мость монтажных работ с учетом существующих несущих конструкций на и вне охраняемой территории, величина эксплуатационных расходов и т.п.

Сама оптимизационная задача является структурно-параметрической. Ввиду сложности для ее решения рекомендуется численный метод.

Список литературы

1. Цыпкин А.Г., Цыпкин Г.Г. Математические формулы. М.: Наука, 1985. 127 с.

2. Johnson J. Analysis of image forming systems // Image Intensifier Symposium. AD 220160 (Warfare Electrical Engineering Department, U.S. Army Research and Development Laboratories). Ft. Belvoir. Va., 1958. P. 244 –273.

3. Аттетков А.В., Галкин С.В., Зарубин В.С. Методы оптимизации М.: Изд-во МГТУ им. Н.Э. Баумана, 2003. 339 с.

M. Tyuchanov

Evaluation of a watched territory squire in a distributed video system

The task of evaluation of a watched territory squire for the case of arbitrary positioning of TV-s and intersection of observation sectors is solved. The task of optimal distribution of chambers on a territory is formulated.

Keywords: TV-chamber, base co-ordinate system, linked co-ordinate system, observation sector squire, intersection structure-parametric optimization, optimization criterion.

Получено 07.04.10

УДК 004.4:414

А.П. Колосов, асп., 9202741745,
alexey.kolosoff@gmail.com (Россия, Тула, ТулГУ)

МЕТОДЫ ОТЛАДКИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

Рассматриваются проблемы повышения качества параллельных программ, а также особенности, затрудняющие их отладку. Выполнен обзор четырех методов такой отладки: традиционная отладка, отладка на основе событий, динамический анализ и статический анализ. Приведена классификация ошибок специфических для параллельного программирования.

Ключевые слова: параллельное программирование, отладка, статический анализ, сети Петри.

Введение

Параллельное программирование появилось достаточно давно. Первый многопроцессорный компьютер был создан еще в 60-х годах прошлого века. Однако до недавних пор в персональных компьютерах при-

рост производительности процессоров обеспечивался в основном благодаря росту тактовой частоты, и многопроцессорные персональные компьютеры были редкостью. Сейчас рост тактовой частоты в персональных компьютерах замедляется, и прирост производительности обеспечивается за счет использования нескольких ядер. В связи с этим задача написания параллельных программ приобретает все большую и большую актуальность. Если ранее для увеличения производительности программы пользователь мог просто установить процессор с большей тактовой частотой, то теперь такой подход невозможен, и существенное увеличение производительности в любом случае потребует усилий от программиста.

В связи с тем, что параллельное программирование для персональных компьютеров начинает набирать популярность только сейчас, процесс распараллеливания существующего приложения или написания нового параллельного кода может вызвать проблемы даже для опытных программистов, так как данная область является для них новой.

По оценкам компании «Evans Data», проводящей опросы среди разработчиков ПО, на сегодняшний день общее количество программистов в мире составляет 14,6 миллионов человек, причем около 70 % из них занимаются разработкой многопоточных приложений сейчас или планируют начать ее в течение года [1] . По данным того же опроса, 33,4 % разработчиков считают, что главной проблемой таких разработок является сложность параллельного программирования, а 13,7 % – нехватка программных средств для создания, тестирования и отладки параллельных приложений. Следовательно, в решении задачи автоматического поиска ошибок в исходном коде непосредственно заинтересованы примерно 2 млн. программистов. При этом параллельное программирование, приход которого многие специалисты сравнивают с приходом ООП [2] , для многих из них является новой областью.

Языки и библиотеки

Так же, как и в классическом, «последовательном» программировании, для параллельного программирования применяется множество различных языков и библиотек, существенно отличающихся по своим принципам и возможностям. График, показывающий наиболее популярные языки параллельного программирования по данным еще одного опроса [3], приведен на рис. 1.

На рис. 1 «Funct.» обозначает языки функционального программирования, «Logical» соответствует языкам логического программирования, а «Other» – другим языкам. Как видно наиболее популярными языками являются C и C++, что, в принципе, ожидаемо, так как к программам именно на этих языках применяются, как правило, требования высокой производительности. В то же время в этих языках отсутствуют встроенные средства распараллеливания. Специальная поддержка параллельного программирования должна появиться только в языке C++0x, который, в частности, под-

держивается компилятором Visual Studio 2010. Сегодня же для создания параллельных программ на этих языках используются специальные библиотеки, реализующие необходимую функциональность. График, иллюстрирующий популярность различных библиотек (по данным упомянутого ранее опроса [3]), представлен на рис. 2.

