ТРАНСФОРМАЦИОННО ОПЕРАЦИОННАЯ СЕМАНТИКА ЯЗЫКА ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

Л. В. Городняя

Аннотация

Рассматривается задача формализации языков параллельного программирования как путь к решению проблемы адаптации программ к различным особенностям используемых многопроцессорных комплексов и многоядерных процессоров. Решение этой проблемы требует разработки новых методов компиляции программ и развития средств и методов описания семантики языков программирования. Предлагаемый подход иллюстрируется на материале языка начального обучения параллельному программированию.

Введение

«Ничем не скроешь фундаментальную трудность параллелизма» – так прозвучало заявление Мартина Одерски (Martin Odersky) в его приглашенном докладе на 20-ой Международной конференции «Конструирование компиляторов», состоявшейся в марте этого года в Германии под председательством Дженса Кнопа (Jens Knoop). Тем не менее, в докладах на этой конференции рассмотрены лишь отдельные, ранее сложившиеся, средства реализации языков и систем программирования (ЯСП), пригодные для переноса в мир параллельного программирования. В их числе – «компиляция на лету», выделение чисто функциональных подмножеств и форм с однократным присваиванием, обратимая компиляция и интерпретация, транзакционная память, анализ достижимости действий, преобразования циклов над большими массивами и т.д. [1] Не смотря на многочисленные призывы и конкурсы пока не появилось общих идей по новому поколению ЯСП, в которых бы параллелизм имел самостоятельное значение, а не рассматривался бы по понятным причинам как пристройка к традиционному программированию.

И все же современное развитие ЯСП по существу ориентировано на решение задач параллельного программирования. Как правило, новые ЯСП включают в себя библиотечные модули, обеспечивающие организацию процессов, или подъязыки, допускающие многопоточное программирование [2]. Это, увы, не исключает реальную практику ручного распараллеливания ранее отлаженных обычных программ, приведения их к виду, удобному для применения производственных систем поддержки параллельных вычислений. Значительная часть таких работ носит технический характер и заключается в систематической реорганизации структур данных, изменении статуса переменных и включении в программу аннотаций, сообщающих компилятору об информационно-логических взаимосвязях. Существенным ограничением результата ручного распараллеливания является не только опасность повторной отладки алгоритма, но и его жесткая зависимость от характеристик целевой архитектуры.

Разнообразие моделей параллельных вычислений и расширение спектра доступных архитектур следует рассматривать как вызов разработчикам ЯСП, как проблему создания методов компиляции многопоточных программ для многопроцессорных конфигураций. Язык должен допускать представление любых моделей параллелизма, проявляемого на уровне решаемой задачи. Абстрактная машина, используемая при определении операционной семантики языка, естественно обобщается как семейство возможно взаимодействующих машин. При компиляции нужна типизация схем управления процессами, преобразования сети потоков и анализ информационных связей с целью верификации и вывода оптимального размещения потоков по процессам, назначения процессоров, что здесь названо трансформационно операционной семантикой языка программирования. Техника преобразований может быть основана на алгебре сетей Петри и преобразованиях графов, используемых при оптимизации программ. Методика определения семантики в данном докладе иллюстрируется на материале языка начального обучения параллельному программированию Кубик, учебный характер которого освобождает от сложностей полного определения современных ЯВУ. Примеры для учебных программ навеяны книгой [3].

Язык начального обучения параллельному программированию

Основные понятия языка программирования – схемы управления программой, образующие программу действия, вычисления и данные – при переходе к параллельному программированию претерпевают изменения, и возникает необходимость в дополнительных понятиях. На уровне языка Кубик предлагается следующая трактовка основных и дополнительных понятий:

Схема определяет варианты управления действиями в многопоточной программе. При выделении схем можно с помощью параметров выразить зависимость потоков от событий, действий, различных процессоров и изменяемых переменных. Формально схема работает как макроопределение – это открытая подстановка, выполняющая ту же работу, что препроцессоры во многих ЯСП. Разница в том, что схема включается в программу на этапе синтаксического анализа, ее параметры обязаны синтаксически соответствовать понятиям

языка и реализация схемы может быть при компиляции заменена ее более эффективным библиотечным эквивалентом. Схемы не рекурсивны, но могут содержать циклы. (На уровне языка высокого уровня пользователь сможет сам объявлять реализационные эквиваленты схем. Такая возможность может быть востребована при распараллеливании последовательных программ.)

