Кроссплатформенный компилятор языка Cloud Sisal

А. А. Малышев, email: alex.m.work@yandex.ru

Институт систем информатики имени А.П. Ершова СО РАН

Аннотация. *Облачная система параллельного программирования CPPS, разрабатываемая в Институте систем информатики СО РАН (ИСИ СО РАН), использует функциональный язык Cloud Sisal для разработки, отладки, верификации и исполнения параллельных программ через веб-браузер. В докладе рассмотрен создаваемый кроссплатформенный компилятор языка Cloud Sisal. Рассмотрены задачи синтаксического разбора, построения промежуточного представления IR программы в виде графа потока данных, и кодогенерации (построения исходного кода на C++ по промежуточному представлению). Рассмотрены способы задания входных данных для программы и вывода результатов вычислений, а также тестирование компилятора.*

Ключевые слова: *компилятор, транслятор, парсер, язык программирования, C++, Python, Sisal, Cloud Sisal, LLVM, CPPS, parsimonious, грамматика, промежуточное представление программ.*

Введение

В докладе рассмотрен разрабатываемый кроссплатформенный компилятор языка Cloud Sisal [1-3] на Python, который транслирует Cloud Sisal-программы в программы на C++. Входным языком системы является язык Cloud Sisal, представляющий собой диалект языка Sisal 3.2 ("Streams and Iteration in a Single Assignment Language") [4-5]. Архитектура компилятора позволяет дописывать кодогенераторы для других языков: в частности, сделан экспериментальный кодогенератор в LLVM, поддерживающий небольшое подмножество языка Cloud Sisal (не включен в проект). Компилятор планируется использовать в системе CPPS [2].

Несмотря на то, что Python имеет репутацию "медленного" интерпретатора, в данном случае трансляция происходит за незначительное, по сравнению с компиляцией получаемого C++ - кода, время (десятки миллисекунд).

Компилятор разделён на две части: парсер и кодогенератор. Части взаимодействуют посредством передачи текстового представления объектов в стандартном виде (JSON), что позволило изолировать

решение двух задач друг от друга, а также легко использовать компоненты по отдельности и с другими реализациями Cloud Sisal.

На данный момент в компиляторе поддерживаются циклы с редукциями, некоторые виды циклов автоматически распараллеливаются с использованием стандарта OpenMP в C++-бэкэнде кодогенератора.

1. Модель внутреннего представления IR

Система CPPS использует единое внутреннее теоретико-графовое представление IR (Intermediate Representation) [3] Cloud Sisal программ, которое ориентировано на их семантическую и визуальную обработку и основано на атрибутированных иерархических графах [6]. В рамках этого представления подразумевается сборка программы из модулей перед ее интерпретацией или оптимизирующей трансляцией. При разработке внутреннего представления учитывались следующие существенные требования:

1. Машинная независимость как для представления параллелизма (нет явного разбиения вычислений на несколько потоков), так и для значений (независимость от разрядности машинной архитектуры) типов данных.
2. Полнота внутреннего представления, позволяющая транслировать любую конструкцию исходного языка в семантически эквивалентный фрагмент внутреннего представления.
3. Возможность ретрансляции в синтаксически корректную программу после преобразований внутреннего представления программы, сохраняющих ее семантику.
4. Простота интерпретации (исполнения заданных внутренним представлением вычислений) без каких-либо дополнительных преобразований внутреннего представления.
5. Структурированность объектов внутреннего представления для задания естественной вложенности одних конструкций исходного языка программирования в другие.
6. Все неявные действия над данными, например, преобразования типов, должны быть выражены явным образом с помощью объектов внутреннего представления.
7. Расширяемость, в смысле лёгкого введения новых объектов внутреннего представления для задания новых конструкций языков программирования и типов данных.

Внутреннее представление программы на языке Cloud Sisal может быть описано в виде иерархического ориентированного ациклического графа с портами [7-8]. Вершины такого графа соответствуют функциям,

описанным на языке Cloud Sisal. Дуги в таком графе отражают передачу данных между вершинами. В рассматриваемых графах также выделяется понятие порта. Порт служит для описания точки входа в вершину или точки выхода из вершины. Выходные порты соответствуют результатам функций, а входные - аргументам функций. Для вершин, соответствующих константам, множество входных портов пусто. Множество выходных портов содержит один или более элементов, т. к. функции языка Cloud Sisal поддерживают множественные результаты. Множества портов вершин конечны и упорядочены. Например, для следующей функции, вычисляющей числа Фибоначчи (листинг 1) IR будет иметь вид, представленный на рис. 1.

