Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный университет»

Кафедра системного программирования

Ковалёв Дмитрий Александрович

Внутреннее представление программ для Transport Triggered Architecture

Курсовая работа

Научный руководитель:
Вербицкая Е. А.

Санкт-Петербург
2015

Оглавление

Введение

Параллелизм на уровне команд (ILP) - это способность процессора исполнять параллельно несколько операций. Для реализации ILP процессор должен располагать несколькими *функциональными устройствами (ФУ)*. Общий вид ILP-архитектуры представлен на Рис. 1.

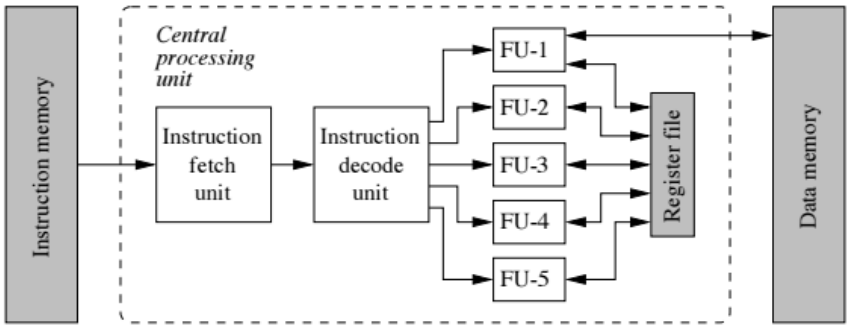


Рис. 1: Общий вид ILP-архитектуры [1]

В настоящее время выделяют три основные ILP-архитектуры: sequential, dependence, independence.

Sequential архитектура предназначена для программ, в которых не содержится информация о зависимостях между операциями. Процессор должен выявлять эти зависимости и переопределять порядок операций, чтобы добиться увеличения скорости вычислений. Главным представителем sequential архитектуры являются *суперскалярные* процессоры.

Dependence архитектура исполняет программы, в которых, помимо операций, содержится информация о зависимостях между ними. Задачей процессора является нахождение операций, готовых к выполнению, и распределение ресурсов между ними.

Independence архитектура исполняет программы, в которых явно указана информация о том, какие операции могут выполняться независимо друг от друга. То есть, задача нахождения параллельности в программе целиком возлагается на компилятор. Примером такой архитектуры может служит *Very Long Instruction Word (VLIW)* архитектура. Её особенностью является то, что компилятор не только указывает, какие операции могут исполняться независимо, но и распределяет их между ФУ.

Transport Triggered Architecture относится к independence архитектурам. Она схожа с архитектурой VLIW, но, в отличие от неё, требует от компилятора, помимо распределения ФУ, указания каналов, по которым будут перемещаться данные во время выполнения операций.

Внутреннее устройство TTA (Рис. 2) отличается от стандартной ILP-архитектуры [1]. ФУ не подключены напрямую к файлу регистров при помощи отдельных каналов. Вместо этого используется сеть с внутрисистемной коммутацией (interconnection network), которая состоит из шин данных и сокетов, устройств для подключения блоков процессора к шинам. Обмен данными между ФУ может проходить напрямую через сеть, без использования файла регистров.

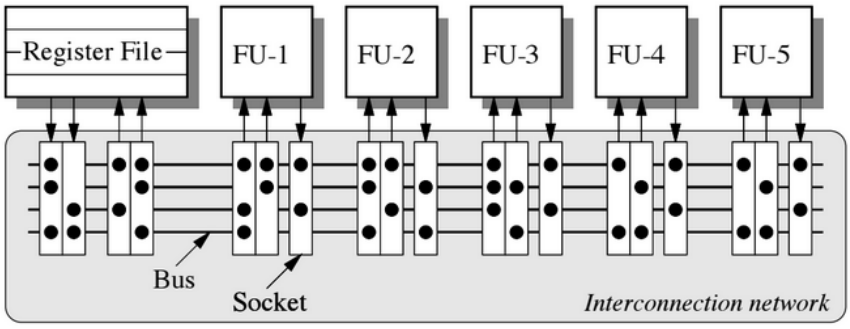


Рис. 2: Устройство TTA [1]

Основной особенностью TTA является то, что вместо команд, которые должен исполнять процессор, в инструкциях указываются перемещения данных внутри него, а выполнение операций является побочным результатом перемещений. Выполнение операции с n аргументами происходит следующим образом: в регистры операндов соответствующего ФУ помещаются $n-1$ аргументов, после чего последний аргумент помещается в специальный регистр-триггер, что вызывает начало вычисления.

Внутреннее представление (ВП) - это структура данных, удобная для хранения и обработки информации о программе.

Наиболее распространенным ВП является *граф потока управления (CFG)* (Рис. 3). Основой такого представления являются базовые блоки - последовательности команд, внутри которых не происходит передачи потока управления. Из базовых блоков и предикатов, контролирующих поток управления, строится искомый граф.

