ТЕХНОЛОГИЧЕСКИЕ АСПЕКТЫ СОЗДАНИЯ, ПРЕОБРАЗОВАНИЯ И ВЫПОЛНЕНИЯ ФУНКЦИОНАЛЬНО-ПОТОКОВЫХ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

А.И. Легалов, И.В. Матковский, М.С. Кропачева, Ю.В. Удалова, В.М. Васильев

Введение

Минимизация времени выполнения программы является одним из основных критериев в процессе разработки параллельного программного обеспечения (ППО). Для этого необходимо учитывать особенности архитектуры используемой параллельной вычислительной системы (ПВС). Ориентация на конкретную архитектуру в свою очередь требует учета специфических ограничений, преодоления различных ресурсных конфликтов, возникающих между взаимодействующими процессами. Необходимость одновременного учета особенностей решаемой задачи, специфики вычислительных ресурсов, требований высокой производительности и балансировки процессов затрудняет разработку, отладку, верификацию, тестирование и сопровождение ППО. Поэтому в настоящее время актуальны подходы, ориентированные на архитектурно-независимую (АН) разработку параллельного программного обеспечения.

Особенностью АН разработки ППО является программирование без учета специфики вычислительных ресурсов. Это возможно в том случае, если параллелизм будет рассматриваться на уровне элементарных операций, вычислительные ресурсы будут считаться неограниченными, а процесс привязки разработанной программы к реально используемым вычислительным ресурсам будет осуществляться после того, как будут выполнены отладка, верификация и тестирование архитектурно-независимой параллельной программы.

Применение данного подхода предъявляет специфические требования к языкам программирования, которые должны поддерживать неявное управление параллельными вычислениями. Данным требованиям удовлетворяют функциональные языки параллельного программирования. Описывая алгоритмы как отношения между функциями, они обеспечивают требуемое описание параллелизма решаемой задачи. Выполнение функций по готовности данных позволяет использовать неявное управление вычислениями. Такие языки считаются перспективным, но в настоящее время их развитие сдерживается рядом факторов. В частности, широко распространено мнение, что использование функциональных языков непривычно для разработчиков. Считается, что проще расширить уже изученный последовательный язык программирования средствами организации параллельных потоков и процессов. Другим сдерживающим фактором является отсутствие эффективных методов и алгоритмов привязки АН параллельного ПО к архитектурам реальных ПВС.

Несмотря на имеющиеся трудности данное направление постепенно развивается, формируя фундамент для последующих исследований методов преобразования архитектурно-независимых параллельных программ в архитектурно-зависимую форму. В рамках его нами предлагается функционально-потоковая парадигма параллельного программирования, базирующаяся на использовании неограниченного параллелизма, в виртуальной машине с «бесконечными» ресурсами. На основе этой парадигмы разработаны функционально-потоковая модель параллельных вычислений [1] и язык программирования Пифагор [2]. Для использования данного языка разработана система программирования, включающая:

- Транслятор с функционально-потокового языка и дополнительные системные утилиты, используемые для порождения выходного представления, используемого во время верификации, отладки и выполнения [3, 4];
- событийный процессор, обеспечивающий выполнение ФПП программ, представленных порожденным выходным представлением [5, 6];
- средства отладки ФПП программ [7, 8].

Транслятор переводит исходный текст программы в промежуточное представление. При этом допускаются различные оптимизационные преобразования. Для обеспечения их эффективного выполнения промежуточное представление программы разделено на следующие слои [5]:

- слой реверсивного информационного графа (РИГ);
- слой данных;
- слой управляющего графа (УГ);
- слой управляющих автоматов.

Формирование слоев осуществляется во время трансляции для каждой функции и состоит из следующих этапов:

- построение РИГ транслируемой функции и частичное заполнение слоя данных используемыми константами;
- создания управляющего и автоматного слоев функции на основе РИГ.