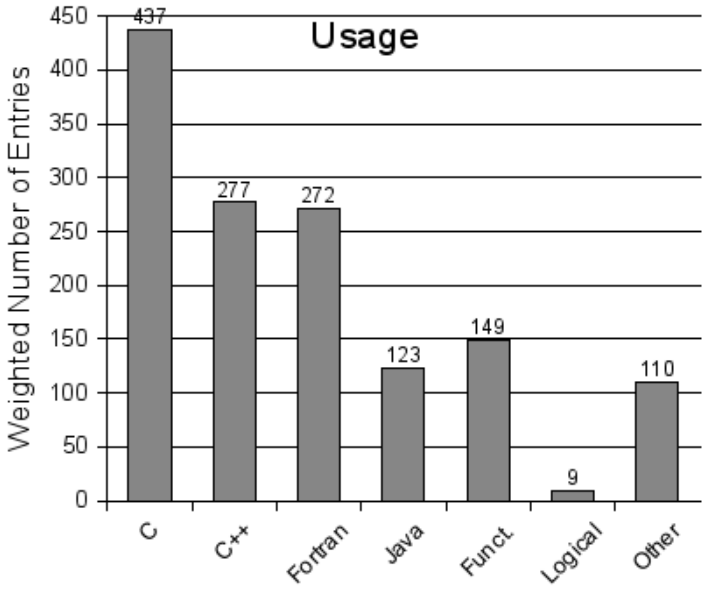

Рис. 1. Наиболее популярные языки параллельного программирования

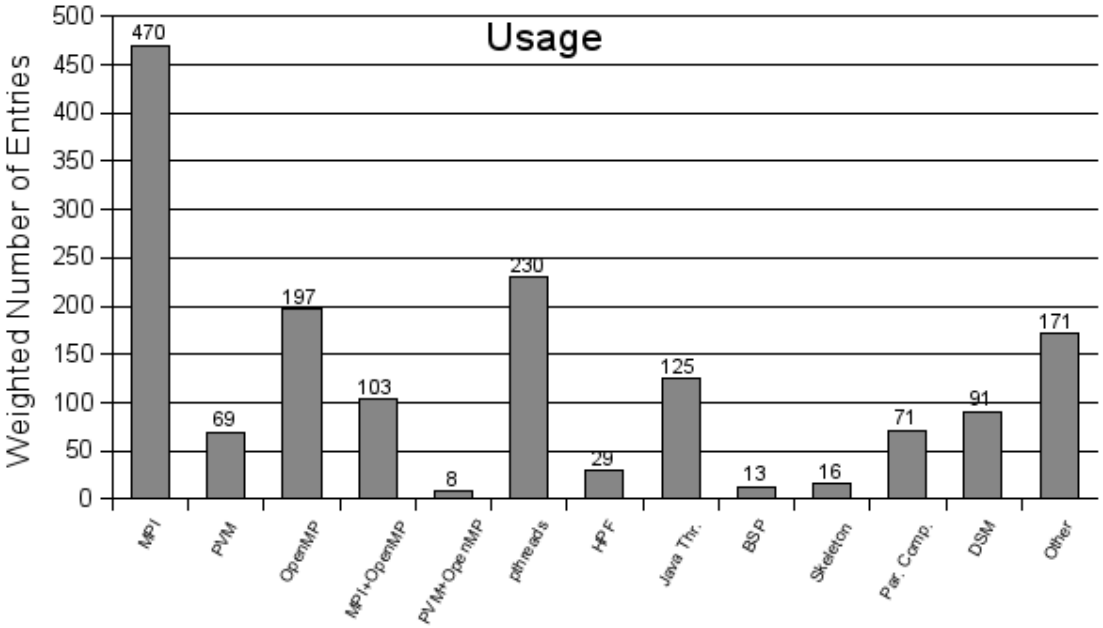

Рис. 2. Наиболее популярные библиотеки параллельного программирования

На рис. 2 «Par.Comp.» соответствует распараллеливающим компиляторам (которые, по данным опроса, не особенно популярны). Наиболее популярными библиотеками являются MPI, OpenMP и POSIX threads (или Pthreads). Отметим, что в языках Fortran и Java, имеются встроенные средства распараллеливания, поэтому, несмотря на то, что упомянутые наиболее популярные библиотеки в том или ином виде доступны и для этих двух языков, в них чаще применяются встроенные средства распараллеливания.