Листинг 1

Функция вычисления числа Фибоначчи

```
function Fib(m: integer returns integer)
    if (m < 2) then
        m
    else
        fib(m-1) + fib(m-2)
    end if
end function
```

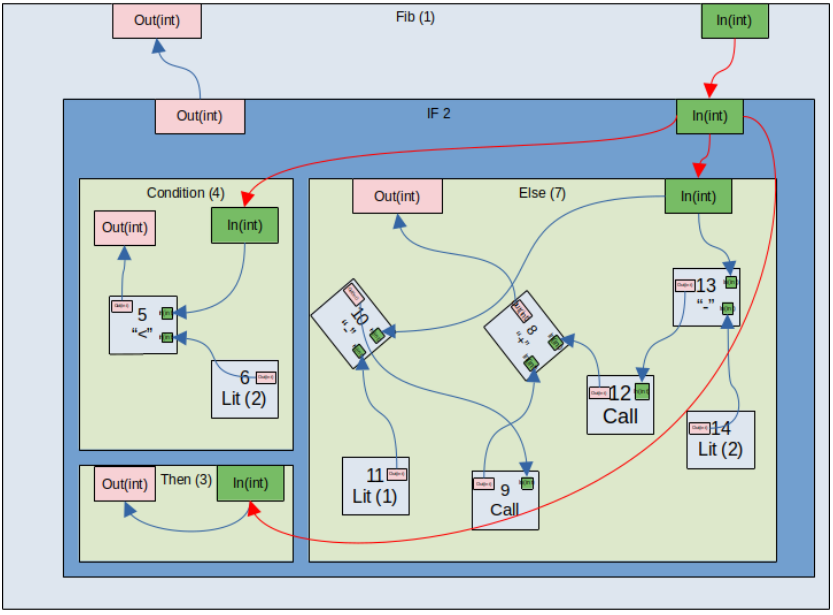

Рис. 1. IR программы вычисления чисел Фибоначчи

Таким образом, IR представляет собой "конвейер", обрабатывающий входные данные.

2. Парсер

Задача парсера – построить промежуточное представление IR (Intermediate Representation, граф потока данных) по исходному коду программы, которое потом будет передано кодогенератору для построения кода на C++, либо визуализатору графа, либо интерпретатору.

Парсер принимает на вход программу на Cloud Sisal (доступны передача по stdin, что удобно при использовании парсера в составе CPPS и для автоматизированного применения, и чтением файла) и преобразует её в промежуточное представление. Промежуточное представление можно экспортировать в виде JSON. Парсер использует библиотеку parsimonious (PEG-парсер) [9], которая не отделяет лексический анализ от синтаксического и позволяет упростить процесс разбора исходного кода за счёт удобных способов задания правил и описания обработки каждой конструкции.

Синтаксис языка задан в виде грамматики (фрагмент приведён на листинге 2).

Листинг 2

Фрагмент грамматики языка Cloud Sisal

```
module = (_ def _)* ~"$"
def = function / function_import / type_definition
type_definition = "type" _ identifier _ "=" _ type
# function:
function           = pragmas _ "function" _ identifier _
                             lpar _
                             function_arguments _
                             function_retvals _
                             rpar
                             _ multi_exp _
                        "end" _ "function"
pragmas           = ( _ pragma _ )*
pragma            = "/?" _ ~"[a-z_][a-z0-9_ (),]*"i _ ~"\n*" _
empty             = ~"\s*"
function_import   = _ "function" _ identifier _  "[" _
      type_list _ function_retvals _ "]" _
function_arguments = args_groups_list / _
function_retvals   = ("returns" _ type_list) / _
type_list          = type (_ "," _ type)*
args_groups_list   = arg_def_group (_ ";" _ arg_def_group)*
```

Итоговый JSON-вывод парсера имеет структуру, приведённую в листинге 3.

Листинг 3

Общая структура IR JSON, выводимого парсером

```
{
      IR: *промежуточное представление программы*,
      errors: [*список ошибок, если они есть, то поле «IR»
                будет пустым*],
      warnings: [*список предупреждений*]
}
```

Для задач визуализации доступен экспорт в GraphML [10].