Программа – это сеть потоков, выполняющийся в общей памяти – заданном контексте, возможно с выделением значимых событий – барьеров или сихронизаторов. Невыделенные барьеры не влияют на выполнение потоков. Одноименность барьеров символизирует одновременность прохождения потоков через них – это событие, общее для всех потоков, включающих в себя значимый барьер Программа правильна, если существует контекст, в котором синхронизация потоков корректна, т.е. не возникает недостижимости синхронизатора в одном из потоков.

Фрагмент – это укрупненное действие, объект управления при определении потоков, схем и программ, основная единица, задающая логику обработки данных в терминах параллелизма, ветвлений, многократности исполнения и распределения работ по процессорам. При работе с переменными фрагмент ограничен требованием однократности присваивания.

Поток строится из фрагментов, возможно синхронизованных по одноименным барьерам с другими потоками. Должен существовать контекст, в котором выполнимы все действия потока, выполняемого автономно. При формировании потока выбирается схема управления фрагментами, точнее порядок и способ выполнения действий.

Слой – это схема, объединяющая действия, порядок выполнения которых не задан, что рассматривается как параллельное выполнение. При необходимости действия можно упорядочить или частично синхронизовать с помощью последовательности барьеров.

Линия – это схема, задающая порядок выполнения действий как порядок их вхождения, т.е. последовательно.

Обход – это схема, которая дает возможность определять ветвящиеся процессы с помощью условий выбора или вероятности срабатывания действий. Условие считается вычисленным лишь, если оно истинно**.** При невыполненном условии недостижимы барьеры, расположенные внутри невыбранного фрагмента.

Включение – это схема, обеспечивающая многократное использование общих фрагментов потока или схемы, задаваемых непосредственно схемой или с подстановкой имен барьеров, фрагментов, переменных и процессоров, а также вызовы функций, включаемых в динамике.

Назначение – это схема, позволяющая явно распределять работу многопоточной программы по процессорам.

Итерация обеспечивает многократность выполнения фрагмента. Можно представлять неограниченную итерацию в расчете, что ее ограничат смежные потоки с помощью синхронизации или ограничение длительности.

Сложные фрагменты программы строятся с помощью схем из более простых и из базовых действий, которые задают точки и методы информационной обработки, а также определяют информационные взаимодействия фрагментов программы друг с другом через общую память. Для базовых действий известны критерии их выполнимости, при нарушении которых действие считается не выполнявшимся.

Вычисление выражения – это простейшее действие. Любой процессор содержит память для текущего результата его работы. Возможна реорганизация структур данных, при которой перемещаются их элементы из левой в правую часть с учетом определения структуры данных правой части и фильтра, выделяющего элементы из левой части. Результат – пара из полученной структуры и остатка преобразуемой. Критерий выполнимости – наличие данных, достаточных для выполнения реорганизации. Например, нельзя в вектор разместить объекты из пустой очереди. Действие «подождет», когда в очереди появятся данные.

Контекст – это память для общедоступных данных многопоточной программы. Она может быть неоднородной по времени доступа. Контекст задан как структура данных, связывающая имена со значениями констант и переменных, определениями функций или схем управления. Контекст можно уточнять заданием новых связей. Все имена уникальны, т.е. нет локализации по блокам, хотя процессоры имеют свою защищенную память. Фактически связывания имен – это присваивания, но с сохранением старых значений, которые можно достать при необходимости специальными функциями.

