Программные средства поддержки дистанционного обучения функциональному программированию

В. Н. Касьянов, email: kvn@iis.nsk.su [1,2]
А. А. Малышев, email: alex.m.work@yandex.ru[1]

[1] Институт систем информатики имени А.П. Ершова СО РАН
[2] Новосибирский государственный университет

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 рассмотрена созданная онлайн среда системы CPPS, которая вместе с созданными компилятором и профилировщиком системы позволяет пользователю на любом устройстве, имеющем выход в Интернет, разрабатывать и исполнять функциональные программы на языке Cloud Sisal.*

Ключевые слова: *дистанционное обучение, параллельное программирование, система программирования, функциональное программирование, язык Cloud Sisal.*

Введение

Первым языком функционального программирования был Лисп, разработанный в 1961 г. американским ученым Дж. Маккарти. Хотя язык и получил широкую известность, благодаря его большей выразительности и элегантности по сравнению с традиционными языками, его применимость ограничивалась в основном задачами искусственного интеллекта. Новый период функционального программирования начался с Тьюринговой лекции 1978 г. изобретателя Фортрана Дж. Бекуса «Может ли программирование освободиться от бремени фон-неймановского стиля? Функциональный стиль и его алгебра программ». Это новое понимание и более широкое принятие функционального программирования определилось в первую очередь начатым в эти годы процессом по переходу к рассмотрению задачи программирования в ее полном контексте, начиная со спецификации задачи и логического анализа ее разрешимости, побочным продуктом которого является сама программа. Появление вычислительных систем с параллельными архитектурами еще более повысили значимость функционального программирования, поскольку оно позволяет

освободить пользователя от большинства сложностей параллельного программирования, присущих императивным языкам, и возложить на компилятор вопросы построения программы, эффективно исполняемой на вычислительной системе конкретной параллельной архитектуры. Кроме того, многие технические проблемы системного и прикладного программирования обретают ясность при изложении их решения в функциональном стиле.

Облачная система параллельного программирования CPPS, разрабатываемая в ИСИ СО РАН, использует функциональный язык Cloud Sisal [1] для разработки, отладки, верификации и исполнения параллельных программ. В рамках создаваемой системы CPPS [2] прикладной программист будет иметь возможность через браузер создавать, отлаживать и верифицировать Cloud Sisal программу в визуальном стиле и без учета целевого вычислителя, а затем с помощью оптимизирующего кросс-компилятора производить настройку отлаженной программы на тот или другой супервычислитель, доступный ему по сети, с целью достижения высокой эффективности исполнения получаемой параллельной программы, а также передавать построенную программу супервычислителю на счет и получать результаты.

В докладе рассмотрена созданная онлайн среда системы CPPS, которая вместе с созданными компилятором и профилировщиком системы позволяет пользователям на любых устройствах, имеющих выход в Интернет, разрабатывать и исполнять функциональные программы на языке Cloud Sisal.

1. Онлайн среда

В текущей версии системы CPPS используется созданная онлайн среда [3], которая поддерживает взаимодействие пользователей с существующими компонентами системы CPPS через веб-браузер. В частности, существующая среда позволяет зарегистрированному пользователю создавать программы на языке Cloud Sisal, запускать их и получать результаты в веб-приложении.

Пользователь может хранить несколько проектов, каждый из которых может состоять из нескольких модулей (проект или модуль можно создать, нажав соответствующую кнопку и введя название и, опционально, описание). Каждому проекту и каждому модулю можно дать описание в соответствующих полях в редакторе системы как до, так и после их создания (см. Рис. 1).

Входные данные программы (аргументы функции main) предлагается описывать в формате JSON. Вот так, например, можно задать различные параметры:

```
{
    "M" : 15, // число
    "a" : 15.1, // число
    "V1": [1 , 2 , 3 , 4.0], // "вектор"
    "A" : [
          [1 , 2 , 3],
          [4 , 5 , 6]
        ] // "матрица"
}
```

где «вектор» и «матрица» являются на самом деле просто числовыми массивами разной размерности.

