МЕТОДЫ И СИСТЕМА ОБЛАЧНОГО ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

Касьянов В. Н., Касьянова Е. В.

Институт систем информатики им. А. П. Ершова СО РАН, Новосибирск

Доклад посвящен проекту CPPS, выполняемому в ИСИ СО РАН при поддержке Российского научного фонда. Цель проекта — разработка языковых и программных средств, поддерживающих создание, верификацию и отладку архитектурно-независимых параллельных программ и их корректное преобразование в эффективный код параллельных вычислительных систем различных архитектур. В докладе описывается сама создаваемая система CPPS, ее входной язык Cloud Sisal и язык внутреннего представления Cloud Sisal программ.

***Ключевые слова*:** *аннотированное программирование, графовое представление программ, параллельное программирование, функциональное программирование, язык программирования.*

Введение

Параллельные вычисления являются одной из главных парадигм современного программирования и охватывают чрезвычайно широкий круг вопросов разработки программ. Ввиду значительно более сложной природы параллельных вычислений по сравнению с последовательными большое значение приобретают методы автоматизации разработки параллельного программного обеспечения, основанные на применении техники формальных моделей, спецификаций и преобразований параллельных программ.

Фундаментальными проблемами организации параллельных вычислении являются следующие: проблема увеличения производительности и эффективности использования многопроцессорных и распределенных вычислительных систем и проблема повышения уровня интеллектуализации программирования параллельных систем. Они не являются независимыми, ибо организация высокопроизводительных вычислений в многопроцессорной системе современной архитектуры оказывается слишком сложной для попыток ее решения без средств интеллектуализации программирования в такой системе. Трудность решения вопросов программирования параллельных систем определяется тем обстоятельством, что вопросы организации взаимодействий и синхронизации параллельных процессов существенно усложняют разработку параллельных алгоритмов и программ по сравнению с их традиционными (последовательными) вариантами. Одним из перспективных путей совместного решения указанных проблем является разработка декларативных средств описания и реализации параллельных вычислений.

Первым языком функционального программирования был язык 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

Касьянов В. Н., Касьянова Е. В. Методы и система облачного параллельного программирования // Проблемы оптимизации сложных систем: Материалы XIV Международной Азиатской школы-семинара (20-31 июля 2018 г.). – Алматы, 2018. – Часть 1. – С. 298-307.

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.

Разработка функциональных методов параллельного программирования успешно продолжилась в конце 70-х в языках VAL и БАРС, а также в ряде более современных проектов DCF, Пифагор, COLAMO и др., среди которых следует выделить язык SISAL (аббревиатура с английского выражения Streams and Iterations in a Single Assignment Language) [1], первая версия которого относится к 1983 г. SISAL разрабатывался как язык функционального программирования, специально ориентированный на параллельную обработку и на замену языка Фортран на суперкомпьютерах в научных вычислениях. О реальном вытеснении говорить еще рано, но SISAL как язык параллельного программирования достаточно интересен сам по себе и уже нашел свое применение в десятках организаций разных стран мира. Существует несколько реализаций языка SISAL (версии 1.2) для суперЭВМ, в частности на Denelcor HEP, Vax 11-780, Cray-1, Cray-X/MP, выполнены работы по созданию прототипа оптимизирующего компилятора с языка SISAL 1.2 в распределенные программы для вычислителей, аппаратно или программно поддерживающих многонитевые вычисления, таких как, например, TERA, *T, TAM и MIDC. Ливерморская национальная лаборатория и Манчестерский университет разработала усовершенствованную версию языка SISAL-90, которая пока еще нигде не была реализована.

Доклад посвящен облачным методам и системе параллельного программирования CPPS, разрабатываемых в лаборатории конструирования и оптимизации программ Института систем информатики им. А.П. Ершова СО РАН за счет гранта Российского научного фонда (проект 18-11-00118). В нем описывается сама создаваемая система CPPS, ее входной язык Cloud Sisal и язык внутреннего представления Cloud Sisal программ.

1. Система параллельного программирования CPPS

Современные подходы к разработке параллельных программ в основном являются архитектурно-ориентированными, когда для достижения эффективной работы создаваемые программы тесно связаны с архитектурами параллельных вычислительных систем, на которых они выполняются и, как правило, разрабатываются. Поэтому требования к квалификации разработчиков параллельных программ весьма высоки, тем более, что протестировать и отладить параллельную программу намного сложнее, чем последовательную, а проблема верификации параллельных программ весьма далека от решения не только практически, но и теоретически. Причем, лишь у узкого круга отечественных пользователей есть доступ к высокопроизводительной вычислительной технике, которая по числу суперкопьютеров и их суммарной мощности весьма уступает имеющейся в развитых странах и сосредоточена в сравнительно небольшом числе мест, вне которых разработка параллельных программ не ведется, но работает основная масса прикладных программистов.

