Серия «Математика»
2016. Т. 15. С. 17—25

Онлайн-доступ к журналу:
http://isu.ru/izvestia

ИЗВЕСТИЯ
Иркутского
государственного
университета

УДК 519.682.2
MSC 68N18, 68N20

Реализация многопоточности в рефал-5е машине

В. А. Гошев
Санкт-Петербургский государственный университет

Аннотация. Статья посвящена встроенной поддержке многопоточного выполнения программ в трансляторе языка рефал-5е. Приводится сравнение реализации многопоточного выполнения программ в различных языках программирования, оцениваются их преимущества и недостатки. В результате произведенного сравнения реализаций выбрана такая реализация, которая позволяет автоматически управлять потоками выполнения и их синхронизацией. При использовании такой реализации многопоточного выполнения программ программист может сосредоточиться на функционале самой программы и при этом получать все плюсы многопоточного выполнения. В статье на различных примерах показано, что использование нескольких потоков выполнения в рефал-5е программе действительно позволяет значительно увеличить скорость выполнения многих приложений.

Ключевые слова: рефал-5, рефал-5е, трансляция, многопоточность.

1. Введение

Язык рефал-5е [1; 2] — современный эффективный диалект языка программирования рефал, разработанный и реализованный [3] автором. Одним из важных преимуществ [4] языка программирования рефал-5е является наличие поддержки выполнения рефал-5е машиной программ в несколько потоков, что позволяет выполнять значительное количество программ существенно быстрее на современном оборудовании. Язык рефал-5е основан на более раннем диалекте языка рефал — языке рефал-5. Язык рефал-5 и его синтаксис описаны в работе В. Ф.Турчина [10].

Язык программирования рефал — функциональный язык программирования, в котором чистые функции являются наиболее естественными функциями языка. Чистыми называются детерминированные функции, не создающие побочных эффектов [8]. Одной из особенностей чистых функций является то, что несколько чистых функций можно

выполнять параллельно без каких-либо опасений в том, что выполнение одной из функций так или иначе может повлиять на результат других.

При разработке и реализации транслятора для языка программирования рефал-5е перед автором стояла цель сделать транслятор максимально эффективным. Для достижения этой цели автором была поставлена задача: реализовать многопоточное выполнение рефал-5е программ для увеличения их производительности. В данной работе описывается реализация параллельного выполнения функций в рефал-5е программах без необходимости каких-либо дополнительных действий со стороны разработчика программы.

В процессе поиска наиболее эффективного способа реализации параллельного выполнения рефал-5е программ, автором было проведено исследование и сравнение реализаций многопоточного выполнения приложений в различных языках программирования. В результате исследования была выбрана реализация, которая позволила добиться наилучшего балланса между производительностью и удобством использования. Добавление в рефал-5е машину возможности многопоточного выполнения рефал-5е программ позволило значительно увеличить производительность приложений.

2. Краткое описание работы многопоточного выполнения рефал-5е программ

Поскольку многие функции в языке программирования рефал-5е являются чистыми, имеется возможность реализовать исполнение функций в отдельных потоках выполнения [7] без необходимости синхронизации доступа из функций к глобальным объектам (в связи с отсутствием такого доступа). Данный факт позволяет значительно упростить автоматизацию многопоточного выполнения различных функций в рефал-5е программах и добиться значительного увеличения производительности программ без необходимости дополнительных действий со стороны разработчика приложения.

Рефал-5е машина реализована при помощи языка программирования C [5]. Язык C не предоставляет встроенных в язык средств разработки многопоточных приложений, однако для этого существует библиотека POSIX Threads [7], которая и была использована для реализации многопоточного выполнения рефал-5е программ.

Для реализации многопоточного выполнения программ рефал-5е машина, являющаяся основой реализации языка рефал-5е, проверяет при запуске количество ядер центрального процессора, доступных для рефал-5е машины и создает соответствующее количество потоков выполнения. Если при выполнении очередной функции (после успешного сопоставления аргумента функции с одним из образцов и выполнения

соответствующих условий) рефал-5е машина встретила вызов функции и имеется хотя бы один свободный поток выполнения, то выполнение вызова функции передается этому свободному потоку выполнения с указанием того, куда необходимо вернуть данные. Этот поток производит выполнение необходимой функции с указанными аргументами, а затем записывает результат в место, которое было передано потоку выполнения вместе с указанием необходимости выполнения функции.

3. Сравнение реализации многопоточности в языке программирования рефал-5е с другими существующими реализациями

Параллельные вычисления — одно из наиболее популярных направлений в современной информатике, поэтому многие современные языки программирования предоставляют те или иные инструменты для разработки многопоточных приложений.

