转换为谓词执行，但是编译器要想判断是否可以进行向量化，必须先对代码进行 if-conversion，这样就将所有的控制依赖转化为数据依赖，然后才能使用基于数据依赖的算法判断是否能够进行向量化，一旦无法向量化，不单单是 if-conversion 执行的时间被浪费掉，而且 if-conversion 会对代码的执行效率产生影响1.2，糟糕的是，if-conversion 已经将代码变得面目全非无法还原。

1987 年，Ferrante 等人提出了程序依赖图（PDG）的概念 [3]。这是一个非常重要的工作，因为 PDG 是一个新的框架，在这个框架下，数据依赖跟控制依赖被统一地进行处理，一切旧有的算法都可以高效地纳入新的框架下。由于不再存在之前的编译器无法处理控制依赖的问题，也就没有必要再利用 if-conversion 将控制依赖转化为数据依赖了。同时这个框架对 if-conversion 的发展也起到了承上启下的作用，后来对 if-conversion 的研究都是在这个框架下的。

虽然不再需要用 if-conversion 将控制依赖转化为数据依赖，但是 if-conversion 并没还有因此失去它的意义。正如上一小节所说，程序中的分支语句已经称为高性能体系结构中性能提升的主要障碍，而 if-conversion 恰好可以用谓词执行来代替分支，进而提高效率。只不过问题的核心已经不是像 Allen 所说的那样，将控制依赖转化为数据依赖，也不是向量化，而是减少由于分支语句的存在对程序性能造成的影响，这也对 if-conversion 之后的代码的性能提出了要求。

3 待办事项

1. 整理参考文献，bibtex 里面现在好多条目都没有写期刊

参考

[1] John R Allen, Ken Kennedy, Carrie Porterfield, and Joe Warren. Conversion of control dependence to data dependence. In *Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 177–189. ACM, 1983.

[2] Roger A. Bringmann, Scott A. Mahlke, Richard E. Hank, John C. Gyllenhaal, and Wen mei W. Hwu. Speculative execution exception recovery using write-back suppression. In *MICRO*, pages 214–223. ACM/IEEE, 1993.

[3] J. Ferrante, K. Ottenstein, and J. Warren. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems*, 9(3):319–349, July 1987.

[4] Y. Patt M. Alsup H. Scales M.Shebanow M. Butler, T. Yeh. Single instruction stream parallelism is greater than two, 1991.

[5] Scott A. Mahlke, Richard E. Hank, Roger A. Bringmann, John C. Gyllenhaal, David M. Gallagher, and Wen mei W. Hwu. Characterizing the impact of predicated execution on branch prediction, 1994.

[6] Michael D. Smith, Mike Johnson, and Mark A. Horowitz. Limits on multiple instruction issue, 1989.

[7] David W. Wall. Limits of instruction-level parallelism, 1991.