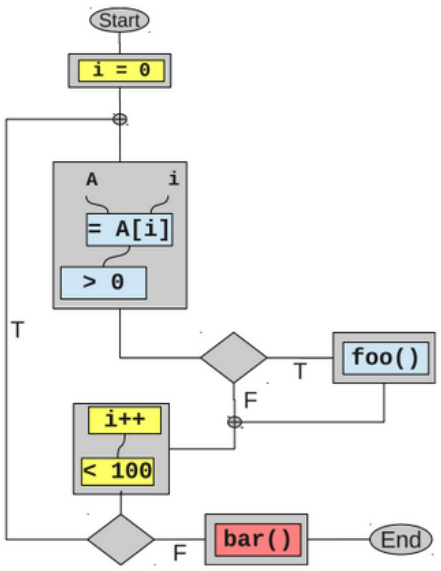


Рис. 3: Граф потока управления [5]

Граф потока управления подходит для использования в компиляторах, которые используются для работы с суперскалярными процессорами, так как последние обладают мощными аппаратными средствами для динамического анализа подобного представления. Примером мо-

жет служить предсказатель переходов, который с точностью более 95% [5] определяет, к какому базовому блоку перейдет поток управления, и загружает соответствующие команды на конвейер процессора.

К сожалению, данное ВП не подходит для TTA, так как она относится к семейству independence архитектур, то есть, не имеет инструментов динамического анализа. Стандартные средства статического анализа, используемые для подобных архитектур, так же не могут обеспечить высокой эффективности работы TTA, так как особенности устройства графа потока управления мешают им находить независимые операции в программе, снижая уровень ILP [5].

В данной работе исследовано альтернативное ВП, позволяющее эффективно использовать TTA процессор, приведено описание реализации данного ВП и его интерпретатора.

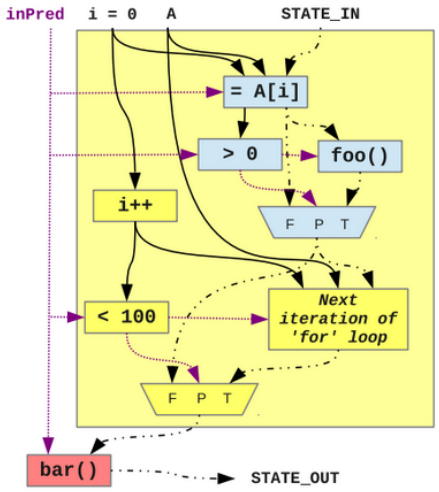
1. Постановка задачи

Целью данной работы является создание и апробация внутреннего представления программ, подходящего для эффективной работы с TTA процессорами. Для её осуществления были поставлены перечисленные ниже задачи.

- Изучить альтернативные внутренние представления программ.
- Реализовать структуру данных, описывающую необходимое представление.
- Реализовать интерпретатор, с помощью которого можно апробировать полученную структуру.

2. Обзор

В статье Ali Mustafa Zaidi и David Greaves [5] было предложено ВП, обеспечивающее высокую степень параллельности исполнения на архитектурах, не обладающих инструментами для динамического распределения операций. Это ВП получило название *Value State Flow Graph (VSFG)* (Рис. 4). Операции в VSFG не разделяются на базовые блоки, и, следовательно, отсутствует понятие передачи потока управления от одного блока к другому. Вместо этого отображаются лишь зависимости между операциями и порядок выполнения операций, имеющих побочные эффекты. Поток управления представлен неявно при помощи вершин-предикатов. Циклы и вызовы функций являются вложенными подграфами, но со стороны родительского графа выглядят как обычные вершины-операции.




```
for (i = 0; i < 100; i++)
    if (A[i] > 0) foo();
bar();
```

Рис. 4: Пример кода и VSFG для него [5]

VSFG позволяет эффективно распараллеливать итерации цикла, так как подграф, соответствующий итерации, может быть встроен в тело родительского графа. Тем самым, операции из очередной итерации цикла станут доступны для выполнения наравне с остальными. На Рис. 5 можно видеть пример такой развертки.

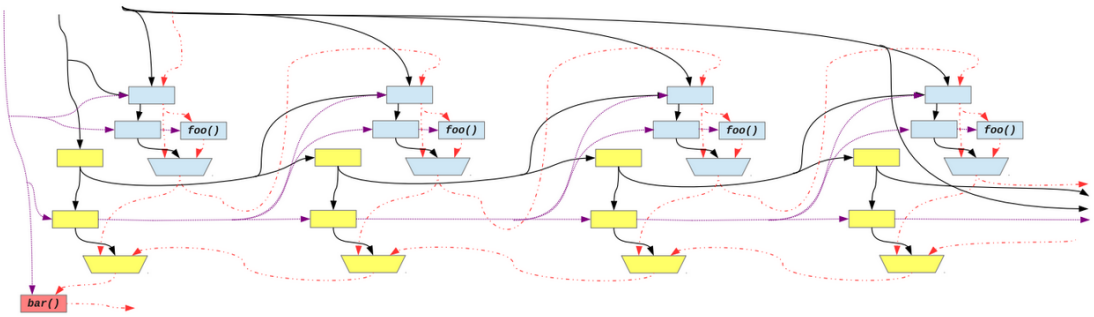


Рис. 5: VSFG с Рис. 4, в котором цикл развернут 4 раза [5]

Еще одним преимуществом данного ВП является возможность одновременного

исполнения нескольких регионов кода. В случае использования CFG невозможно, к примеру, перейти к исполнению функции, следующей за циклом, пока цикл не завершится, даже если функция получила все данные, необходимые для её запуска. Такая ситуация возникает из-за использования единственного потока управления. В VSFG такая проблема отсутствует, так как потока управления в явном виде нет, и все операции запускаются сразу после получения данных.