Основные методы представления вычислений связаны с использованием неявных циклов, позволяющих избежать выписывания однотипных схем над стандартными структурами данных типа многомерных векторов. Так, например, результат операции над скалярами может быть распространен на произвольные однородные структуры данных. Список операций, допускающих распространение, определен реализацией. Обычно это арифметические операции (+ - * /). В языке Кубик этот метод используется более широко, распространен на технику применения функций, что повышает лаконизм выражений.

Из бинарных операций можно конструировать фильтры. Результат фильтрации исчезает из аргумента – он переносится в другую структуру данных или сохраняется как значение. Структура из фильтров дает структуру из результатов их применения к одному и тому же аргументу.

Результативность вычислений можно повышать с помощью специально устроенных данных – так

называемых «рецептов» – это отложенное исполнение фрагмента программы, его замыкание, т.е. пара из действия и контекста. Барьеры в рецептах не действуют.

В определении языка Кубик предпринята попытка разделить последовательность вычислений и последовательность размещения их результатов в элементах памяти. Последовательность вычисления не обязана совпадать с последовательностью размещения вычисленных значений. Основные структуры данных – это очереди, элементы которых доступны последовательно и размещены рассредоточено, новые добавляются в хвост очереди, и вектора с обычным индексированием элементов, размещенных в соседних регистрах, допускающих произвольный доступ. Скобки символизируют порядок размещения, а знаки – порядок вычисления. Т.о. «( )» и «;» символизирует последовательность, а « [ ]» и «,» – произвольный порядок или доступ.

(A, B, C,…) – очередь из элементов, порядок вычисления которых не определен.

(A; B; C;…) – очередь из элементов, вычисляемых в порядке записи.

[A; B; C;…] – вектор из элементов, вычисляемых в порядке записи.

[A, B, C,…] – вектор из элементов, порядок вычисления которых не определен.

В дальнейшем, при разработке учебного языка высокого уровня, ориентированного на параллелизм, предполагается исследовать возможность языковых средств, обеспечивающих при работе с функциями пространство аргументов, подобное пространству итерации, а также, типизацию действий и схем, поддерживающую вычисления, реорганизацию памяти, изменение памяти и кроме того, учет длительности действий, мемоизацию, конструирование потоков, динамический учет производительности и др.

Трансформации

Трансформационная семантика обеспечивает сведение конструкций языка программирования к его базовым средствам, что позволяет упростить операционную семантику, а также выбрать реализационное ядро системы программирования при его экспериментальной раскрутке. Похожая техника применяется при сведении грамматик языка к форме, удобной для автоматизации построения анализатора, и при оптимизации программ.

Направление преобразований программ обычно связано с определенными критериями применимости и оптимальности, учитывающими результаты анализа логических и информационных связей. При организации параллельных процессов такие критерии обладают спецификой, отражающей особенности эксплуатации многоядерных архитектур. В частности возрастает роль учета скоростей доступа к разнородной памяти и статического планирования загрузки процессоров наряду с обеспечением обратимости обработки данных и динамического управления производительностью вычислений. Поддержка такой семантики вычислений выходит за границы традиционных решений по реализации языков высокого уровня

Задача трансформационной семантики – сведение программы к нормализованной форме, удобной для интерпретации программ или генерации исполнимого кода. В случае многопоточных программ преобразование сети потоков нацелено на сведение к однородной системе потоков, однозначно отображаемых на заданный комплекс процессоров – размещение потоков по процессам или назначение процессоров для выполнения потоков.

Такое требование можно выразить формулой:

[(b0: ; b1: ; … bK:), p1!(b0: d0i; …), … pN!(b0: d0j; ...)],

где i и j обозначают принадлежность потоку.
, – параллельное или одновременное исполнение,
; – последовательное исполнение,
! – назначение процессора,
b0:, b1:, … – барьеры,
p1, pN, … – процессоры,
d0i, d0j, … – действия,
(b0:; b1:; … bK:) – шкала событий/барьеров,
pM!(b0: d0i; …) – программа (последовательность) действий для процессора pM.