В большинстве языков программирования, например, в C, C++, Perl, PHP и других программисту предоставляются средства для самостоятельного создания и управления дополнительными потоками или процессами. В частности, для этого используется библиотека POSIX Threads [7], о которой упоминалось выше.

Из плюсов такого решения можно выделить то, что программист имеет полный контроль над многопоточностью приложения и, как следствие, может самостоятельно решать, какие операции нужно выполнять последовательно, а какие — параллельно. Также имеется возможность указывать приоритет выполнения для различных потоков, тем самым позволяя тонко настраивать поведение потоков и добиваться максимальной производительности.

Однако полный контроль над потоками имеет и обратную сторону. Программисту приходится самостоятельно решать такие задачи, как синхронизация потоков, доступ к общим ресурсам, а также другие связанные с многопоточностью проблемы. Как следствие, многопоточные приложения в этих языках нередко имеют ошибки, которые сложно обнаружить в силу недетерменированности выполнения потоков в многопоточном приложении.

Такие языки как, например, Haskell в свою очередь предоставляют более дружественные к разработчику инструменты для разработки многопоточных приложений. Для того, чтобы указать, что какие-то действия необходимо выполнить параллельно, достаточно лишь перед этими операциями при помощи специальной конструкции указать, что они должны выполняться параллельно с последующим кодом, а уже компилятор самостоятельно проследит за синхронизацией потоков и распределит доступ к ресурсам [9]. При запуске программы необхо-

димо указать, сколько потоков должна создать программа для своего выполнения.

Значительным плюсом такого решения является то, что программисту не нужно следить за тем, чтобы потоки были правильно синхронизированы и не пытались одновременно получать доступ к одному и тому же ресурсу. Всю эту работу берет на себя транслятор языка.

Для разработки многопоточных приложений на языке рефал-5е не нужно выполнять никаких дополнительных действий. Рефал-5е машина автоматически выполнит функцию параллельно, если такое возможно. Разработчику лишь необходимо написать программу так, чтобы появилась возможность выполнять ее в многопоточном режиме. В результате такого решения программист может сосредоточиться на реализации нужного алгоритма, а не на управлении многопоточностью программы.

4. Описание алгоритма многопоточного выполнения рефал-5е программ

Алгоритм 1 описывает многопоточное выполнение программ рефал-5е машиной.

Алгоритм 1.

– При запуске рефал-5е машины, она проверяет количество ядер процессора на машине и создает потоки выполнения в количестве, равном количеству ядер процессора. Для создания и управления потоками, как было указано выше, используется библиотека POSIX Threads [7].
– При выполнении функции, после того как рефал-5е машина сопоставила переданные в функцию данные с каким-либо образцом, и все условия для него были выполнены, рефал-5е машина начинает выполнение указанных действий и формирование результата. При этом, когда рефал-5е машина встречает операцию вызова функции, она выполняет следующий алгоритм:

1) Проверяет, присутствуют ли в аргументах вызываемой функции другие функции, если присутствуют, то сначала обрабатываются они (начиная с первого вхождения другой функции).
2) Рефал-5е машина проверяет, является ли вызываемая функция чистой или нет. Для проверки чистоты функции производится рекурсивная проверка на чистоту всех функций, вызываемых данной функцией. Сами функции, написанные на языке рефал-5е, могут быть нечистыми только в том случае, когда они (возможно, опосредованно) вызывают нечистые библиотечные функ-

ции. По этой причине проверка сводится к проверке, вызывается ли в ходе выполнения данной функции какая-нибудь нечистая библиотечная функция. Для оптимизации данная проверка происходит на этапе трансляции программы на языке программирования рефал-5е в язык рефал-5е машины. Как следствие во время выполнения данная проверка сводится к проверке записанной в коде информации о чистоте функции.

3) Если вызываемая функция является чистой, то информация о необходимости вызова функции добавляется в очередь с указанием того, куда необходимо вернуть результат. В качестве результата ставится временная метка, сообщающая о том, что результат еще не вычислен.

4) Если функция не является чистой, то перед тем, как ее вызвать, поток выполнения проверяет очередь выполнения чистых функций.

   а) Если все чистые функции в очереди выполнены и при этом ни одна из них не вернула «Неуспех», то поток выполнения производит вызов и выполнение функции самостоятельно.

   б) Если одна из ранее вызванных чистых функций вернула «Неуспех», то поток выполнения производит переход к обработке следующего выражения после разделителя «;» (при этом при выполнении любого правила рефал-5е машина не может вернуться левее знака «=» в этом правиле) [1]. Если же следующее выражение отсутствует, то в качестве результата текущей функции устанавливается «Неуспех». Все функции, встреченные при выполнении текущей функции, но не успевшие выполниться, удаляются из очереди выполнения.