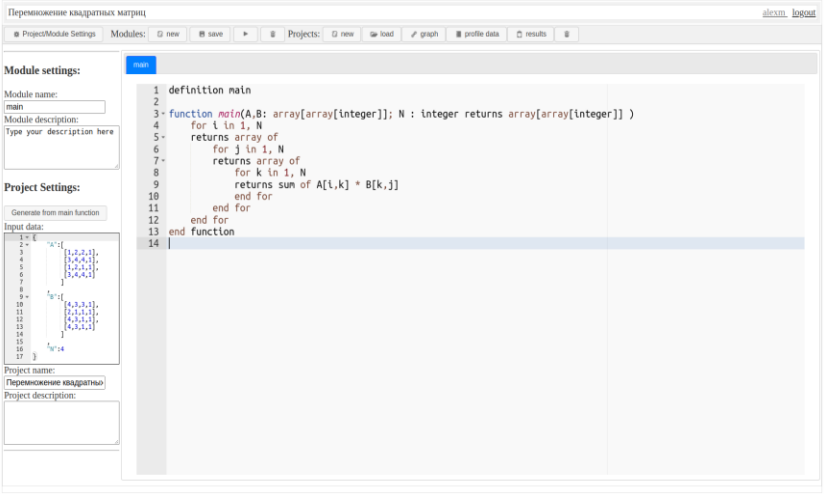

Рис. 1. Внешний вид редактора системы CPPS

Для упрощения понимания формата и задания входных данных, система может для пользователя генерировать по аргументам функции main текст примера задания её входных данных. Для этого нужно сначала нажать кнопку «Generate from main function» над полем ввода данных, а затем просто можно вписать свои значения в автоматически сгенерированный шаблон входных данных функции main.

Онлайн среда системы CPPS создана с использованием фреймворка django и построена по принципам сервисной архитектуры. Для работы сервисов используются LXC для linux-систем и VirtualBox для Windows (в которой работает компилятор Sisal 3.1, рассмотренный ниже), системы обмениваются данными (например, для трансляции исходного

кода) посредством HTTP-запросов. На данный момент в системе реализованы два сервиса: сервис редактора (он же осуществляет компиляцию и контролируемое исполнение C#-кода) и сервис транслятора на Windows. В сервисе Windows приёмом и обработкой запросов занимается сервер, написанный на языке Python.

```
1  definition main1
2
3  function Fib( M : integer returns integer )
4      if M < 2 then
5          M
6      else
7          Fib(M - 1) + Fib(M - 2)
8      end if
9  end function
10
11 function main( M : integer returns integer )
12     Fib(M)
13 end function
14
```

Рис. 2. Sisal-программа

Для редактирования кода на языке Cloud Sisal во frontend-части использован редактор ACE [4] с подсветкой синтаксиса, который встроен в Web-приложение. Для создания диалоговых окон и элементов управления используется библиотека jQuery UI [5]. Для упрощения разработки frontend-части также использована библиотека jQuery [6]. Последние две библиотеки, несмотря на репутацию устаревших, прекрасно справляются с возникающими задачами и за счёт своей легковесности делают редактор более доступным для работы на широком спектре пользовательских устройств.

2. Профилировщик

Существующая версия системы CPPS содержит компилятор Sisal 3.1, который осуществляет преобразование модулей, написанных на языке Cloud Sisal в классы, написанные на C#, которые могут быть использованы для составления программ на языке C# и скомпилированы системой при помощи MONO или .NET.

Рассмотрим, например, приведенный на Рис. 3 код, полученный в результате обработки компилятором программы, изображенной на Рис. 2. Данный код представляет собой C#-класс, содержащий образы функций на Cloud Sisal, этот код включается в состав проекта MONO. Система формирует решение (solution) MONO, используя построенный

