ИПМ им.М.В.Келдыша РАН

Абрау-2017 • Труды конференции

А.А. Ермичев, В.А. Крюков

Развитие метода сравнительной отладки DVMH-программ.

Рекомендуемая форма библиографической ссылки

Ермичев А.А., Крюков В.А. Развитие метода сравнительной отладки DVMH-программ. // Научный сервис в сети Интернет: труды XIX Всероссийской научной конференции (18-23 сентября 2017 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2017. — С. 150-156. — URL: http://keldysh.ru/abrau/2017/15.pdf doi:10.20948/abrau-2017-15

Размещена также презентация к докладу

Развитие метода сравнительной отладки DVMH-программ

А.А. Ермичев[1,2], В.А. Крюков[1,2]

1 Институт прикладной математики им. М.В. Келдыша РАН
2 Факультет вычислительной математики и кибернетики Московского государственного университета им М.В. Ломоносова

Аннотация. Отладка параллельных программ является трудоемкой и нетривиальной задачей. Для автоматизации этого процесса DVM-система предоставляет механизм сравнительной отладки, который позволяет обнаруживать расхождения промежуточных результатов параллельного и последовательного выполнения DVMH-программы. Сравнительная отладка в DVM-системе реализована посредством трассировки при выполнении программы следующих событий: чтения и модификации переменных, итерации циклов и т.п. Получаемые при параллельном выполнении промежуточные результаты сравниваются с эталонными, в качестве которых обычно рассматриваются результаты последовательного выполнения, ранее сохраненными в виде файлов с трассами. Но при отладке реальных программ размер этих файлов может значительно превысить возможности файловой системы. Для таких случаев требуется другой способ организации сравнительной отладки – одновременный запуск последовательного и параллельного выполнения программы и сравнение их промежуточных результатов «на лету». В данной статье описываются принципы реализации в DVM-системе такого режима отладки.

Ключевые слова: параллельное программирование, автоматизация параллельного программирования, отладка параллельных программ

1. Введение

Программирование для высокопроизводительных систем становится все более значимым с быстрым развитием и распространением многоядерных машин, многопроцессорных комплексов и кластеров, использующих для вычислений различные ускорители (графические платы, сопроцессоры). С этой тенденцией все большее внимание уделяется проблемам, возникающим при создании параллельных программ.

Написание параллельного приложения «с нуля» – довольно сложная задача. Конечно, в большинстве случаев в распоряжении программиста имеется последовательный код, который можно модифицировать вызовами функций модели передачи сообщений MPI или (если планируется использование графического ускорителя) технологии CUDA. Но, тем не менее, объем

проводимой работы обычно оказывается довольно значительным, а выходная программа – громоздкой и зачастую сложной для понимания программисту, привыкшему работать с последовательными алгоритмами. Для облегчения данной задачи были разработаны высокоуровневые параллельные модели и языки. Распараллеливание программы в рамках данных языков осуществляется путем добавления в код спецификаций (в форме директив компилятора или комментариев специального вида), управляющих процессом преобразования исходного алгоритма для параллельного выполнения. Примерами подобных моделей являются OpenMP, OpenACC, HPF [1], DVMH [2,3,4]. Подобный подход позволяет упростить создание параллельных реализаций объемных программных комплексов, состоящих из тысяч строк кода.

Несмотря на упростившееся распараллеливание при использовании высокоуровневых моделей, идеальным инструментом для программиста остается система, автоматически преобразующая последовательную программу в параллельную, способную эффективно выполняться на вычислительном комплексе. К сожалению, современные технологии еще не позволяют реализовать такое полностью автоматическое распараллеливание. Поэтому исследования были направлены в сторону разработки полуавтоматических систем, которые создают параллельную программу, опираясь на информацию о задаче, полученную от программиста. Были созданы такие инструменты как Parawise [5], ДВОР[6] (оба генерируют код в моделях OpenMP и MPI) и САПФОР [7] (генерирует код в моделях DVM и OpenMP). Однако перечисленные системы являются экспериментальными и находятся в разработке.

В процессе разработки параллельных языков и автоматизированных систем распараллеливания возникла потребность в эффективных способах отладки получаемых программ. Параллельные алгоритмы обычно значительно сложнее последовательных вариантов решения тех же задач. Более того, параллельный код может содержать нетипичные для последовательной отладки ошибки, вызванные некорректным использованием функций, обеспечивающих параллелизм, или несинхронизированным взаимодействием процессов и/или нитей. Недетерминированное поведение параллельной программы также может усложнить повторное воспроизведение ошибки при локализации с помощью многократных запусков программы с отладчиком.

Ручные методы отладки, такие как пошаговое исследование процесса выполнения алгоритма в определенных точках останова или отладочная печать, часто не позволяют адекватно работать с реальными научными и инженерными программными комплексами, спроектированными на непрерывную работу в течении часов или даже дней параллельно в сотни потоков. Кроме того, ручную отладку очень трудно применить к программам, полученным методами автоматизированного распараллеливания, после которых код становится сложнее для понимания при ручном исследовании.