Более того, в современной вычислительной технике идет постоянная смена архитектурных парадигм, что, в свою очередь, ведет к проблеме переносимости уже разработанных параллельных программ. Приходится постоянно адаптировать уже созданный продукт под изменившиеся аппаратные средства. Это обуславливается тем, что различные параллельные вычислительные системы имеют свойственные только им ресурсные ограничения, которые необходимо учитывать во время разработки программы. Проведение таких адаптаций является весьма интеллектуальной задачей, требующей существенного переписывания параллельных программ и выполнения практически заново их верификации и

отладки. В результате адаптированные параллельные программы зачастую содержат новые ошибки и не являются столь эффективными, как хотелось бы и могло быть.

Поэтому представляется весьма перспективным выполняемый в ИСИ СО РАН проект по разработке языковых и программных средств, поддерживающих создание, верификацию и отладку архитектурно-независимых параллельных программ и их корректное преобразование в эффективный код параллельных вычислительных систем различных архитектур с помощью семантических преобразований.

Будут разработаны методы и создана экспериментальная версия облачной расширяемой интегрированной визуальной системы параллельного программирования CPPS на языке Cloud Sisal [2], который продолжает традицию предыдущих версий языка SISAL, оставаясь функциональным потоковым языком, ориентированным на написание больших научных программ, и расширяет их возможности средствами поддержки облачных вычислений.

Функциональная семантика языка Cloud Sisal гарантирует детерминированные результаты для параллельной и последовательной реализации — то, что невозможно гарантировать для традиционных императивных языков, подобных языкам Фортран или С. Более того, неявный параллелизм языка снимает необходимость переписывания исходного кода при переносе его с одного вычислителя на другой. Гарантировано, что Cloud-Sisal-программа, правильно исполняющаяся на персональном компьютере, будет гарантированно правильно исполняться на любом высокоскоростном параллельном или распределенном вычислителе.

При этом, методы аннотированного программирования [3] и конкретизирующих преобразований [4], поддерживаемые системой CPPS, позволят в рамках декларативного стиля программирования настраивать процессы адаптации переносимых параллельных программ на классы задач и архитектуру вычислителя при сохранении их корректности, а также получать более эффективный параллельный код за счет использования при адаптации знаний пользователя о задаче, программе и вычислителе, выраженные в аннотациях.

Система CPPS разрабатывается как интегрированная облачная среда программирования на языке Cloud Sisal, которая содержит как интерпретатор, поддерживающий диалоговое взаимодействие с пользователем при построении и отладке программы, так и оптимизирующий кросс-компилятор, осуществляющий построение параллельной программы по ее функциональной спецификации.

В рамках CPPS прикладной программист будет иметь возможность через браузер создавать, верифицировать и отлаживать 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.

Система CPPS использует внутреннее теоретико-графовое представление функциональных программ, которое ориентировано на их визуальную обработку и основано на атрибутированных иерархических графах [5].

Система CPPS будет поддерживать построение наглядных изображений графовых внутренних представлений Cloud-Sisal-программ и их использование при конструировании корректных функциональных программ. Она будет также поддерживать построение визуальных представлений внутренних структур данных, возникающих в компиляторе при построении параллельных программ, и динамических процессов, возникающих при исполнении построенных параллельных программ, и их использование для управляемой оптимизации с целью повышения рабочих характеристик параллельных программ, получаемых с помощью компилятора по их функциональным спецификациям.

2. Язык Cloud Sisal

Язык Cloud Sisal обладает обычными преимуществами языков функционального программирования, такими, как, например, однократное присваивание (т. е. каждая переменная в программе определяется только один раз), но содержит массивы и циклы, которые не присуще функциональным языкам. Рассмотрим следующий фрагмент Cloud Sisal программы:

```
type OneDim = array [..] of integer;
type TwoDim = array of OneDim;
function generate( N : integer returns TwoDim, OneDim )
    for i in 1, N cross j in 1, N do
        A := i * j;
        B := i + j
    returns array [.., ..] of A array of B
    end for
end function
```

Первые два строчки определяют имена типов для массивов. Видно, что размеры в них не указаны, и все экземпляры описанных составных типов данных должны динамически создаваться, изменяться и удаляться во время выполнения программы. Только форма и типы элементов содержатся в данных спецификациях типов массивов. Во второй строчке (в определении типа TwoDim) форма опущена и по умолчанию полагается равной [..].

