编译技术 Project-1 实践报告

组员

1700012759 周厚健

1700012934 刘添翼

1700012771 张旭睿

1700016629 常可

我们组在本次 Project 上的工作主要集中在/project1/solution/codeBuilder.cc 和/include/JsonParser.h 两个文件上，另外对/include/type.h，/include/IR.h 以及/src/IRPrinter.cc 几个文件也有较小改动。

JsonParser.h 负责从给定的文件中读取 json 信息，并将其中的所有信息提取出来放置于 Case 类中。具体处理流程大致如下：首先从 json 文件中每读一行并识别该行为五种信息中的哪一种，并将其分别放入 Case 类中的相应字符串槽位。读取完毕后除去 kernel 中的所有空格并从 kernel 中提取每个张量的维度大小信息放入 Tensor 类中，以及从每个张量的维度大小和数组下标提取成不等式组存入 Case 类（如$A<16,32>[i,j] \rightarrow \{i<16, j<32\}$），最后对每个不等式调用 Case::handleIndexBound()求出每个在数组下标里出现的变量的上下界并存入 Case::indexbound 中，以便之后生成循环体。

codeBuilder.cc 负责对 kernel 字符串进行语法分析，并在分析过程中利用 IR.h 提供的功能形成语法分析树。该文件主体为如图所示的六个函数，分别用于处理文法中对应的六个非终结符号：

```
 42 Group handleP(Case c) ...
104:
105 Stmt handleS(string s) ...
115:
116 Expr handleRHS(string s) { ...
182:
183: // get TRef into a Var (for IR.h)
184 Expr handleTRef(string s) { ...
212:
213 Expr handleSRef(string s) { ...
218:
219 Expr handleIdExpr(string s) { ...
```

其中，**handleP()**将整个 kernel 字符串按分号拆分成多个语句，并对每个语句调用 handleS()进行处理；随后从每个语句中找出所用到的循环索引并将其利用 Loopnest::make()包装成循环体，最后利用 Case 类中给出的实例输入和输出将整个 kernel 包装成一个 Kernel。

另外，此处对于需要求和的循环变量进行了特殊处理。由于求和的对象是以最外层的加减划分的，并且不同项所带的循环变量不同，所以我们在这里就将每一个 S 拆分为若干项再依次调用 handle(S)。

例如：A = B - (C + D * E) + F / G ; 可以拆分为如下形式：

(1) tmp_A = 0 ; (2) tmp_A = tmp_A + B ; (3) tmp_A = tmp_A - (C + D * E) ;

(4) tmp_A = tmp_A + F / G ; (5) A = tmp_A

handleS()简单地将在 handleP()中去掉了分号的语句按'='分成两半，并将左边交给 handleTRef()，右边交给 handleRHS()，并把两边的返回结果用 Move::make 组合起来。

handleRHS()中判定了加减号以及其他二元运算符的存在性，若有则只处理加减，没有再处理其他运算符，处理方式均为将字符串按该二元运算符切分成两半并递归交给handleRHS()，最后用 Binary::make()将左右两边组合起来；若没有任何运算符则判断其是否为括号、变量引用或是常数并分别作出处理。

handleTRef()中将张量引用按文法分成name <> []三部分，将<>中的常数储存为shape 并将[]中的每个下标传递给 handleIdExpr()形成 arg，最终用 Var::make()形成变量引用；**handleSRef()**则简单提取出变量名并返回一个标量引用。

handleIdExpr()中对运算符的处理方式与 handleRHS()中相似，同时还记录了整个语句中所使用到的循环 index，以便 handleP()中包装循环体。

除了这两个自编文件外，为满足从 IR 树生成 C++代码的需求，我们对 IRPrinter.cc 和 type.h 做出了适当修改；为满足向 C++表达式中生成括号的需求，我们向 IR.h 中添加了一个一元运算 UnaryOpType::Bracket，并向 IRPrinter.cc 中添加了相关语句。

以上为本组对本次 project 所做的工作。

部分编译过程截图及运行测试结果如下：

```
[100%] Linking CXX executable test1
[100%] Built target test1
zhou@ubuntu:~/Desktop/CompilerProject-2020Spring-Part1-master/build$ ./project1/test1
Random distribution ready
Example Wrong answer
Case 1 Success!
Case 2 is hidden
Case 3 is hidden
Case 4 Success!
Case 5 Wrong answer
Case 6 Success!
Case 7 Success!
Case 8 is hidden
Case 9 is hidden
Case 10 Success!
zhou@ubuntu:~/Desktop/CompilerProject-2020Spring-Part1-master/build$
```

其中，Case 5 的 Wrong Answer 为误差所致，经测试实际误差为 3e-4 左右：

```
zhou@ubuntu:~/Desktop/CompilerProject-2020Spri
ng-Part1-master/build$ ./project1/test1
Random distribution ready
Example Wrong answer
Case 1 Success!
Case 2 is hidden
Case 3 is hidden
Case 4 Success!
Case 5 Success!
Case 6 Success!
Case 7 Success!
Case 8 is hidden
Case 9 is hidden
Case 10 Success!
```

```
    std::cout << "Failed because of runtime error\n";
    return false;
}

// check
for (int i = 0; i < 16; ++i) {
    for (int j = 0; j < 32; ++j) {
        if (std::abs(golden[i][j] - A[i][j]) >= 3e-4) {
            std::cout << "Wrong answer\n";
            return false;
        }
    }
}
```

C++ Tab Width: 8 Ln 16

（图右为测试时修改的/project1/run.cc）

小组分工如下：

张旭睿：找出循环索引范围、给出语法分析器框架

周厚健：完成语法分析器、文件读取器

常可：修改 IRPrinter

刘添翼：求和处理，项目 Debug