Еще раз напомним, что все эти языки и библиотеки различаются своими возможностями и даже самими принципами организации параллельности. Однако есть между ними и кое-что общее: ошибки, которые могут привести к неожиданному поведению параллельных программ.

Общие ошибки

Наиболее распространенные ошибки, встречающиеся в параллельных программах, делятся на два вида: тупики (зависания) и состояния гонок. Эти ошибки могут возникнуть в программе вне зависимости от языка или библиотеки, использующейся для реализации параллельности, поскольку обусловлены они самими основами параллельного программирования (которые остаются неизменными уже не первое десятилетие).

Теперь рассмотрим каждый из двух видов ошибок подробнее.

Тупики (зависания) возникают тогда, когда для продолжения работы многопоточной программы каждому из потоков необходимо дождаться освобождения какого-либо объекта, но этот объект не может быть освобожден. Как правило это происходит в том случае, когда поток А ждет освобождения одного объекта (например, семафора) потоком Б, а поток Б, в свою очередь, ждет освобождения другого объекта потоком А. Зависание также может произойти, если один из потоков заблокирует и не освободит объект, необходимый для дальнейшей работы остальных потоков.

Состояние гонок возникает при доступе к одной и той же области общей памяти двух или более потоков одновременно тогда, когда выполнены два условия:

1) хотя бы одна из этих операций доступа является операцией записи;

2) потоки не используют механизм, явно предотвращающий одновременные операции доступа (например, критические секции) [4] .

В то время как последствия первой ошибки достаточно очевидны, программа перестанет реагировать на действия пользователя, последствия второй ошибки могут быть весьма разнообразны. Чтение или запись неправильного значения переменной может вызвать неожиданное поведение тестируемой программы, которое к тому же очень трудно будет воспроизвести, ведь при повторном выполнении программы та же переменная может принять другое значение (в том числе и корректное). Такое поведение обуславливается устройством современных многоядерных процессоров: поскольку каждое ядро, фактически, работает в своем собственном времени, которое не синхронизируется с временем других ядер, операции, вы-

полняемые одновременно, могут совпасть, а могут и не совпасть в определенный момент реального, физического времени. Вероятность такого совпадения зависит от числа ядер и количества операций, выполняемых ими одновременно с общей областью памяти. Оба значения, естественно, увеличивают эту вероятность.

Например, цикл из 100000 итераций в приложении на языке C, распараллеленный с помощью OpenMP на два ядра, приведет к состоянию гонок почти в 100 % случаев, если его тело будет содержать одну лишь простейшую операцию увеличения общей переменной на единицу. Переменная в результате будет принимать почти случайные значения в интервале примерно от 45.000 до 99.000.

Диагностика ошибок

Существует целый ряд причин, делающий отладку параллельных программ намного сложнее, чем отладку последовательных. Рассмотрим некоторые их них.

Эффект наблюдателя – любая попытка слежения за поведением параллельной системы может изменить поведение этой системы. Типичным примером этой проблемы является попытка обнаружить состояние гонок с помощью выполнения кода по шагам, в этом случае проблема одновременного доступа к общей памяти, скорее всего, не возникнет, так как действия параллельных потоков будут разделены во времени.

Нестабильное воспроизведение – одна и та же программа может давать различные результаты с одними и теми же данными.

Отсутствие единых, синхронизированных часов (являющееся причиной двух предыдущих проблем) может затруднить анализ результатов наблюдений.

Для отладки многопоточных приложений применяются следующие основные методы.

Традиционная отладка (выполнение кода по шагам, использование точек останова). Преимущество данного подхода заключается в том, что он позволяет детально изучить состояние системы в каждый момент времени и проконтролировать изменение этого состояния. Недостаток заключается в том, что этот подход, как уже упоминалось ранее, подвержен воздействию эффекта наблюдателя.

Отладка на основе событий (представление выполнения программы как параллельных цепочек событий). Такой подход позволяет получить стабильное воспроизведение той или иной проблемы в виде записи последовательности приводящих к ней изменений состояний системы. Однако, если запись информации о событиях не может производиться непрерывно, не оказывая существенного влияния на ход выполнения программы, на результаты может оказать влияние эффект наблюдателя.