Заголовок функции «generate» показывает, что ожидается один целочисленный аргумент, «N», и вычисляются (возвращаются) два неименованных значения. Каждое возвращаемое значение представляет собой массив целых чисел, но опять же, указываются только формы массивов, а не их размеры. Имена могут быть привязаны к этим возвращенным значениям в месте вызова функции, если это потребуется программисту.

Вызов функции семантически эквивалентен воспроизведению кода функции на месте вызова с соответствующей заменой параметров. Эта эквивалентность, часто называемая «ссылочной прозрачностью», является фундаментальным свойством функциональных языков и представляет собой одно из достоинств языка Cloud Sisal. Данное свойство в частности упрощает процессы анализа, выполняемые оптимизирующим компилятором, поскольку функции не имеют побочных эффектов и являются детерминированными. Другими словами, две любые функции могут выполняться параллельно, если между функциями не существует зависимости по данным, а одна и та же функция с одинаковыми фактическими параметрами всегда возвращает одинаковые значения.

Все выражения Cloud Sisal программы, включая функции целиком, вычисляют наборы значений. В приведенном выше случае функция «generate» вычисляет двумерный и одномерный массивы, которые являются значениями выражения, содержащегося в определении функции. Указанное выражение представляет собой конструкцию цикла, который говорит компилятору для Cloud Sisal о потенциальном параллелизме. Этот цикл имеет индексный диапазон, определяемый как декартово произведение двух более простых диапазонов. Это означает, что тело цикла будет выполняться столько раз, сколько есть значений в диапазоне индексов, в данном случае N*N, и все экземпляры тела будут независимыми, поскольку между ними нет зависимостей по данным. Те наборы независимых тел цикла, какие будут выполняться параллельно, а какие нет, будут выбираться на основе осуществляемого компилятором и системой выполнения анализа связанных с этим затрат, а также по опциям, заданным программистом.

Нахождение имен «A» и «B» внутри тела цикла не следует рассматриваться как повторное использование этих имен в смысле присваивания переменной в императивной программе. Здесь эти имена используются для обозначения значений в теле цикла, и на самом деле они, вероятнее всего, не будут реально существовать в исполняемой программе. Важным моментом здесь является то, что каждый экземпляр тела цикла, содержащий конкретные значения для *i* и *j*, будет независимо вычислять конкретные экземпляры целых

значений, определенных как *i* * *j* и *i* + *j*; затем все эти отдельные значения будут собраны вместе в пару массивов и возвращены. Позиции значений в массивах результатах, а также общий размер и размерность возвращаемых массивов определяются их формами и диапазонами индексов цикла. В данном случае возвращается два массива, каждый из которых состоит из N^2 целых чисел: двумерный массив с индексом от 1 до N по каждому измерению и одномерный массив с индексом, который изменяется от 1 до N^2. Использование временных имен в цикле не является обязательным, и вышеприведенное условие возврата может быть переписано следующим образом:

returns array [.., ..] of i * j array of i + j

без изменения конечных результатов. При таком изменении тело цикла станет пустым, и по существу язык рассматривает выражения в «array of» как анонимные временные.

Язык предлагает пользователю богатый набор разнообразных стандартных редукций, а также допускает определение и использование своих собственных редукций. Использование редукций хорошо тем, что их реализация может зависеть от целевой вычислительной системы. Когда программа выполняется в однопоточной среде, редукцию можно выполнять последовательно, но при выполнении в нескольких потоках она может выполняться параллельно.

Механизм Try-catch весьма популярен сегодня для обработки ошибок, но этот подход имеет конфликты с параллельным выполнением программы. Когда возникает исключение, все потоки выполнения должны быть остановлены, конвейер очищен и т. д. Также возникают сложности с сохранением программного детерминизма в случае параллельного выполнения и возникновения исключений. Для языка Cloud Sisal таких проблем не существует, поскольку в нем используется семантика «всегда завершаемых вычислений», что означает, что поток выполнения Cloud Sisal программы никогда не останавливается и всегда возвращает результирующее значение (возможно, содержащее значения «ошибка»), даже если возникают какие-либо ошибочные ситуации. Для этого, в каждом типе существует выделенное ошибочное значение, например, булевский тип состоит из значений истины (true), лжи (false) и ошибочного значения (error[boolean]). Если не оговорено обратное, и ошибочны какие-нибудь аргументы операций над встроенными типами или предопределённых функций, то результаты их тоже будут ошибочными значениями. Узнать, ошибочно ли значение выражения, всегда можно с помощью специальной операции.