Полученные структуры определяют промежуточные представления разработанных функций. Перед выполнением осуществляется их компоновка в единый исполняемый модуль. Интерпретатор получает на вход единый исполняемый модуль и обеспечивает выполнение функционально-потоковых параллельных программ.

Отладка и верификация являются важнейшими этапами жизненного цикла программного обеспечения. Разработанные к настоящему моменту средства позволяют анализировать корректность ФПП программы и выполнять отладку в различных режимах, обеспечивающих изменение методов обхода информационного графа функции. Реализация нескольких способов обхода РИГ позволяет более гибко проводить отладку ФПП программ [7].

Особенности процесса трансляции ФПП программ

Транслятор ориентирован на обработку текстовых файлов, каждый из которых может содержать множество функций, написанных на языке Пифагор. В результате трансляции для каждой функции создается промежуточное представление РИГ в виде композиции классов, которое сохраняется в репозитории функций в текстовой форме. Выбор текстового представления для описания РИГ обусловлен тем, что формирование на его основе внутреннего представления в памяти компьютерной системы может быть легко выполнено с помощью простых транслирующих программ. Помимо этого разработчик может легко читать и анализировать оттранслированные функции, рассматривая данную форму как аналог языка ассемблера.

В качестве примера рассмотрим трансляцию функции вычисления факториала. На языке Пифагор она выглядит следующим образом:

```
math.fact << funcdef n {
  (-1, 1, {(n,(n,1):-:math.fact):*} ):[((n,0):[<,=,>]):?]:. >>return
}
```

Функция возвращает значение -1 в случае некорректного отрицательного аргумента. После трансляции формируется промежуточное представление в текстовой форме:

```
External
      0     math.fact
Local
      0     -1
      1     1
      2     1
      3     {1}5
      4     0
id    delay       operation   links             positions
0     0           arg                           pos 1 22 1 23
1     1           (---)       0 loc:2           pos 5 10 5 15
2     1           :           1 -               pos 5 15 5 16
3     1           :           2 ext:0           pos 5 17 5 18
4     1           (---)       0 3               pos 5 7 5 28
5     1           :           4 *               pos 5 28 5 29
6     0           (---)       loc:0 loc:1 loc:3 pos 2 3 6 4
7     0           (---)       0 loc:4           pos 6 7 6 12
8     0           [---]       < = >             pos 6 13 6 20
9     0           :           7 8               pos 6 12 6 13
10    0           (---)       9                 pos 6 6 6 21
11    0           :           10 ?              pos 6 21 6 22
12    0           [---]       11                pos 6 5 6 24
13    0           :           6 12              pos 6 4 6 5
14    0           :           13 .              pos 6 24 6 25
15    0           return      14                pos 6 29 7 1
```

Оно содержит ссылки на внешние функции и константы, внутренние константы, а также реверсивный информационный граф заданной функции. Область описаний внешних ссылок начинается с ключевого слова External и содержит список строк, в каждой из которых задан дескриптор (номер) внешней ссылки и имя внешней ссылки. На нулевой позиции всегда располагается ссылка на саму функцию. Это позволяет использовать обращение к самой себе в случае рекурсивных вызовов.

Область локальных констант начинается с ключевого слова Local и содержит список констант, используемых в ходе выполнения функции. Каждой константе соответствует свой дескриптор (номер), который при выполнении программы обеспечивает доступ к соответствующим данным. Наряду с числовыми и символьными данными в этой области хранятся константы, определяющие параметры задержанных списков. Каждая из таких констант хранит номер задержанного списка, а также дескриптор узла РИГ, возвращающего вычисленное значение из данного задержанного списка (подобный узел существует в любом задержанном списке). Назначение и использование задержанного списка представлены при описании модели вычислений и языка программирования [1, 2]. Область описания РИГ содержит список вершин, и связей между ними, локальными константами и внешними ссылками. Данная область начинается с заголовка, описывающего содержание каждой строки:

```
id    delay operation   links positions
```