Динамический анализ (инструментирование исполняемого кода программы) позволяет обнаружить ошибки во время работы программы

(например, при выполнении ряда автоматизированных тестов). Этот подход (как и два предыдущих) требует запуска тестируемой программы, позволяет проанализировать поведение, зависящее от пользователя, но проверяет, с другой стороны, только код, выполняющийся во время теста, а также существенно замедляет выполнение тестируемой программы из-за необходимости сбора данных о ее состоянии (профилей). Сам процесс анализа также называется профилированием и, по сравнению с обычным временем работы программы продолжаться он может в сотни и даже тысячи раз дольше [5] .

Статический анализ (разбор и анализ исходного кода программы). Этот подход позволяет проанализировать исходный код программы и обнаружить некоторые ошибки, не запуская при этом саму программу. Статический анализ не подвержен воздействию эффекта наблюдателя, но позволяет проверять лишь то поведение, которое не зависит от пользователя. Еще одним недостатком статического анализа является то, что он обнаруживает лишь потенциально подозрительные, а не гарантированно ошибочные строки исходного кода. Это означает, что решение об исправлении исходного кода должен принимать программист, а процесс такого тестирования является автоматизированным, но не автоматическим.

Статический анализ как средство диагностики

Как уже упоминалось выше, данный подход имеет ряд преимуществ и недостатков, которые, впрочем, можно компенсировать параллельным применением динамического анализатора или какого-либо иного отладчика. Совместное применение отладчиков, использующих различные методы, вообще позволяет добиться наибольшей эффективности при поиске ошибок.

Статический анализ интересен своей универсальностью – он позволяет рассматривать тестируемую программу как некую абстракцию, не связанную с конкретным языком программирования и/или библиотекой. Процесс статического анализа фактически состоит из двух этапов: построения модели исследуемой программы и анализа этой модели. Таким образом, один и тот же анализ можно провести для моделей различных программ, более того, подаваемая на вход модель не обязательно должна быть моделью программы, она может представлять любую систему, лишь бы допустимые вид и язык, использующиеся для описания этой модели, были понятны лексическому и синтаксическому анализаторам (то есть модулям, отвечающим за построение модели из поданных на вход данных).

Инструменты, используемые для анализа параллельных программ, можно условно разделить на три вида по их функциональности:

1) средства создания моделей, например, C2Petri [6] или Evinrude [7] , создающие сети Петри из исходного кода на языке C, использующего библиотеку POSIX threads;

2) средства анализа моделей (в англоязычных источниках называемые model checkers);

3) статические анализаторы, функциональность которых включает в себя и создание моделей (как правило, деревьев кода), и их анализ. Например, PVS-Studio – анализатор, помимо прочего, обнаруживающий ошибки в параллельных программах на C/C++, использующих технологию OpenMP.

Наиболее распространенным средством моделирования параллельных программ являются сети Петри, представляющие процесс работы параллельной программы в виде графа. Реже применяются UML и специальный метаязык PROMELA (например, в анализаторе Spin). Преимущество сетей Петри заключается в наличии развитого математического аппарата [8], позволяющего производить анализ, руководствуясь математически доказанными теоремами, а не только эвристическими правилами, специфичными для каждого конкретного языка. Анализ с использованием сетей Петри может проводиться на различных этапах разработки программы. В частности, она может быть изначально разработана и отлажена в виде сети Петри, которая затем автоматически преобразуется в исходный код на Java, C/C++ или каких-либо еще языках высокого уровня (например, с помощью AC/DC (Automatic Code Generation from Design/CPN), LOOPN++ или других инструментов, общее количество которых превышало 50 уже к 2000 году) [9] . Однако далеко не каждый программист строит модели планируемой программы, прежде чем начинать ее писать, поэтому куда более распространенной задачей является анализ уже имеющегося кода. Такая задача может быть выполнена статическим анализатором, сочетающим в себе функциональность построения модели и ее анализа.

Отметим, что общие шаги в процессе работы такого инструмента могут различаться в зависимости от конкретной реализации. Возможные варианты таких шагов представлены на рис. 3 (здесь под моделью понимается сеть Петри).

Рис. 3. Различные варианты анализа параллельной программы

Вариант, показанный слева на рис. 3, является наиболее распространенным среди статических анализаторов, в частности, его использует упомянутый ранее анализатор PVS-Studio, вариант справа используется, в частности, инструментами, описанными в [7, 10]. Он может также достигаться путем комбинированного использования нескольких инструментов: генератора сети Петри из исходного кода и анализатора, проверяющего построенную сеть на предмет состояний гонок и тупиков.