VSFG поддерживают возможность *спекулятивного исполнения*. То есть, операции могут быть запущены до того, как вычислится значение предиката, от которого они зависят. После вычисления предиката ненужные вычисления завершаются.

3. Реализация

3.1. Структура данных

Для создания основы структуры, описывающей VSFG, была использована библиотека QuickGraph [3]. Разработка велась на платформе .NET, язык программирования - F#. VSFG реализован при помощи класса `AdjacencyGraph<Node, Edge<Node>>`. Тип Node представляет собой discriminated union возможных типов вершин графа:

```
type Node =
    | IntVar of IntV
    | Int of int ref
    | Bool of bool ref
    | Pred of Predicate
    | Inc of Increment
    | Div of Division
    | Gate of Multiplexer
    | NextIter of NestedGraph
```

В данной версии доступно использование переменных типа int (IntV), операций инкремента (Inc) и деления (Div), вершин-предикатов (Pred) и мультиплексоров (Gate). Также возможно реализовать цикл, используя тип NextIter, представляющий собой вложенный граф.

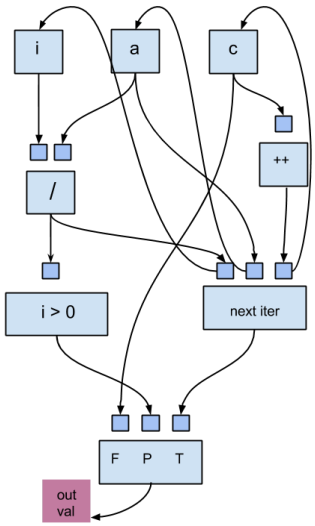
Тип `Edge<'T>` - стандартный класс библиотеки QuickGraph, используется для связи узлов в графе. Вершины-порты (типы Int и Bool) не связываются дугами с вершинами-операциями, а передаются в конструктор класса операции. Пример такого класса приведен ниже.

```
Division (fst: Node, snd: Node) =
    member this.Ports
    member this.Out
```

Каждый класс вершин содержит методы Ports (возвращает список портов) и Out (производит вычисления внутри вершины и возвращает результат).

3.2. Интерпретатор

Для апробации структуры был использован пример, приведенный на Рис. 6. Для обработки графа интерпретатор использует алгоритм обхода в ширину [4]. В качестве аргументов основной метод интерпретатора получает граф и набор стартовых вершин. Во время обхода интерпретатор выполняет операцию в вершине, если для




```
int func () =
    int a = 10;
    int c = 0;
    for (int i = 616; i > 0; ){
        i /= a;
        c++;
    }
    return c;
```

Рис. 6: Структура данных для представленного кода

неё доступны все входные данные, иначе он помещает ее в конец очереди фронта обхода. Если вершина является вложенным подграфом, то её выполнение запускается в отдельном потоке. Для реализации многопоточности был использован один из стандартных механизмов .NET 4.5 [2]. Вложенные структуры выпоняются спекулятивно, после вычисления значения предиката поток, вычисления в котором нецелесообразны, может быть завершен.

Заключение

В рамках данной работы были достигнуты следующие результаты.

- Изучено альтернативное внутреннее представление программ (VSFG).
- Реализована структура данных, описывающая данное представление.
- Реализован интерпретатор, с помощью которого можно апробировать полученную структуру.

Исходный код опубликован в репозитории https://github.com/lares77/Brahma.FSharp Учетная запись автора - lares77.

Список литературы

[1] Janssen Johan. Compiler Strategies for Transport Triggered Architectures. — 2001. — URL: http://ce-publications.et.tudelft.nl/publications/1171_compiler_strategies_for_transport_triggered_architectures.pdf (online; accessed: 30.05.2015).

[2] Microsoft. System.Threading.Tasks // MSDN. — 2015. — URL: https://msdn.microsoft.com/ru-ru/library/system.threading.tasks(v=vs.110).aspx (online; accessed: 30.05.2015).

[3] QuickGraph, Graph Data Structures And Algorithms for .NET // CodePlexProject. Hosting for Open Source Software. — 2015. — URL: http://quickgraph.codeplex.com (online; accessed: 30.05.2015).

[4] Wikipedia. Breadth-first search // Wikipedia. The Free Encyclopedia. — 2015. — URL: https://en.wikipedia.org/wiki/Breadth-first_search (online; accessed: 30.05.2015).

[5] Zaidi Ali Mustafa, Greaves David. A New Dataflow Compiler IR for Accelerating Control-Intensive Code in Spatial Hardware. — 2014. — URL: http://www.cl.cam.ac.uk/~samz2/RAW_submission.pdf (online; accessed: 30.05.2015).