Столбец id задает дескриптор (номер) вершины РИГ. В столбце delay указывается номер задержанного списка, в котором расположена соответствующая вершина. Если вершина РИГ не находится в задержанном списке, в данный столбец заносится ноль. В столбце operation размещается операция, выполняемая в соответствующей вершине РИГ. Столбец links указывает на информационные связи вершин. В нем задается список ссылок на источники данных для текущей вершины. Источниками информации могут быть: внешние ссылки, локальные константы, предопределенные символы и узлы РИГ. Каждый из этих источников данных по-своему идентифицируется в качестве элемента списка связей. Различие в заданиях связей в дальнейшем определяет обращение к разным областям памяти в СП.

Указание на внешнюю ссылку задается в следующем формате: ext:<символическое_имя_внешней_ссылки>. Для локальной константы используется формат: loc:<значение_константы>. Предопределенные символы, обычно связанные со знаками различных операций, задаются своими значениями, например: +, -, * и другие. Если источником операндов служит другая вершина РИГ, то в качестве связи задается целое число, равное дескриптору этой вершины.

В столбце positions задаются координаты, определяющие местоположение каждого из операторов РИГ в исходном тексте функции. Эта информация используется во время отладки, позволяя визуализировать на исходном тексте функции результаты вычислений и точки останова.

Формирование управляющего графа

Порожденный транслятором РИГ используется не только во время вычислений для организации доступа к данным. Он также позволяет построить управляющий граф (УГ), задающий управление вычислением функции. Для этого предназначена специальная утилита, которая порождает управление вершинами РИГ по готовности данных. В дальнейшем возможна оптимизация УГ, позволяющая сократить число связей и сигналов. Описание УГ сохраняется в репозитории в текстовом виде. Для ранее приведенной функции вычисления факториала он будет выглядеть следующим образом:

```
math.fact
id    delay       automat     inode       links
0     0           arg,0       0           links:1,1;4,1;7,1;
1     1           (---),0     1           links:2,1;
2     1           :,0         2           links:3,1;
3     1           :,0         3           links:4,2;
4     1           (---),0     4           links:5,1;
5     1           :,0         5
6     0           (---),0     6           links:13,1;
7     0           (---),0     7           links:9,1;
8     0           [---],0     8           links:9,2;
9     0           :,0         9           links:10,1;
10    0           (---),0     10          links:11,1;
11    0           :,0         11          links:12,1;
12    0           [---],0     12          links:13,2;
13    0           :,0         13          links:14,1;
14    0           :,0         14          links:15,1;
15    0           return,0    15
Signals: (Number, Node, Input)
0           1           2
1           2           2
2           3           2
3           5           2
4           6           1
5           6           2
6           6           3
7           7           2
8           8           1
9           8           2
10          8           3
11          11          2
12          14          2
Dynamic links: (Number, Delay list, Node)
0           1           5
```

Первая строка формируемого представления служит для идентификации УГ. В ней задается имя выполняемой функции. Следующая строка носит справочный характер, описывая названия столбцов управляющего графа, узлы которого располагаются в виде списка строк непосредственно за ней. Столбец id используется для задания дескриптора (номера) узла УГ. В столбце delay указывается номер задержанного

списка, в который входит тот или иной узел ИГ. Если узел не входит ни в один из имеющихся задержанных списков, то в данном поле стоит значение, равное нулю. В столбце automat задается тип автомата, используемого для управления узлом, и через запятую указывается его начальное состояние. Обычно при начальном построении автоматы находятся в начальном (нулевом) состоянии. Они могут измениться в ходе оптимизации управляющего графа с использованием соответствующих утилит. В столбце inode указывается узел РИГ, непосредственно связанный с данной вершиной УГ. При первоначальной генерации УГ количество его вершин совпадает с числом вершин РИГ, и вершины обеих графов находятся в однозначном соответствии. Однако в ходе оптимизации УГ, возможно изменение этого соответствия. Могут также измениться и автоматы, размещенные в вершинах УГ. Столбец links указывает на список управляющих связей, направленных от источников управляющих сигналов к их приемникам. В отличие от РИГ, в УГ направление связей определяется таким образом, чтобы каждый источник данных инициировал выполнение процессов в связанных с ним приемниках. Описание каждой связи содержит номер узла-приемника и номер входа в нем.

