



wuhui_gdnt ⑤ 已于 2023-10-27 14:22:38 修改 ◎ 阅读量913 ☆ 收藏 2 👍 点赞数

文章标签: 学习 笔记 Ilvm 编译

4.1.4. DAG 合并与合法化

来自SelectionDAGBuilder的SelectionDAG输出还不能进行指令选择,必须通过额外的转换——显示在上图。在指令选择前应用的遍序列如下:

• 匹配一组节点,在有利时使用更简单的构造来替换它们,DAG合并遍优化SelectionDAG的结构。例如,(add (Register X), (constant 0))可以折叠 X)。类似的,目标机器的合并方法可以识别节点模式,决定是否合并以及折叠它们,提升目标机器指令选择的质量。方法setTargetDAGCombine 器希望合并的节点。例如,MIPS后端尝试合并加法——参考lib/Target/Mips/MipsISelLowering.cpp的setTargetDAGCombine(ISD::ADD)与 performADDCombine()。

(注:在每个合法化阶段后运行DAG合并以尽量减少SelectionDAG冗余。另外,DAG合并知道运行到遍链(pass chain)的何处(例如,在合合法化之后),并且可以更精确地使用这个信息。)

- 类型合法化遍确保指令选择仅需要处理合法的类型。合法类型是目标机器原生支持的类型。例如,在仅支持i32类型的目标机器上带有i64操作数法的。在这个情形里,类型合法器执行整数扩展,将一个i64操作数分解为两个i32操作数,同时生成处理它们的合适的代码。目标机器定义每个寄存器类相关,明确声明支持的类型。因此,必须相应地检测与处理非法类型:标量类型可以被提升、扩展或者弱化,而向量类型可以被分裂、加宽。同样,目标机器也可以定制类型合法化的方法。类型合法器运行两次,在第一次DAG合并后,以及向量合法化后。
- 存在后端直接支持一个向量类型的情形,这意味着对此有一个寄存器类,但在一个给定向量类型上的一个特定操作不一定。例如,有SSE2的XI 向量类型。不过,在ISD::OR上没有支持v4i32类型的X86指令,仅支持v2i64。因此,向量合法化器使用指令的合法类型提升或扩展来处理这些作 ISD::OR情形里,操作被提升为使用v2i64类型。

(注:对某些类型,扩展将消除向量类型,使用标量类型。这会导致目标机器不支持的标量类型。不过,后续的类型合法化器将清除之。)

• DAG合法化器具有与向量合法化器相同的任务,但处理任何带有不支持类型(标量或向量)的遗留操作。它支持相同的操作:提升,扩展,处理例如,x86节点不支持以下三个中的任意一个: i8类型有符号整数到浮点数的操作(ISD::SINT_TO_FP),要求合法化器提升该操作; i32操作数上法,要求一个扩展,发布一个库调用来处理该除法; f32操作数上浮点绝对值(ISD::FABS),使用一个定制句柄来生成具有相同效果的代码。X86来发布这样的行为(参考lib/Target/X86/X86ISelLowering.cpp):

setOperationAction(ISD::SINT_TO_FP, MVT::i8, Promote);

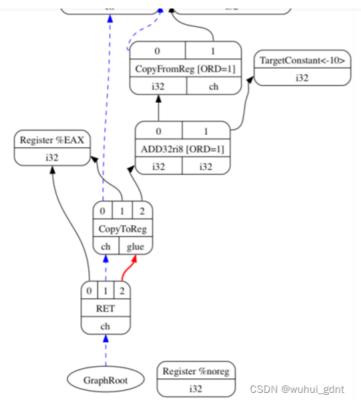
setOperationAction(ISD::SDIV, MVT::i32, Expand);

setOperationAction(ISD:: FABS , MVT::f32, Custom);

4.1.5. DAG到DAG的指令选择

DAG到DAG指令选择的目的是通过模式匹配 ,将目标机器无关的节点翻译 为目标机器特定的节点。指令选择算法是局部的,一次作用在一个Se(基本块)实例上。

作为一个例子,在指令选择后,我们的SelectionDAG结构展示如下。CopyToReg,CopyFromReg及Register节点没有触及,并一直维持到寄存器分配上,指令选择阶段甚至可能生成额外的节点。在指令选择后,ISD::ADD节点被翻译为X86指令ADD32ri8,而X86ISD::RET_FLAG被翻译为RET。



(注:在同一个DAG中可能有3种指令表示共存:通用LLVM ISD节点,比如ISD::ADD;目标机器特定的<Target>ISD节点,比如X86ISD::RET_FLAG 机器的物理指令,比如X86::ADD32ri8。)

4.1.6. 模式匹配

每个目标机器通过在名为<Target Name>DAGToDAGISel的 SelectionDAGISel子类中实现 Select()方法来处理指令选择、例如、S SparcDAGToDAGISel::Select()。这个方法接受一个要匹配的SDNode参数,返回代表一条物理指令的一个SDNode值;否则出错。

Select()方法允许两个方式来匹配物理指令。最直接的方式是通过调用从TableGen模式生成的匹配代码,就像下面列表中的第一步。不过,模式表达 足以处理某些指令的古怪行为。在这种情形下,必须在这个方法里编写定制的C++匹配逻辑的实现,就像下面列表的第二步。这个做法的细节如下:

1. Select()方法调用SelectCode。TableGen为每个目标机器生成SelectCode()方法,在这个代码里,TableGen还生成了MatcherTable,将ISD及<Tare 映射到物理指令节点。这个匹配者表从.td文件(通常是<Tarqet>InstrInfo.td)里的指令定义生成。SelectCode()方法以调用使用该目标机器匹配者表 的目标机器无关方法SelectCodeCommon()结束。TableGen有一个专用的指令选择后端来生成这些方法即这个表

\$ cd < llvm_source > / lib/Target/Sparc

\$ Ilvm-tblgen -gen-dag-isel Sparc.td -I ../../include

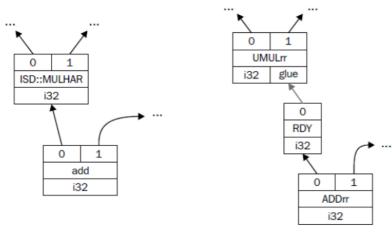
对每个目标机器,在C++生成文件 < build_dir>/lib/Target/<Target>/<Target>GenDAGISel.inc 中有相同的输出;例如,SPARC的方 <build dir>/lib/Target/Sparc/SparcGenDAGISel.inc文件里。

2. 在调用SelectCode()之前提供定制的匹配代码。例如, i32节点ISD::MULHU执行两个i32的乘法,产生一个i64结果,并返回高i32部分。在32位SF 法指令SP::UMULrr在特殊寄存器Y中返回高部分,这要求使用SP::RDY指令读出。TableGen不能体现这个逻辑,我们使用下述代码来解决之:

case ISD::MULHU: { SDValue MulLHS = N->getOperand(0); SDValue MulRHS = N->getOperand(1); SDNode *Mul = CurDAG->getMachineNode(SP::UMULrr, dl , MVT::i32, MVT::Glue, MulLHS, MulRHS); return CurDAG->SelectNodeTo(N, SP::RDY, MVT::i32, SDValue(Mul, 1));

这里,在这个上下文中,N是要匹配的SDNode实参。它等于ISD::MUI HU。因为在这个case语句之前已经执行讨完备性检查。我们着手生成 的操作码来替换ISD::MULHU。为此, 我 wuhui_gdnt (关注 < 分享





Before instruction selection

After instruction selection

4.1.7. 指令选择过程的可视化

有几个IIc选项允许在指令选择的不同阶段可视化SelectionDAG。如果你使用这些选项,IIc将生成一个类似于之前展示图的.dot图,不过你需要使用do它,或使用dotty来编辑它,两者都能在www.graphviz.org的Graphviz包里找到。下表以执行序展示了各个选项:

Llc选项	阶段
-view-dag-combine1-dags	在DAG合并1之前
-view-legalize-types-dags	在类型合法化之前
-view-dag-combine-lt-dags	类型合法化2之后,DAG合并之前
-view-legalize-dags	合法化之前
-view-dag-combine2-dags	DAG合并2之前
-view-isel-dags	在指令选择之前
-view-sched-dags	指令选择之后,调度之前

44.1.8. 快速指令选择

LLVM还支持另一个称为快速指令选择的实现。快速指令选择的目的是,以代码质量为代价,提供快速的代码生成,它适合于-O0级别优化过程的哲学归因于避免复杂的折叠与降级逻辑。TableGen描述也用于简单的操作,但指令更复杂的匹配要求目标机器特定的处理代码。

(-O0过程还使用快速的次优寄存器分配器及调度器,以代码质量换取编译速度。)

LLVM中,使用LLC生成可视化SelectionDAG

qq_27885505的†

LLVM静态<mark>编译</mark>器(**LLVM** Static Compiler,llc)是一个将**DAG**内容可视化的优秀工具。使用llc的命令行选项,在特定的执行阶段之前或之后显示Selection**DAG**。但是关于

LLVM学习笔记(53)

wuhui_gdnt的

3.11.2.模式分析 GloballSel是以<mark>DAG</mark>指令选择的TD定义处理与分析为基础的。因此,GloballSelEmitter包含了一个CodeGen<mark>DAG</mark>Patterns类型的const成员CGP(一旦创建5

Hikari-LLVM15.0.0.xctoolchain_llvm资源-CSDN文库 资源-CSDN文库

LLVM是一个开源的<mark>编译</mark>器基础设施项目,广泛用于构建<mark>编译</mark>器、工具和语言。它提供了中间表示(IR)和其他<mark>编译</mark>阶段所需的基础组件。XCToolchain通常是指Xcode使用的工

【免费】Hikari-LLVM15.0.2.zip_iosollvm资源-CSDN文库

资源浏览查阅36次。拷贝到/Applications/Xcode.app/Contents/Developer/Tooiosollvm更多下载资源、学习资料请访问CSDN文库频道.

llvm后端之SDNode设计

fs3296的†

DAG节点的类型(也就是操作类型,对应于指令类型)是定义在<mark>llvm::</mark>lSD::NodeType枚举类型中;EVT是对MVT的封装,此外还提供了对MVT类型的扩展。当表示MVT之外的

LLVM Cpu0 新后端7 第一部分 DAG调试 dot文件 Machine Pass

lml435035844的†

LLVM手动开发一个新后端的系列课程的记录分享

LLVM学习笔记(15)_llvm complexpattern

LLVM学习笔记(15) 3.4.DAG指令选择器的生成代码 3.4.1