3. Кодогенератор

Кодогенератор принимает на вход в текстовом JSON-представлении IR программы и генерирует по нему C++-код программы. Получаемый код компилируется при помощи GCC. Было принято решение сосредоточиться на C++ так как предполагаемый супервычислитель поддерживает этот язык, а также из за относительной простоты описания распараллеливания программ при помощи стандарта OpenMP.

Функция main Cloud Sisal-программы переименовывается в функцию sisal_main, а функция main C++-программы загружает входные данные и вызывает sisal_main, используя входные данные в качестве аргументов. Имена остальных функций в C++-программе соответствуют именам в исходной программе на Cloud Sisal.

Сгенерированные программы оптимизируются компилятором gcc при компиляции.

Приложены усилия для улучшения читаемости получаемого C++-кода, что облегчило его анализ: совпадают имена переменных там, где это возможно, переменные сгруппированы по типу, комментариями обозначены границы разных частей тела функции и пр. Например, в листинге 4 показано, как будет выглядеть часть программы, в которую транслируется описанная ранее программа вычисления чисел Фибоначчи (не приведён служебный код).

Листинг 4

Пример исходного кода на C++, полученного на выходе кодогенератора

```
int Fib(int M)
{
  int function_result;
  bool if_test, bin;
  bin = M < 2;
  if_test = bin;
  if(if_test)
  {
    function_result = M;
```

```
  }
  else
  {
    int call, bin2, call2, bin3, bin4;
    bin2 = M - 1;
    call = Fib(bin2);
    bin3 = M - 2;
    call2 = Fib(bin3);
    bin4 = call + call2;
    function_result = bin4;
  }
  return function_result;
}

int sisal_main(int M)
{
  int call;
  call = Fib(M);
  return call;
}

int main(int argc, char **argv)
{
  Json::Value root;
  std::cin >> root;
  Json::Value json_result;
  CHECK_INPUT_ARGUMENT("M");
  int M = root["M"].asInt();
  int main_result = sisal_main(M);
  json_result["port0"] = main_result;
  std::cout << json_result << "\n";
  std::cout << std::endl;
  return 0;
}
```

Получаемые исполняемые файлы принимают через stdin входные данные программы (аргументы функции main) в виде JSON-текста (используется библиотека JSONCpp), что облегчает работу с компилятором в автоматическом режиме и его использование в составе других систем, например сред разработки. Наличие нужных аргументов главной функции среди входных данных также проверяется программой. Результаты вычислений также выводятся в стандартный вывод в виде JSON.

Рассмотрим программу перемножения матриц, представленную в листинге 5 (в языке есть встроенная функция получения длины массива, данный пример - учебный, как и остальные, приведённые в докладе).

Листинг 5

Программа перемножения матриц на Cloud Sisal

```
function main(A, B: array[array[integer]]; M,N,K: integer
returns array[array[integer]])
    for i in 1,M
    repeat
        Ci := for j in 1,N
                  repeat
                      Cij := for k in 1,K repeat
                                 Sij := A[i][k]*B[k][j]
                                 returns sum of Sij
                             end for
                  returns
                      array of Cij
                  end for
    returns
        array of Ci
    end for
end function
```

Если на вход были поданы данные, представленные в листинге 6, то на выходе будут данные, представленные в листинге 7 (также в виде текста, отформатированы вручную для удобного чтения).

Листинг 6

Входные данные для программы перемножения матриц

```
{
    "A": [[ 5, 74, 76, 42, 47],
          [37, 73, 53, 59, 38],
          [46, 25, 89, 81, 59],
          [ 0,  3,  9, 59, 18],
          [ 0, 84, 73, 85, 41]],
    "B": [[94, 79, 31, 26, 57],
          [ 0, 25, 83, 91, 27],
          [65, 25, 51, 99, 57],
          [ 3, 69, 56, 25, 79],
          [68, 64, 64, 38, 53]],
    "M": 5,
    "N": 5,
    "K": 5
}
```

Листинг 7

Результат вычислений программы перемножения матриц

```
{
    "port0":
    [[ 8732, 10051, 15533, 17224, 12424],
     [ 9684, 12576, 15645, 15771, 13776],
     [14364, 15849, 16352, 16549, 17896],
```

```
      [ 1986,  5523,  5164,  3323,  6209],
      [ 7788, 12414, 18079, 18554, 15317]]
}
```

Стандартные типы данных Cloud Sisal транслируются во встроенные типы данных C++ и типы, построенные по шаблонам библиотеки STL C++, что делает код более понятным и лаконичным. Например, тип integer транслируется в int C++, а array[integer] (или, также допустимый, "array of integer") - в vector<integer>.