Язык поддерживает аннотированное программирование [3] и конкретизирующие преобразования [4], позволяя пользователю описывать известные ему семантические свойства программы в виде формализованных комментариев. Комментарий, который начинается с символа доллара «$», называется прагмой и задаёт свойства конструкции, идущей следом (одной конструкции можно сопоставлять несколько прагм). Результат унарного выражения в прагме, к которому она относится, обозначается через одиночный символ подчеркивания «_», а арности n-арного (n>1) выражения обозначаются как «_[1]», …, «_[n]». Прагма может иметь вид «имя» или «имя = список выражений», где в выражениях списка могут принимать участие имена, видимые в месте расположения прагмы. Нераспознанные прагмы вызывают предупреждения компилятора.

Приведем некоторые примеры прагм.

Перед каждым выражением могут располагаться прагма «assert = булевское условие», которое должно быть истинным сразу после вычисления выражения. Прагмы «assert» могут располагаться в объявлениях функций как перед ключевым словом «returns» и накладывать условия на возвращаемые значения, так и перед первым формальным параметром и задавать условия на указанные в них имена формальных параметров, которые должны быть справедливы при вызове функции непосредственно перед выполнением тела функции.

Разрешается на месте булевского условия в прагме «assert» также использовать так называемое расширенное булевское условие, которое имеет вид «(all имя : булевское условие : расширенное булевское условие)» или вид «(is имя : булевское условие : расширенное

булевское условие)» и определяет область действия для заданного в нем имени. Например, расширенное булевское условие (all i : i>2 : A[i]=0) истинно, если равны нулю в массиве или потоке A все те его элементы, у которых индекс больше двух, а условие (is i : i>2 : A[i]=0) истинно, если в A существует хотя бы один нулевой элемент с индексом больше двух.

Например, в данном описании функции прагма в ее заголовке указывает, что указанная функция всегда используется для возведения в степень, которая является степенью двойки:

```
function power (//$ (is k : k>=0 & k<n : n = 2**k)
                    x : real n : integer returns real )
    if n =0 then returns 1
    elseif n%2 =0 then  returns (power(x, n/2)**2
         else returns (power(x, n-1))*x
    endif
end function
```

и поэтому она может быть эквивалентно преобразована в следующую функцию:

```
function power (//$ (is k : k>=0 & k<n : n = 2**k)
                    x : real n : integer returns real )
    if n =0 then  returns 1
    else returns (power(x, n/2)**2
    endif
end function
```

Перед каждым выражением может располагаться прагма «non_used = список значений», которые сразу после вычисления выражения становятся ненужными (в дальнейшем при исполнении программы никак не используются) и могут быть удалены из программы. Например, если, как это указано в аннотации, приведенной ниже, второй результат функции «generate» никогда не будет использоваться, то его вычисление в теле функции может быть удалено:

```
function generate ( N : integer
                    //$ non_used = _[2]
                    returns TwoDim, OneDim )
    for i in 1, N cross j in 1, N do
    returns array  [.., ..] of i * j
    end for
end function
```

3. Внутреннее представление Cloud Sisal программ

Система CPPS использует единое внутреннее теоретико-графовое представление IR (Intermediate Representation) Cloud Sisal программ, которое ориентировано на их семантическую и визуальную обработку и основано на атрибутированных иерархических графах [5]. В рамках этого представления производится сборка программы из модулей (как в интерпретаторе, так и в компиляторе) 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.

Существует несколько способов задания внутреннего представления, например, в этом качестве можно рассматривать дерево разбора транслируемой программы. Однако в нашем случае оно не удовлетворяет требованию интерпретируемости, так как это требование подразумевает наличие контекстной (семантической) информации. Вместе с этим, требование машинно-независимого параллелизма и функциональность представляемых языков программирования приводит нас к использованию потоковых графов как естественной основе структуры внутреннего представления. Таким образом, IR-граф в отличие управляющего графа, обычно используемого в оптимизирующих компиляторах для императивных языков (таких как языки C или Фортран), выражает не поток управления, а поток данных в программе. Потоковые графы обладают рядом полезных для требуемого внутреннего представления свойств, включая следующие.

1. Явно заданные информационные (семантические) связи (дуги) между операндами операций (портами вершин) делают процесс интерпретации осуществимым без дополнительных преобразований. Это влечет отсутствие побочных эффектов вычислений (ввиду отсутствия понятия переменной) — естественного свойства чисто функциональных языков.

2. Параллелизм на уровне отдельных информационно независимых операций, не зависящий от машинной архитектуры.