код и данные с переменными в формате JSON. Требуется сгенерировать код главного класса, в который включена инициализация входных данных и вызов функций из классов, сгенерированных транслятором. Входные данные используются при генерации главного класса решения и размещаются там как hardcoded данные. Транслятор Sisal 3.1 был разработан с использованием технологии COM. По этой причине в системе используется отдельный (виртуальный) Windows-сервер для транслятора.

```
1  // This file is generated by Sisal 3.1 compiler.
2
3  using System;
4  using System.Threading;
5  using ConsoleApp1;
6
7  namespace ConsoleApp1
8  {
9
10     public class SISAL_CODE
11     {
12         public static bool  Fib(CI32 V_1,  out CI32 V_3)
13         {
14             CBOOL  V_7;
15             CI32  V_17;
16             CI32  V_21;
17             CI32  V_27;
18             CI32  V_31;
19
20             V_7 = ( new CBOOL(V_1 < new CI32(0x2)) );
21             if ( V_7._Value == true )
22             {
23                 V_3 = V_1;
24             }
25             else
26             {
27                 V_17 = V_1 - new CI32(0x1);
28                 V_27 = V_1 - new CI32(0x2);
29                 Fib(V_17,  out V_21);
30                 Fib(V_27,  out V_31);
31                 V_3 = V_21 + V_31;
32             }
33             return true;
34         }
35         public static bool  main(CI32 V_33,  out CI32 V_37)
36         {
37
38             Fib(V_33,  out V_37);
39             return true;
40         }
41     }
42
43 } // namespace Sisal_C_sharp
44
45
```

Рис. 3. Код на языке C#, получаемый компилятором Sisal 3.1

Если исходный код модуля не менялся с момента последней трансляции, то для него используется сохраняемый в базе данных результирующий код, чтобы не тратить время на ненужную его повторную (иногда весьма затратную по времени) трансляцию.

MONO-проект исполняется под контролем системы. При этом ограничивается время исполнения, чтобы учесть те случаи, когда программа не может завершиться из-за бесконечной рекурсии (или бесконечного цикла) или требует слишком много времени для своего исполнения или профилирования.

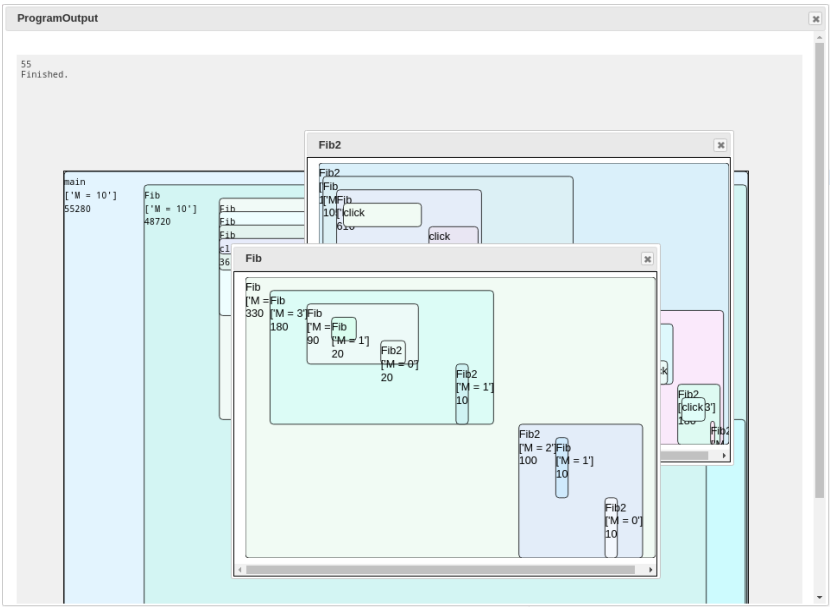

Рис. 4. Интерактивный просмотр результата профилирования программы, вычисляющей числа Фибоначчи

Результаты работы программы, её профилирования и ошибки, происходящие в процессе трансляции и исполнения, передаются во frontend серверной частью и показываются пользователю в отдельном окне.

Одной из задач при реализации системы является профилирование программ (Рис. 4). Этот процесс позволяет пользователю лучше понять свою программу и исследовать её «узкие места», т. е. те её сегменты, обработка которых занимает наибольшее количество системных ресурсов, главным образом, процессорных вычислений. Процесс

профилирования функциональных программ имеет ряд особенностей, например, для него заранее неизвестна последовательность вычислений отдельных функций [7].

Для профилирования Cloud Sisal программы полученный при её трансляции код на языке C# снабжается вставками кода в виде вспомогательных функций и структуры (см., Рис. 5). Эти функции сохраняют время начала и время конца выполнения функций, потоки, в которых выполнялась функция (на данный момент не реализовано), значения аргументов (реализовано не полностью), переданные функции, а также вложенность вызовов (какая функция какой функцией была вызвана). Данные функции размещаются в начале и в конце тел функций. После того, как инструментированная таким образом программа завершает свою работу, собираемая информация используется для составления интерактивного отчёта о работе программы.