Заключение

Параллельное программирование для персональных компьютеров с каждым днем становится все более и более популярным. В течение этого года с ним так или иначе придется столкнуться около 70 % программистам. При этом сложность параллельного программирования и нехватку средств для создания и отладки параллельных программ чаще всего называют причинами, мешающими развитию параллельного программирования.

Наиболее распространенными проблемами, встречающимися в параллельных программах, являются тупики (зависания) и состояния гонок (одновременная запись данных в общую область памяти несколькими потоками и (опционально) чтение из этой же области). Автоматическое или даже автоматизированное нахождение таких ошибок связано с немалыми трудностями в связи с тем, что они могут не воспроизводиться стабильно, а любая попытка наблюдения за системой может привести к изменению ее поведения. Для обнаружения таких ошибок разработан ряд методов, одним из которых является статический анализ. Данный метод интересен тем, что позволяет в автоматизированном режиме за разумное время получить информацию о корректности тестируемой программы, не запуская эту программу. При этом параллельные программы можно моделировать с помощью сетей Петри, для которых разработан универсальный, не зависящий от типа моделируемой системы, математический аппарат, позволяющий анализировать корректность данной системы.

В связи с этим перспективными представляются создание инструмента, реализующего соответствующие правила на практике и работающего с несколькими (наиболее популярными) языками программирования, а также разработка некоторого универсального алгоритма, позволяющего легко расширять множество поддерживаемых языков, описывая соответствия между конкретными синтаксическими конструкциями и соответствующими сетями Петри.

Список литературы

1. Evans Data Market Alert: Intel’s Parallel Studio Brings Parallel Programming into the Mainstream, Issue 1, June 2009. URL: http://www.evansdata.com/research/market_alerts/EDC_Market_Alert_2009_June_Issue.pdf (дата обращения: 13.06.2009).

2. Sutter H. A Free Lunch Is Over. URL: http://www.gotw.ca/publications /concurrency-ddj.htm (дата обращения: 30.03.2008).

3. Suess M, Leopold C. Observations on the Publicity and Usage of Parallel Programming Systems and Languages: A Survey Approach. URL: http://kobra.bibliothek.uni-kassel.de/bitstream/urn:nbn:de:hebis:34-2007050818071/6/survey.pdf (дата обращения: 08.05.2007).

4. A Dynamic Data Race Detector for Multithreaded Programs / S. Savage [et al.]. ACM Transactions on Computer Systems. 1997. Vol. 15. № 4/ P. 391-411.

5. Intel Thread Checker in Cimatron ltd. URL: http://software.intel.com/ en-us/articles/intel-thread-checker-in-cimatron/ (дата обращения: 14.10.2009).

6. Kavi M., Moshtahi A. Modeling Multithreaded Applications Using Petri Nets. International Journal of Parallel Programming, Vol. 30, № 5. P. 353-371.

7. Voron J., Kordon F. Evinrude: A Tool To Automatically Transform Program's Sources into Petri Nets // Petri Nets Newsletter. Vol. 75.

8. Котов В.Е. Сети Петри. М.: Наука, 1984. 159 с.

9. Mortensen K. Automatic Code Generation from Coloured Petri Nets for an Access Control System. URL: www.daimi.au.dk/CPnets/ workshop99/ papers/Mortensen.pdf (дата обращения: 11.01.2010).

10. Haziza F. Model Checking Race-Freeness. URL: http:// portal.acm.org/ft_gateway.cfm?id=1556454&type=pdf (дата обращения: 15.11.2009).

A. Kolosov

Concurrent applications debugging techniques

Parallel programming development and debugging problems are discussed. Four debug techniques are described: traditional debug, event-driven debug, static analysis and dynamic analysis.

Keywords: parallel programming, debug, static analysis, Petri nets.

Получено 07.04.10

УДК 621.3.07,681.58

С.А. Шопин, асп., (4872)-40-27-25, 953-423-83-72,
sshopin@mail.ru (Россия, Тула, ТулГУ)

СИСТЕМА УПРАВЛЕНИЯ АСИНХРОННЫМ ПРИВОДОМ НА ПРОЦЕССОРЕ BLACKFIN

Описаны структура и состав блоков системы управления асинхронным приводом, позволяющей исследовать сложные нелинейные законы управления и наблюдения, требующие большого объема вычислений.

Ключевые слова: асинхронный привод, система управления, микропроцессор.

В настоящее время для реализации систем векторного управления асинхронным приводом существует большое количество стандартных реше-