   в) Если одна или более чистых функций в очереди еще не выполнены, то выполнение текущей функции приостанавливается и текущий поток пытается выполнить функцию из списка чистых функций, ожидающих выполнения. Это делается для того, чтобы поток выполнения не просто ожидал завершения выполнения необходимых функций, но и совершал при этом полезную работу. После выполнения функции (или просто ожидания, в случае отсутствия функций, которые ожидают своего выполнения) поток выполнения опять проверяет, все ли чистые функции, встреченные при выполнении текущей функции, выполнены. Таким образом, происходит переход к пункту 4 данного алгоритма.

– Все потоки выполнения, не занятые исполнением кода в данный момент (то есть свободные потоки выполнения), периодически производят проверку, очереди выполнения функций на присутствие

функций, которые необходимо выполнить и при этом все их аргументы вычислены. Если такая функция найдена, то дальше поток выполнения запоминает все необходимые данные о вызываемой функции (то есть имя функции, ее аргументы и место, куда необходимо записать результат) и удаляет вызов функции из очереди выполнения, чтобы другие потоки выполнения не пытались ее выполнить. После этого поток вычисляет функцию согласно описываемому алгоритму.

5. Примеры многопоточного выполнения рефал-5е программ

В качестве одного из примеров, который хорошо демонстрирует преимущество многопоточного выполнения кода, приведем функцию, вычисляющую разность квадратов двух чисел. Пример 1 показывает, как данная функция может выглядеть на языке программирования рефал-5е.

Пример 1.

```
SubSqr {
  s.1 s.2 = ⟨Math::Sub ⟨Math::Pow s.1 2⟩ ⟨Math::Pow s.2 2⟩ ⟩;
}
```

Данная функция принимает на вход два символа, которые должны быть числами, и вычисляет разность между их квадратами. При однопоточном выполнении рефал-5е машина сначала выполнила бы функцию ⟨*Math::Pow s.1 2*⟩, затем ⟨*Math::Pow s.2 2*⟩ и в конце ⟨*Math::Sub*⟩, в качестве аргументов которой были бы результаты выполнения функций. При многопоточном выполнении поток выполнения, согласно алгоритму описанному в предыдущем разделе, добавляет функции ⟨*Math::Pow s.1 2*⟩, ⟨*Math::Pow s.2 2*⟩, а затем и ⟨*Math::Sub*⟩ в список выполнения, где первые две функции одновременно начинают выполняться свободными потоками выполнения (при условии, что свободные потоки выполнения имеются в наличии), и после их завершения начинает выполняться функция ⟨*Math::Sub*⟩. В силу того, что возведение в степень является длительной операцией, то выполнение двух возведений в степень параллельно, а не последовательно, значительно ускоряет выполнение указанной функции. Данная функция при выполнении на компьютере с процессором i.MX6 Dual, имеющим два ядра, в многопоточном режиме была выполнена в 1.8 раза быстрее, чем в однопоточном режиме.

В качестве другого примера приведем сортировку списка методом быстрой сортировки [6]. Одним из преимуществ данного алгоритма яв-

ляется то, что он хорошо распараллеливается на несколько потоков выполнения. Так, программа, реализующая алгоритм быстрой сортировки для произвольного списка целых чисел, при выполнении на компьютере с процессором i.MX6 Quad, имеющим четыре ядра, в многопоточном режиме была выполнена в 3,4 раза быстрее, чем в однопоточном режиме.

Производительность многопоточного выполнения программ также проверялась на следующих программах: параллельный алгоритм Флойда поиска кратчайших путей в графе, умножение матрицы на вектор, искусственная нейронная сеть. Результаты измерений многопоточных выполнений программ по сравнению с однопоточным представлены в следующей таблице.

Таблица 1

Программа	Производительность многопоточного выполнения по сравнению с однопоточным
Разность квадратов	1.8
Быстрая сортировка	3.4
Параллельный алгоритм Флойда	2.5
Умножение матрицы на вектор	3.1
Нейронная сеть	2.7
В среднем:	2.7

6. Заключение

Сегодня разработчики процессоров делают все больший упор на многоядерные и многопроцессорные компьютерные системы. По этой причине для создания эффективных приложений необходимо использовать все доступные приложению ядра процессора. Поэтому многопоточное выполнение программы является одной из необходимых технологий, используемых при разработке современных эффективных приложений.

В результате проведенных исследований поставленная задача была достигнута. Реализация многопоточного выполнения программ в рефал-5е машине, сделанная автором, позволила, в среднем, увеличить производительность программ в 2.7 раза. В результате того, что многопоточное выполнение программ в языке программирования рефал-5е работает без вмешательства разработчика, программист может сосредоточиться на функционале при разработке программы, а не на оптимизации программного кода для многопоточного выполнения.