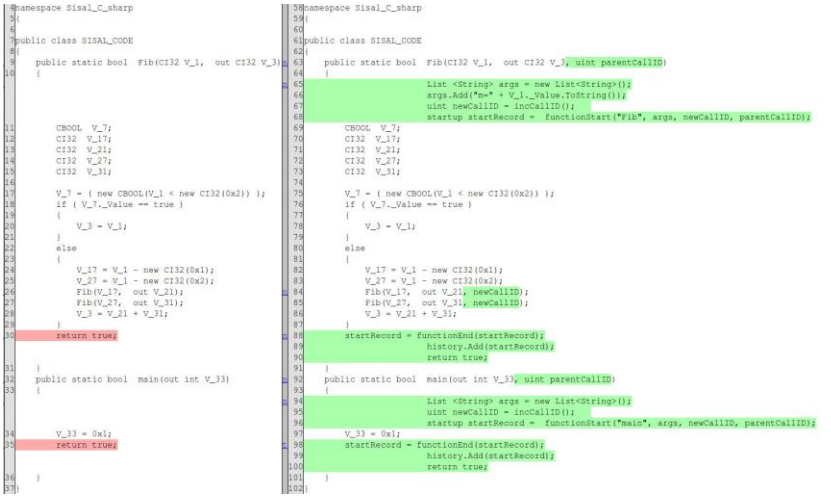

Рис. 5. Автоматическая вставка кода для профилирования функциональной программы

В отчёте собираемые данные визуализируются в виде интерактивного дерева вызовов в векторном формате SVG. Для формирования изображения используется библиотека svgwrite Python. Графический интерактивный отчёт (Рис. 4) представляет собой сегмент веб-приложения, отображающий дерево вызовов функций на плоскости в виде прямоугольников, изображающих вызовы и содержащих

информацию о том, сколько времени потребовалось на выполнение данного вызова, и с какими аргументами он выполнялся. Слишком большие для визуализации поддеревья отображаются в виде отдельных вершин, но пользователь всегда может развернуть любое поддерево, нажав на изображение соответствующей вершины. Смещение и ширина изображения вершины по горизонтали соответствуют времени (началу и длительности) выполнения соответствующего вызова функции. Изображение вызова содержит также идентификатор вызываемой функции, значения аргументов и длительность его выполнения.

Заключение

В докладе была рассмотрена созданная онлайн среда облачной системы параллельного программирования CPPS, которая вместе с созданными компилятором и профилировщиком системы позволяет обучать функциональному программированию пользователя при наличии у него устройства, имеющего выход в Интернет. Её использование может происходить по следующей схеме. Сначала зарегистрированный пользователь в редакторе создаёт тексты функциональной программы и ее входных данных. Затем текст программы транслируется на сервере транслятора в C#, далее входные данные тоже транслируются в C# и вместе с исходным кодом используется для формирования MONO-Solution, которое затем компилируется и исполняется. Результатами исполнения являются данные в stdout и json-файл с данными профилирования, которые затем используются для построения итогового отчёта, предоставляемого пользователю.

Использование других создаваемых инструментов системы CPPS будет позволять пользователям представленной в докладе онлайн среды через веб-браузер визуально отлаживать и формально верифицировать функциональные программы на языке Cloud Sisal, а также исполнять параллельные программы, автоматически построенные компилятором по их функциональным спецификациям.

Литература

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 / В. Н. Касьянов, Д. С. Гордеев, Т. А. Золотухин [и др.]; под ред. В. Н. Касьянова. — Новосибирск: ИПЦ НГУ, 2020. — 256 с.

3. Онлайн среда для системы CPPS [Электронный ресурс] — Режим доступа: http://cpps.iis.nsk.su
4. Редактор кода ACE [Электронный ресурс] — Режим доступа: https://ace.c9.io/
5. Библиотека jQuery UI [Электронный ресурс] — Режим доступа: http://jqueryui.com
6. Библиотека jQuery [Электронный ресурс] — Режим доступа: http://jquery.com
7. Sansom, P. M. Profiling Lazy Functional Programs / P. M. Sansom, S. L. Peyton Jones // Proceedings of the 1992 Glasgow Workshop on Functional Programming. — London: Springer, 1993. — pp. 227–239.