高层次综合 Project
项目文档

基于 LLVM 语法的 project

21307140085 秦振航 21307140085@m.fudan.edu.cn

21307140079 杨远达 yangyuanda922@gmail.com

2024 年 6 月 26 日

目录

1 任务需求分析

1.1 问题

以上述 IR 作为输入，我们提供基本的 IR 的 parser，根据上述 IR，完成调度、寄存器及操作的绑定、控制逻辑综合，函数输入的数组综合为 SRAM 存储器，需要根据 load/store 指令来读写存储器数据。最终生成 RTL 代码。基本计算操作可以调用 RTL 计算模块，或直接使用原始操作符。

1.2 需求分析

1. 实现寄存器的绑定和调度 2. 实现控制逻辑综合

2 团队分工情况

2.1 分工情况

姓名	分工
秦振航	寄存器的绑定和操作调度
杨远达	控制逻辑综合

3 程序实现思路与数据结构

3.1 代码结构

项目的源代码在 HLS/src 文件夹中，包含以下模块：

1. Paser 输入文件解析模块。
2. main 主函数。
3. Basic 基础类，包括语法块，句子，以及调度绑定时需要的点类。
4. HLSHandler 高层次综合的实现类

3.2 命令行参数解析

-i 输入解析文件名

3.3 实现思路

整体上分为两个步骤，首先将解析后的文件进行寄存器的绑定和操作调度，然后根据调度结果进行控制逻辑综合。

3.3.1 数据结构

3.3.1.1 绑定和调度 对于操作的调度和寄存器绑定而言，主要沿用 Basic.hpp 中的类，在 function 类下添加 $<$ 变量，寄存器 $>$ 的哈希表，以及全局变量 $global\_regs$ 和常量 $constant\_regs$。

在 statement 类下添加在该块逻辑里操作开始和结束的时间。

对于调度过程中，需要采用 ASAP 算法进行拓扑排序，所以引入一个节点类 $point$，带有节点值，前序节点，后序节点等信息，用于进行拓扑排序。

3.4 寄存器的绑定和操作调度

首先，在分配寄存器之前需要将寄存器分为全局寄存器，局部寄存器和常量三类，对于全局寄存器，整个函数流程稳定分配一个寄存器，可以在程序开头就分配好，对于常量而言不需要寄存器直接赋值即可，对于局部寄存器需要在块内做左边算法来获得其生命周期和所指的变量。

对于常量，既不是函数输入，仅出现在右值的量。将其设置为寄存器-1。

对于全局变量，即在两个或以上的块出现的量，对于每一块设置一个 get_vars 的方法，用于统计出现在本块中的所有变量，然后将块与块之间做交集，将交集求并即可得到全局变量表。在程序开始时依次分配寄存器。

对于局部变量，采取左边算法进行寄存器的动态绑定。需要首先建立拓扑排序树。对于一个 block:

1. 依次取出 Block 中的 statement，对于每个 statement，将 var 建立节点放入树中。
2. 遍历操作数，查看是否已经在树中，如果在，则将 var 的入度加 1，并将操作数对应节点的后置节点设为当前节点，当前节点前置节点设为操作数对应节点，如果不在，说明为常量或者全局变量，对入度没有影响。
3. 遍历 $vector < point > points$，寻找入度为 0 的点，入队，设置开始时间为 0，从寄存器数组 $vector < color > colors$ 中取出一个颜色，如果寄存器数组为空，则新增一种颜色。
4. 如果队列不为空，循环，取出队首，将队首的后置节点入度减 1，将颜色寄存器释放存。遍历队列中入度为 0 的点，更新点的开始时间，将颜色寄存器分配给该点，入队。

```
binding block:0
0 c(0) 0(-1)
binding block:start
0 i(2) 0(-1) 0(-1) i_inc(4) calc(-1)
0 cl(1) c(0) 0(-1) cr(3) calc(-1)
1 cond(5) i(2) n(5)
binding block:calc
0 ai(5) a(-1) i(2)
0 bi(6) b(-1) i(2)
2 ci(6) ai(5) bi(6)
3 cr(3) cl(1) ci(6)
0 i_inc(4) i(2) 1(-1)
binding block:ret
```

图 1: 拓扑排序树

5. 队列为空，程序结束，遍历 statement，设置 statement 的开始时间为 *var* 的开始时间。完成操作数的调度。

4 测试方法和验证

程序的输入数组 a,b 综合为 ram，在 vivador 中进行仿真。对于函数的输入，n 为外界规定的数组长度，a[i],b[i] 为从 ram 中读取的数值。对于输出，cl 对应输出结果 result，需要结果标志信号 done_flag。在执行 load 指令时，需要制定地址 i，即 a_ddr 和 b_ddr，同时需要读使能信号 a_rd_en 和 b_rd_en。

其外部完整电路为:

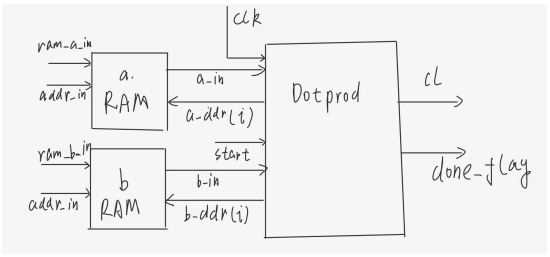

图 2: 结构示意图

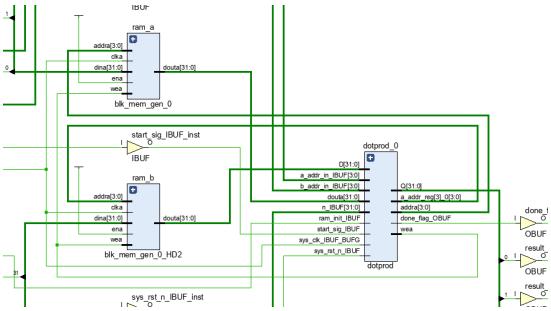

图 3: 综合结果

4.1 测试方法

4.2 验证结果

输入后的结果如下 (见 output.txt)