Вершины IR графа соответствуют выражениям программы, а дуги отражают передачи данных между портами вершин, упорядоченные множества которых приписаны вершинам в качестве их аргументов (входных портов или входов) и результатов (выходных портов или выходов). В силу свойства языка Cloud Sisal этот граф является ациклическим и не содержит двух дуг, заходящих в один и тот же вход.

Вершины обозначают операции над своими входами (аргументами), результаты которых находятся на выходах вершин. Существует, однако, специальный вид вершин, обозначающих литералы (константы) любого типа, каждая из которых имеет один выход и пустое множество входов.

Вершины бывают простыми и составными. Простые вершины (или просто вершины) не имеют внутренней структуры и представляют элементарные операции, такие, как, например, плюс или минус. Составные вершины (или фрагменты) соответствуют составным выражениям Cloud Sisal программы, таким, как, например, циклическое выражение или тело функции, и дополнительно непосредственно содержат множества вершин, соответствующих выражениям, из которых они состоят. Для каждого фрагмента F количество и типы того множества вершин M, которые непосредственно содержатся в нем, а также того множества дуг (p, q), которые существуют между его портами и портами этих вершин, задаются типом (или семантикой) составного выражения, но удовлетворяют следующему свойству: p – выход некоторой вершины из M или вход фрагмента F, а q — вход некоторой вершины из M или выход фрагмента F. Понятно, что фрагмент F помимо указанных выше вершин M и инцидентных их портам дуг, которые непосредственно содержатся в F, будет содержать и другие вершины и дуги графа, если среди вершин из M есть фрагменты, но в силу свойств языка Cloud Sisal среди них нет дуг, которые инцидентны портам фрагмента F и не являются непосредственно вложенными в F.

Предполагается, что IR представления Cloud Sisal программ демонстрируются пользователям системы наряду с их текстовыми представлениями и используются пользователями для целей визуальной отладки Cloud Sisal программ и управляемой их оптимизации.

4. Заключение

Речь в докладе шла о разработке методов и создании средств поддержки параллельного программирования нового типа, аналогов которых пока не существует и так не хватает сейчас, когда число супервычислителей все еще мало, но каждый из них можно сделать доступным по сети практически для каждого пользователя.

Создаваемая технология открывает для широкого круга прикладных программистов реальный мир супервычислений, не требуя больших инвестиций в новые компьютерные системы и позволяя разгружать существующие супервычислители от их неэффективного использования для конструирования и отладки параллельных программ.

Ее применение не только делает супервычислители, включенные в сети, доступными для широкого круга прикладных программистов, но и повышает надежность создаваемых параллельных программ, поскольку позволяет формулировать решения задач на абстрактном уровне в декларативном стиле и без привязки к конкретным вычислительным ресурсам, а также проверять создаваемые параллельные программы с помощью формальных методов.

Применение технологии также повышает эффективность использования супервычислителей за счет переноса работ программистов по конструированию и отладке программ с дорогих супервычислителей на более дешевые и привычные персональные компьютеры, а также за счет снятия необходимости прикладному программисту выполнять построение, верификацию и отладку программы для решения одной и той же задачи каждый раз заново при переходе с одного супервычислителя на другой.

Исследование выполнено за счет гранта Российского научного фонда (проект 18-11-00118).

Список литературы

1. Gaudiot J.-L., DeBoni T., Feo J. X.`, et al. The Sisal project: real world functional programming // Lecture Notices in Computer Science. – 2013. Vol. 1808. – pp. 84–72.
2. Касьянов В. Н., Касьянова Е. В. Язык программирования Cloud Sisal. – Новосибирск, 2018. – 45 с. – (Препринт/ РАН, Сиб. отд-ние, ИСИ; N181).
3. Касьянов В. Н. Аннотирование программ и их преобразование // Программирование.– 1989. № 4. – С. 3–16.
4. Касьянов В. Н. Трансформационный подход к конкретизации программ // Кибернетика. – 1989. № 6. – С. 28–32.
5. Касьянов В. Н. Иерархические графы и графовые модели: вопросы визуальной обработки // Проблемы систем информатики и программирования. – Новосибирск: ИСИ СО РАН, 1999. – С. 7-32.

Касьянов Виктор Николаевич – д.ф.-м.н., профессор,
г.н.с. Института систем информатики им. А.П. Ершова СО РАН,
профессор Новосибирского государственного университета;
630090, Новосибирск; e-mail: kvn@iis.nsk.su

Касьянова Елена Викторовна – к.ф.-м.н., доцент,
с.н.с. Института систем информатики им. А.П. Ершова СО РАН,
доцент Новосибирского государственного университета;
630090, Новосибирск; e-mail: kev@iis.nsk.su