В такой, как бы «причесанной» форме все потоки начинаются с барьеров, и общая шкала событий упорядочена так, что последовательность событий потока ей не противоречит. Можно считать, что процессоры включаются сами. Шкала событий содержит списки ожидающих потоков. Действия, выполняемые

процессорами, соотнесены с их исходными потоками.

Достаточно простые преобразования сети потоков позволяют варьировать схемы потоков и многие конструкции языка программирования сводить к взаимодействию простых потоков:

{1}Здесь и далее в формулах малыми латинскими буквами обозначены барьеры, а большими – действия.

{2}Потеря эффективности и влияние информационной связности потока C могут быть устранены реализацией в стиле так называемых «рецептов», принятых при организации «ленивых» вычислений (lazy evaluation).

(A;B) ↔ (a:A; b:B) {1}– Расстановка – стирание барьеров.

(a:A; b:B) ↔ [[a:A, b:B], (a: ; b:)] – вынесение последовательного управления в отдельный поток – восстановление последовательности действий..

(a: A; b: B) ↔ [(a: A; b:), (b: B)] – разрез последовательности – перенос «хвоста», если нет локальной информационной зависимости между A и B.

(a: A; b: B) ↔ [a: A, b: B] – разбиение последовательности на потоки – сплющивание линии в слой, если A и B информационно не связаны.

[a: A, b: B] ↔ { (a: A; b: B) | (b: B; a: A)} – слияние потоков в одну последовательность – вытягивание слоя в линию. Выбор варианта требует учета последовательности барьеров в других потоках и общей шкале событий. Критерий оптимальности – объем выполнимых вычислений.

((a: A; b: B) ; c: C) ↔ (a: A; (b: B ; c: C)) ↔ (a: A; b: B ; c: C)

[[a: A, b: B] , c: C] ↔ [a: A, [b: B , c: C]] ↔ [a: A, b: B, c: C]

([A,B];C) ↔ [(A;C), (B;C)] – исключение слияний – слияние совпадающих продолжений.{2}

[(A;B),C]↔ ([A,C]; [B,C]) – распределение параллельного потока – слияние совпадающих потоков

[[a: A], [a: B]]↔ [a: [A, B]] – варьирование числа одновременных потоков.

Обратимость преобразований и чувствительность их результата к информационным связям между фрагментами программы требуют формализации критериев применимости трансформаций и выбора подходящего варианта. Можно констатировать, что выяснение информационной связанности действий B и A сводится к проверке существования контекста, в котором различны результаты программы C при изменении порядка вычислений B и A.

[(A; B), C] ≠ [(B; A), C]

Для динамического анализа хода вычислений можно предложить операцию «Выполнялось?», реализуемую не как базовое средство, а сведением к пометке барьером позиции сразу вслед за проверяемым действием.

((? Выполнялось X); Y) ↔ [[X ; x:] , x: Y] – проверка, выполнялось ли действие X. Y выполнится лишь если X выполнялось

Аналогично из числа базовых средств можно вывести ветвления, циклы и вызовы функций, реализуя их средствами синхронизации потоков:

(if A then B) ↔ [(A;b:);(b:B)] – сведение обхода к синхронизации и обратно.

При отладке формируется ряд контекстов, на которых демонстрируются отдельные свойства фрагментов, из которых собирается полная программа. Это контексты для отдельных потоков, для пар синхронизованных потоков, для интегрированной из потоков программы, а кроме того контексты для удостоверения наличия-отсутствия информационных связей между фрагментами.

Возможны пользовательские преобразования схем управления процессами, что позволит не только минимизировать «ручную» оранжировку распараллеливаемых программ, но и даст основу для формирования библиотек преобразования схем программ.