Область управляющих сигналов, следует после описания вершин УГ, отделяясь от него заголовком:

```
Signals: (Number, Node, Input)
```

Представленные в УГ сигналы, обеспечивают начальную инициализацию вычислений, определяя готовность к использованию констант РИГ. Именно эти сигналы, наряду с сигналом, информирующим о появлении аргумента функции, определяют начало вычислений. Каждый сигнал имеет свой дескриптор (номер) и задает узел-приемник, а также номер входа в этом узле.

Для перемещения между узлами РИГ в ходе вычислений нераскрытых задержанных списков необходима дополнительная управляющая информация. Она задается в области задержанных связей. Описание каждой связи содержит номер задержанного списка и узел РИГ, являющий источником результата, порождаемого раскрытым задержанным списком. Данная область начинается с заголовка:

```
Dynamic links: (Number, Delay list, Node)
```

Параллельная событийная машина

Выполнение функционально-потоковых программ осуществляется специальным интерпретатором, состоящим из множества событийных процессоров (СП). Каждый из этих процессоров осуществляет обработку только одной функции, запускаемой в отдельном потоке. Выполнение операций внутри функции в настоящий момент осуществляется последовательно за счет изменения состояния вершин УГ, которые инициируют вычисления в вершинах РИГ по готовности данных. Сигналы, определяющие начальную разметку управляющего графа, задают события, связанные с наличием данных, формируемых в ходе вычислений. Каждое событие содержит номер узла, которому передается управление. Получение события активизирует управляющий узел, переводя его в новое состояние, определяемое как функцией связанного с ним узла информационного графа, так и своим текущим состоянием.

Функционирование СП осуществляется следующим образом. Исходные сигналы, определяемые начальной разметкой УГ, загружаются в очередь событий, из которой они передаются обработчику событий в соответствии с дисциплиной обслуживания. В простейшем случае это может быть дисциплина FIFO. Обработчик событий анализирует параметры сигала и выбирает указанный в нем узел управляющего графа. На основе анализа состояния узла УГ он может обратиться к связанной с ним вершине информационного графа за кодом выполняемой операции. В случае, когда операция обработки данных должна быть выполнена, происходит обращение к обработчику данных, который осуществляет требуемые функциональные преобразования и сохраняет промежуточные результаты. После обработки данных управляющий узел переходит в новое состояние и при необходимости формирует сигнал, передаваемый следующему узлу.

Перед началом интерпретации производится компоновка программы из отдельных функций, размещенных в репозитории. Компоновщик проверяет наличие всех элементов, перечисленных в РИГ в разделе внешних ссылок. В том случае, если какой-то из этих элементов отсутствует, интерпретация объявляется невозможной. Для каждого из найденных элементов также производится компоновка; все найденные в процессе компоновки РИГ и УГ сохраняются в памяти, в специальной карте – откуда легко могут быть извлечены при необходимости по имени.

Карту загруженных компоновщиком РИГ и УГ хранит центральный менеджер. Процесс интерпретации начинается с создания первого (начального) СП; ему передаются ссылки на РИГ и УГ той функции, с которой интерпретация должна начаться. СП формирует автоматный слой и слой данных исполняемой функции, после чего начинает анализ управляющего графа. Автоматы, связанные с вершинами УГ, которым в РИГ соответствуют константы, будут изначально находиться в состоянии порождения новых событий; эти события, следуя по связям УГ, активируют следующие автоматы. Процесс передачи событий по связям продолжается либо до тех пор, пока не будет обработана вершина, соответствующая в РИГ вершине «return» – в таком случае функция считается выполненной, либо до тех пор, пока очередь событий не опустеет – тогда СП перейдет в спящий режим, предварительно оповестив об этом центральный менеджер.