Разработана система автоматического тестирования компилятора с набором тестовых Cloud Sisal-программ и наборами входных данных и ожидаемых правильных результатов вычислений. Можно задать несколько наборов входных и ожидаемых выходных данных. Пример данных для тестировния представлен в листинге 8 (тестовая программа) и в листинге 9 (входные данные и ожидаемые результаты). В данном примере программа из "Sisal For Science" [11] имеет два значения на выходе).

Листинг 8

Пример тестовой программы

```
function main(x, y: array[integer] returns array[integer],
array[integer])
    for i in 1, size(x)
    repeat
      minimum, maximum := if x[i] < y[i] then
                                    x[i], y[i]
                              else
                                    y[i], x[i]
                          end if
      returns array of maximum, array of minimum
    end for
end function
```

Листинг 9

Пример входных данных и ожидаемых результатов

```
[
  {
    "input": {
      "x": [1, 3, 2],
      "y": [3, 2, 1]
    },
    "output": {
      "port0": [3, 3, 2],
      "port1": [1, 2, 1]
    }
  }
]
```

Заключение

В докладе представлено ознакомительное описание кроссплатформенного компилятора языка Cloud Sisal в текущем состоянии, в котором компилятор уже поддерживает определённое подмножество языка Cloud Sisal, что позволяет решать учебные и исследовательские задачи. На данный момент не все запланированные конструкции языка поддерживаются компилятором: предстоит реализовать пользовательские типы и пользовательские редукции, взаимодействие нескольких модулей, исполнение директив компилятора и автоматизацию распараллеливания других структур.

Список литературы

1. Касьянов, В. Н. Язык программирования Cloud Sisal / В. Н. Касьянов, Е. В. Касьянова. – Новосибирск, 2018. – 45 с. – (Препринт/ РАН, Сиб. отд-ние, ИСИ; N181).
2. Система облачного параллельного программирования CPPS: визуализация и верификация Cloud Sisal программ / Касьянов, В. Н. [и др.]; отв. ред. В. Н. Касьянов. – Новосибирск: ИПЦ НГУ, 2020. – 256 с.
3. Касьянов, В. Н. Методы и система облачного параллельного программирования / В. Н. Касьянов, Е. В. Касьянова // Информатика: проблемы, методология, технология: Сборник материалов XIX международной научно-методической конференции. Под ред. Д. Н. Борисова – Воронеж: Вэлборн, 2019. – С. 1552-1556.
4. Касьянов, В. Н. Язык программирования SISAL 3.2 / В. Н. Касьянов, А. П. Стасенко // Методы и инструменты конструирования программ. – Новосибирск: ИСИ СО РАН, 2007. – С. 56-134.
5. Касьянов, В. Н. Функциональный язык Sisal 3.0 / В. Н. Касьянов, Ю. В. Бирюкова, В. А. Евстигнеев // Поддержка супервычислений и интернет–ориентированные технологии. – Новосибирск, 2001. – С. 54-67.
6. Касьянов, В. Н. Иерархические графы и графовые модели: вопросы визуальной обработки / В. Н. Касьянов // Проблемы систем информатики и программирования. – Новосибирск: ИСИ СО РАН, 1999. – С. 7-32.
7. Гордеев, Д. С. Визуализация внутреннего представления программ на языке Cloud Sisal // Научная визуализация. – 2016. – Т. 8. – №2. – С. 98-106.
8. Касьянов, В. Н. Методы и алгоритмы визуализации графовых представлений функциональных программ / В. Н. Касьянов, Т. А. Золотухин, Д. С. Гордеев // Программирование. – 2019. – № 4. – С. 19-27.

9. Парсер parsimonious [Электронный ресурс]. – Режим доступа: https://github.com/erikrose/parsimonious

10. Описание языка GraphML [Электронный ресурс]. – Режим доступа: http://graphml.graphdrawing.org

11. Raymond, D. J. Sisal for Science [Электронный ресурс]. – Режим доступа: http://kestrel.nmt.edu/~raymond/software/sisal/science/science