Поэтому исследования были направлены на поиск иных подходов к отладке. Результатом стало появление понятия автоматизированной отладки. Данные методики позволяют найти большинство ошибок, в том числе особые, специфичные для многопоточных программ, в автоматическом режиме с минимальным участием программиста.

Одним из используемых автоматизированных методов отладки является сравнительная отладка. Принцип работы данного метода заключается в сравнении процесса выполнения двух программ, посредством контроля значений переменных в определенных контрольных точках. Сравнение может проводиться как между параллельно запущенными программами, так и с использованием файлов трассы, в которые заносятся все необходимые данные об операциях и значениях переменных в контрольных точках. Сравнение производится с заданной точностью, и при выходе за её пределы, либо при нарушении порядка операций выдается соответствующая диагностическая информация.

Сравнительная отладка применяется в тех случаях, когда у пользователя есть две версии программы, одна из которых работает верно (эталонная), а другая, разрабатываемая – неверно. В качестве эталонного варианта обычно используют последовательную программу, и поэтому данный метод как нельзя лучше подходит для использования в процессе распараллеливания.

2. Сравнительная отладка в DVM-системе

Текущая реализация автоматизированного отладчика DVM-системы[8,9] использует файлы трасс для выполнения сравнительной отладки. Это позволяет исключить серьезные временные затраты на синхронизацию эталонной и проверяемой программ, но, с другой стороны требует довольно существенных объемов памяти на хранение трассировки операций.

Для расширения возможностей применения отладки к реальным научно-инженерным алгоритмам были реализованы режимы отладки, уменьшающие размер файла трассы без заметного ущерба покрытию операторов отлаживаемой программы[9]. Самыми значительными нововведениями были:

- метод интегральных характеристик массива, который позволял заменить трассировку обращений к элементам массива в параллельном цикле на подсчет и сравнение контрольной суммы значений массива после каждой итерации;
- метод граничных итераций, трассирующий только граничные витки параллельных циклов, но при этом практически не теряющий точность определения ошибки.

Первая оптимизация позволила сократить размер файла трассы в десятки тысяч раз (в таблице приведены размеры трасс для некоторых алгоритмов из

комплекса тестов для оценки производительности суперкомпьютеров NAS Parallel Benchmarks[10]), тогда как вторая более сильно сократила время работы отлаживаемой программы. В совокупности данные оптимизации смогли сделать процесс отладки более быстрым и относительно нетребовательным к ресурсам.

Таблица 1. Размер файлов трасс в разных режимах отладки

	Без оптимизаций	Режим контрольных сумм	Режим граничных итераций	Оба режима
NAS MG	1.63 Gb	53 Kb	1.18 Mb	119 Kb
NAS LU	2.31 Gb	71 Kb	17.3 Mb	300 Kb

Тем не менее, размер файла трассы все еще остается одним из главных сдерживающих факторов использования сравнительной отладки, так как значения из таблицы соответствуют небольшим тестовым наборам входных данных, и на порядки меньше трасс, получаемых при отладке реальных приложений.

3. Расширение возможностей сравнительной отладки DVM-программ

3.1. Сравнение одновременно работающих программ

Дальнейшее развитие DVM-системы ставит новые задачи перед сравнительной отладкой. Специфика работы параллельных вычислительных систем позволяет выполнять параллельную программу в течение значительного промежутка времени. Для подобных условий можно использовать иной алгоритм работы отладки — одновременный запуск эталонной и отлаживаемой программ и обмен информацией между ними в процессе выполнения.

Подобные методы были реализованы в большинстве существующих инструментов параллельной отладки, включая отладчики Guard[11] и Wizard, в описании которых впервые был использован термин "сравнительная отладка". Однако в данных инструментах методы отладки отличаются от используемых в DVM-системе: контрольные точки, в которых происходит сравнение значений переменных, задаются пользователем. Предлагаемое же расширение DVM-системы предполагает автоматический выбор точек сравнения значений, аналогично существующей реализации[8], но проверка корректности выполнения проводится не через файл трассы, а путем пересылок отдельных блоков трассировки по сети между двумя работающими программами.

Подобное расширение позволит сделать сравнительную отладку более гибкой, допуская как одновременный запуск эталонной и отлаживаемой программы на одной многопроцессорной машине, так и удаленную отладку – для сравнения корректного выполнения программы на одной машине с

экспериментальной версией, запущенной на удаленном вычислительном комплексе и использующей иные средства компиляции.

3.2. Контроль результатов вещественных вычислений

Существенной проблемой, обнаруженной в ходе эксплуатации существующей реализации сравнительной отладки DVM-системы, стало различие в результатах операций над вещественными числами на разных машинах или при использовании различных средств компиляции. В таких условиях элементарная операция умножения или деления может вернуть результаты, отличающиеся в одном-двух младших десятичных знаках мантиссы. Поначалу это не сказывается на работе отладки и при сравнении чисел с погрешностью такие ошибки игнорируются отладчиком. Однако в процессе дальнейших вычислений расхождения накапливаются и распространяются на другие переменные, участвующие в арифметических операциях, и в итоге эталонная и отлаживаемая программы могут пойти по разным ветвям графа потока управления, что полностью исключит возможность дальнейшей отладки.