Отладка функционально-потоковых параллельных программ

Методы отладки и верификации ФПП программ [7] позволяют использовать ряд специфических приемов, определяемых особенностью функционально-потоковой модели вычислений, в которой отсутствует понятие переменной, выполняемые функции связаны напрямую, а данные имеют динамическую типизацию. Модель вычислений определяет ФПП программу как информационный граф, в котором вершины являются операторами, а дуги определяют связи между ними. Параллельное выполнение основано на том, что одновременно выполняться могут только те операторы, между которыми нет информационных связей. Чтобы оператор мог быть вычислен, необходимо выполнение всех тех предшественников, которые связаны с ним. Отладка ФПП программы может проходить в одном из четырех режимов: пошаговой отладки, отладки слоев, отладки ветвей и проверки формул.

Заключение

Разработанные инструментальные средства, обеспечивают трансляцию, выполнение, отладку и верификацию функционально-потоковых параллельных программ в интерпретаторе, состоящем из множества событийных процессоров. Предлагаемые архитектурные решения позволяют объединить событийный процессор и подсистему отладки в единый комплекс.

Работа выполнена при поддержки Федеральной целевой программы «Научные и научно-педагогические кадры инновационной России». Проект № 14.A18.21.0396.

ЛИТЕРАТУРА:

1. А.И. Легалов, Ф.А. Казаков, Д.А. Кузьмин, А.И. Водяхо. Модель параллельных вычислений функционального языка. // Известия ГЭТУ, Сборник научных трудов. Выпуск 500. Структуры и математическое обеспечение специализированных средств. – С.-Петербург, 1996. - С. 56-63.
2. А.И. Легалов. Функциональный язык для создания архитектурно-независимых параллельных программ. // Вычислительные технологии, № 1 (10) – 2005. - С. 71-89.
3. А.И. Легалов, О.В. Непомнящий, А.В. Редькин, И.В. Матковский. Транслятор с функционально-потокового языка параллельного программирования. / Свидетельство о государственной регистрации программы для ЭВМ № 2011615829. Зарегистрировано в реестре программ 27 июля 2011 года.
4. А.И. Легалов, О.В. Непомнящий, А.В. Редькин, И.В. Матковский. Генератор управляющего графа функционально-потоковых параллельных программ. / Свидетельство о государственной регистрации программы для ЭВМ № 2011615830. Зарегистрировано в реестре программ 27 июля 2011 года.
5. А.И. Легалов, Г.В. Савченко, В.С. Васильев. Событийная модель вычислений, поддерживающая выполнение функционально-потоковых параллельных программ. / Системы. Методы. Технологии. № 1 (13). - 2012. - С. 113-119.
6. А.И. Легалов, О.В. Непомнящий, А.В. Редькин, И.В. Матковский, В.А. Хабаров. Интерпретатор функционально-потоковых параллельных программ. / Свидетельство о государственной регистрации программы для ЭВМ № 2011615832. Зарегистрировано в реестре программ 27 июля 2011 года.
7. Ю.В. Удалова, А.И. Легалов, Н.Ю. Сиротинина. Методы отладки и верификации функционально-потоковых параллельных программ. // Журнал Сибирского федерального университета. Серия «Техника и технологии». Апрель 2011 (том 4, номер 2) – С. 213-224.
8. А.И. Легалов, О.В. Непомнящий, Ю.В. Удалова, И.В. Матковский, В.А. Хабаров. Графический отладчик функционально-потоковых параллельных программ на языке Пифагор под ОС Linux. / Свидетельство о государственной регистрации программы для ЭВМ № 2012612190. Зарегистрировано в реестре программ 28 февраля 2012 года.