Список литературы

1. Гошев В. А. Проект языка программирования рефал-5е с удобными расширениями препроцессором / В. А. Гошев, Н. К. Косовский // Компьютер. инструменты в образовании. – 2014. – №1. – С. 3–13.
2. Гошев В. А. Реализация транслятора для языка программирования рефал-5е со встроенным интерпретатором и возможностью подключения библиотек кода / В. А. Гошев // Компьютер. инструменты в образовании. – 2014. – № 5. – С. 16–25.
3. Гошев, В. А. Рефал-5Е: Разработка языка и реализация транслятора / В. А. Гошев, Н. К. Косовский // Технологии Microsoft в теории и практике программирования. Новые подходы к разработке программного обеспечения по технологиям Microsoft и EMC : материалы учеб.-практ. конф. школьников, студентов, аспирантов и молодых ученых Сев.-Зап. федер. округа. – СПб. : Изд-во политехн. ун-та, 2014. – С. 91–92.
4. Гошев, В.А. Особенности языка программирования рефал-5е / В.А. Гошев // Материалы Всерос. науч. конф. по проблемам информатики. Санкт-Петербург, 23–25 апр. 2014 г. – СПб. : Изд-во ВВМ, 2014. – С. 66–68.
5. Керниган Б. Язык программирования Си / Б. Керниган, Д. Ритчи. – 2-е изд. – М. : Вильямс, 2007.
6. Левитин А. В. Алгоритмы: введение в разработку и анализ / Ананий В. Левитин. – М. : Вильямс, 2006.
7. David R. B. Programming with POSIX Threads / David R. Butenhof. – Addison-Wesley, 1997.
8. Love R. Linux System Programming. Talking Directly to the Kernel and C Library / R. Love. – O'Reilly, 2007.
9. Marlow S. Parallel and Concurrent Programming in Haskell: Techniques for Multicore and Multithreaded Programming / S. Marlow. – O'Reilly, 2013.
10. Turchin V.F. Refal-5. Programming Guide and Reference Manual / V. F. Turchin. – New England Publishing Co., Holyoke, 1989.

Гошев Владимир Андреевич, аспирант, Санкт-Петербургский государственный университет, 199034, Санкт-Петербург, Университетская набережная, 7-9 тел.: +7 (981) 8434068 (e-mail: sunx@sunx.cc)

V. A. Goshev
Implementation of Multithreading in Refal-5e Machine

Abstract. This paper contains a description of one of the features of the Refal-5e programming language, a built-in support for the multi-threading programs execution. This feature allows much faster performance for many programs on modern hardware in comparison with single-threaded techniques. In the paper we have also compared implementations of milti-threading in different programming languages, their advantages and disadvantages. Our implementation of multi-threading in Refal-5e machine can automatically control the flow and timing. As a result, a software developer can focus on the functionality of the program and still get all the advantages of multi-threaded execution of the programs.

Keywords: refal-5, refal-5e, translation, multithreading.

References

1. Goshev V.A., Kosovskiy N.K. Project of programming language Refal-5e with convenient extensions by means of preprocessor (in Russian). *Computer instruments in education*, 2014, vol. 1, pp 3-13.
2. Goshev V.A. Implementation of REFAL-5E programming language translator with embedded interpreter and support of external code libraries (in Russian). *Computer instruments in education*, 2014, vol. 5, pp 16-25.
3. Goshev V.A, Kosovskiy N.K. Refal-5e: Development of language and implementation of translator (in Russian). *Technology of Microsoft in theory and practick of software development* St. Petersburg, 2014, pp 91-92.
4. Goshev V.A. Features of programming language refal-5e (in Russian). *Proc. Russian Symposium on Problems of informatics*, St. Petersburg, April 2014, pp 66-68.
5. Brian W. Kernighan, Dennis M. Ritchie. The C Programming Language 2nd Edition. AT&T Bell Laboratories, 2007.
6. Anany Levitin. Introduction to The Design and Analysis of Algorithms . Villanova University, 2006.
7. David R. Butenhof. Programming with POSIX Threads. Addison-Wesley, 1997.
8. Love R. Linux System Programming. Talking Directly to the Kernel and C Library. OŔeilly, 2007.
9. Marlow S. Parallel and Concurrent Programming in Haskell: Techniques for Multicore and Multithreaded Programming. O'Reilly, 2013.
10. Turchin V.F. Refal-5. Programming Guide and Reference Manual. New England Publishing Co., Holyoke, 1989.

Goshev Vladimir Andreevich, Postgraduate, Saint Petersburg State University, 7-9, University nab., St. Petersburg, 199034
(e-mail: `sunx@sunx.cc`)