Для решения данной проблемы механизмам отладки DVM-системы требуется особый режим коррекции вещественных вычислений. Схожая возможность уже присутствует в текущей реализации сравнительной отладки – коррекция значений редукционных переменных. Разрабатываемый режим расширяет данный подход на все операции записи значений. При успешном сравнении полученного вещественного значения с вариантом из трассы, в переменную будет записано не вычисленное значение, а вариант из трассы, что исключит накопление расхождений и обеспечит синхронизированность выполнения эталонной и отлаживаемой программ.

Метод сравнения одновременно выполняющихся программ также способен поддерживать описанный режим работы, но для этого он должен быть модифицирован – а именно, выполнение очередного параллельного блока отлаживаемой программы не должно быть начато до момента получения от эталонной программы полной трассы для данного блока. Таким образом, эталонная программа будет выполняться с опережением в один блок.

4. Реализация новых возможностей отладки в DVM-системе

Текущая реализация DVM-системы реализует частный случай сравнения параллельно выполняющихся программ. В рамках выполнения программного комплекса на графическом ускорителе может быть включена опциональная возможность проверки корректности работы параллельных циклов.

В этом случае в процессе компиляции DVM-программы создается два варианта цикла – для выполнения на CPU и на GPU. Далее, во время работы программы оба варианта цикла получают одинаковый набор входных данных и

выполняются одновременно, после чего сравниваются значения переменных и массивов, изменяющихся в цикле.

5. Заключение

В данной работе обсуждается метод сравнительной отладки параллельных программ, а также реализация данного метода в DVM-системе.

Обозначаются направления дальнейшего развития и совершенствования механизмов сравнительной отладки DVM-программ. Новые возможности позволят решить проблемы, проявившиеся в процессе активного использования отладки при работе над распараллеливанием реальных научно-инженерных программных комплексов. Реализация кардинально нового подхода – обмена отладочной информацией между двумя одновременно выполняющимися вариантами программы, сделает процесс отладки более гибким и гораздо менее требовательным к памяти вычислительного комплекса.

В качестве первого шага к расширению функционала метода отладки описывается процесс одновременного выполнения параллельного кода на CPU и GPU с последующим сравнением результатов вычислений.

Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект проекты 16-07-01067 и 17-01-00820.

Литература

1. High Performance Fortran Forum. High Performance Fortran Language Specification, version 1.0. Technical Report CRPC-TR92225, Rice University, Center for Research on Parallel Computation, Houston, Tex., 1993.
2. Коновалов Н.А., Крюков В.А., Михайлов С.Н., Погребцов А.А. Fortran DVM – язык разработки мобильных параллельных программ // Программирование, М.: Изд-во РАН, 1995, № 1, С. 49-54
3. Коновалов Н.А., Крюков В.А., Сазанов Ю.Л. C-DVM – язык разработки мобильных параллельных программ // Программирование, М.: Изд-во РАН, 1999, № 1, С. 54-65
4. Бахтин В.А., Клинов М.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н., Сазанов Ю.Л. Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами. – Вестник Южно-Уральского государственного университета, серия "Математическое моделирование и программирование", №18 (277), выпуск 12 – Челябинск: Издательский центр ЮУрГУ, 2012. С. 82-92
5. ParaWise – Widening Accessibility to Efficient and Scalable Parallel Code. // Parallel Software Products White Paper WP-2004-01, 2004
6. Юрушкин М.В., Петренко В.В., Штейнберг Б.Я., Алымова Е.В., Абрамов А.А., Баглий А.П., Гуда С.А., Дубров Д.В., Кравченко Е.Н., Морылев Р.И., Нис З.Я., Полуян С.В., Скиба И.С., Шаповалов В.Н.,

Штейнберг О.Б., Штейнберг Р.Б. Диалоговый высокоуровневый оптимизирующий распараллеливатель (ДВОР) // Труды Международной суперкомпьютерной конференции “Научный сервис в сети Интернет: суперкомпьютерные центры и задачи”, Новороссийск, сентябрь 2010 г., – М.: Изд-во МГУ, 2010. С. 71-75.

7. Бахтин В.А., Бородич И.Г., Катаев Н.А., Клинов М.С. Ковалева Н.В., Крюков В.А., Поддерюгина Н.В. Диалог с программистом в системе автоматизации распараллеливания САПФОР // Вестник Нижегородского университета им. Н.И. Лобачевского. – Н. Новгород: Изд-во ННГУ, № 5 (2), 2012. С. 242– 245.
8. Крюков В.А., Удовиченко Р.В. Отладка DVM-программ // Программирование, 2001, № 3. С. 19-29.
9. Крюков В.А., Кудрявцев М.В. Автоматизация отладки параллельных программ // Вычислительные методы и программирование, 2006, Т.7, вып. 4. С. 102-110.
10. NAS Parallel Benchmarks, URL: http://www.nas.nasa.gov/publications/npb.html
11. Sosic, R. and Abramson, D. A. "Guard: A Relative Debugger", Griffith University Technical Report, School of Computing and Information Technology, Report # CIT- 94-21