Абстрактная машина

Операционная семантика языка программирования базируется на определении абстрактной машины, которая в случае многопроцессорных конфигураций естественно становится конструкцией из абстрактных процессоров. В основном определение команд абстрактной машины наследует решения SECD-машины, предложенные Лэндиным и описанные в книге Хэндерсона [4].

АМ = <Регистры, [АП-1, … АП-n], СК>

АП-i = <Пам-i, СК-i>

Регистры = < стек результатов,
структура контекста,
сеть потоков/процессоров,
очередь барьеров>

СК – система команд, включающая в себя универсальные для всех процессоров команды, типичные для языков управления заданиями:

Stop – завершить текущий процесс.

Kill (i, …) – завершить перечисленные процессы, но не текущий.

Off/On (i, …) – включить/выключить указанные процессы (приостановить/возобновить).

Read /Print – обмен данными с общими файлами из контекста.

Send/Receive D, (i, …) – разослать/получить данные D указанным процессам/от указанных процессов.

Init (i,…) – запуск процесса.

Lock/Unlock (v, …) – блокировка памяти – захват переменной из общего контекста на время ее обработки для обеспечения транзакции.

Move (dl, dr) – безотходная реорганизация данных.

Load (rl, rr) – передача данных между регистрами внутри АП.

Wait (time) – объявление времени ожидания, пауза.

Barier (b:) – объявление, что барьер достигнут.

Set (Name, val_or_func) – именование данного: значения или функции.

Call (Name) – вызов функции или тела цикла.

Кроме того, на уровне абстрактной машины поддержаны обычные бинарные операции над данными и дополнительные операции, возникающие при реализации системы программирования.

При подготовке примеров для демонстрации языка приняты следующее ограничения:

- по мере выполнения действий корректируется список барьеров: удаляется ссылка на поток для достигнутого барьера или пополняется список барьеров и список выполняющихся действий при рекурсиях;
- если «свободных» барьеров нет, то программа останавливается;
- индексируются имена барьеров при повторных вызовах итераций и рекурсий;
- взаимодействия потоков выполняются только через синхронизацию;
- одновременно исполняемые фрагменты могут обмениваться данными через общую память.
- при взаимоисключении каждый вариант работает в своей копии контекста
- завершение варианта посылает Kill остальным вариантам;
- поддерживается список восстановления многократно выполняемых фрагментов.

Заключение

В определении языка Кубик представлена базовая модель организации взаимодействующих процессов, позволяющая показать природу трудностей параллельного программирования и простейшие пути решения известных проблем. Подробное описание языка готовится к публикации в журнале «Компьютерные инструменты в образовании». Программа факультативного курса для школьников по параллельному программированию на базе этого языка изложена в материалах секции «Информатика образования» Ершовской конференции по информатике в июне 2011 года.

Следующий шаг – разработка языка высокого уровня параллельного программирования, приспособленного к ознакомлению студентов с более сложными моделями вычислений и с методами верификации программ, без которых надежность параллельного программирования весьма проблематична. Для нужд этого шага потребуется более строгая формализация семантики в трансформационно операционном стиле, что и опробовано при реализации макетного образца системы программирования для языка Кубик. Представленный в докладе подход к определению языка параллельного программирования предназначен для поддержки экспериментов по разработке новых языков, ориентированных на учебно-исследовательские проекты в области создания распределенных информационных систем.

ЛИТЕРАТУРА:

1. LNCS 6601. Jens Knoop Compiler Construction. 20th International Conference, CC 2011. Held as Part of the Joint European Conferences on Theory and |Practice of Software, ETAPS 2011 Saarbrücken, Germany, March 26 – April 3, 2011. Springer. 330 p.
2. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления.– СПб.: БХВ-Петербург, 2002. – 608 с.
3. Хоар Ч. Взаимодействующие последовательные процессы. М.: Мир, 264 с.
4. П. Хендерсон. Функциональное программирование. Применение и реализация. М.:Мир. 1983. 349 с.