第九-十章 中间代码优化与目标代码生成

1、何谓代码优化？进行优化所需要的基础是什么？

答：代码优化是编译时为改进目标程序的质量而进行的各项工作，包括提高空间效率和时间效率。优化的目标是改善程序的性能或其它运行时特性，而不改变程序的功能或输出结果。进行优化需要以下基础：

- 中间代码表示：优化通常在中间代码层面进行，因此一个清晰、规范且易于操作的中间代码表示形式是必需的。中间代码应该便于分析和转换。
- 依赖分析：确定代码中各个操作之间的依赖关系，包括数据依赖和控制依赖。这有助于识别可以并行化的计算，以及在不改变程序意义的前提下重新排序操作。
- 优化技术：了解和掌握各种代码优化技术，如常量传播、死代码删除、循环优化（例如循环展开、循环融合）、函数内联等。
- 目标平台特性：优化应考虑目标硬件平台的特性，如处理器的指令集、内存层次结构、缓存大小和行为、流水线结构等。这有助于做出最适合特定硬件的优化决策。
- 理解代码的基本原理和算法：了解代码的工作原理和背后的算法，可以帮助识别潜在的性能瓶颈和优化机会。
- 掌握编程语言和开发工具：熟悉所使用的编程语言和开发工具，了解其特性、语法和性能特点，可以更好地进行优化。

2、编译过程中可进行的优化如何分类？

答：按照机器相关性可以分为：

机器相关优化：寄存器优化，多处理器优化，特殊指令优化，无用指令消除等。

- 机器代码优化：机器代码优化是编译过程的后期阶段，针对具体的目标处理器架构进行优化。这些优化考虑了目标机器的指令集、寄存器分配、管线化等硬件特性，以实现更高的运行效率。

机器无关优化：中间代码优化，源程序优化。

- 中间代码优化：中间代码优化发生在编译器将源代码转换成目标代码的中间阶段。在这个阶段，代码以一种更抽象、标准化的形式存在，易于进行跨平台的优化处理。
- 源程序优化：这类优化发生在编译过程的早期阶段，通常是在源代码被转换成中间代码之前。它可能涉及到简化表达式、移除冗余代码或改变代码结构来增强后续阶段的优化效果。源代码级优化往往依赖于编程语言的特性，比如循环展开、条件语句简化等。

3、最常用的代码优化技术有哪些？

答：

1) **合并常量计算：**在编译过程中提前计算那些能够确定结果的常量表达式。例如，如果代码中出现了表达式 3+4，编译器在编译时就可以将其替换为 7。
2) **消除公共子表达式：**识别并消除在一个块中多次计算的相同表达式。例如，如果一个表达式如 x * y 在同一函数的多个地方被计算，而在这些计算之间 x 和 y 的值没有改变，那么该表达式可以只计算一次，其结果存储在一个临时变量中，之后使用这个临时变量代替原来的表达式。

3) **削减计算强度：**削减计算强度是指用成本较低的操作替换更昂贵的操作。常见的例子包括将乘法替换为加法，或者将幂运算替换为乘法。
4) **删除无用代码：**移除那些不会影响程序最终结果的代码部分。这包括那些永远不会被执行的代码（如永远不满足的条件分支内的代码），以及其结果不被使用的计算。
5) **循环不变表达式外提：**识别循环中不依赖于循环变量的计算，并将这些计算移出循环。例如，如果在一个循环体内有一个表达式依赖于不在循环中改变的变量，那么这个表达式可以在循环开始前计算一次，并在循环中复用其结果。
6) **归纳变量删除：**归纳变量是在循环中用来控制循环次数的变量，归纳变量删除是指减少或消除这些归纳变量的使用。例如，如果一个循环使用两个归纳变量，而其中一个变量是另一个变量的简单线性函数（如 i=2j），那么可以去掉一个变量，只用一个变量来控制循环。
7) **循环展开：**循环展开是一种减少循环开销的优化技术，它通过减少循环迭代次数来减少循环控制的开销。这通常通过执行多个循环体的副本来实现。
8) **内联函数：**函数内联是将函数调用替换为函数体本身的技术。这可以消除函数调用的开销，如参数传递、返回地址的处理等。内联适用于那些体积小、调用频繁的函数。但过度内联可能导致代码膨胀。
9) **分支预测优化：**编译器可以通过重排分支顺序来优化预测错误的成本，将最有可能执行的分支放在前面，或者添加预测提示。这种优化有助于减少现代 CPU 中因分支预测错误而造成的性能损失。